Counterattack

Turning the tables on exploitation attempts from tools like Metasploit

Matthew “scriptjunkie” Weeks

Abstract

In hostile networks, most security is aimed at preventing compromise. However, it is often possible for the intended victim to not only confuse and frustrate the attacker, but actually trade places and exploit the attacker. Vulnerabilities have already been shown in security tools such as Firesheep, Nessus, Cain and Abel, and Wireshark. This paper introduces vulnerabilities in Metasploit discovered through code analysis that result from assumptions about threat vectors. These include XSS and CSRF vulnerabilities in the msfweb interface that enable remote code execution from a malicious URL. The meterpreter controller has had directory traversal vulnerabilities, as well as the TFTP and FTP server modules and many scripts. Cross-platform payloads can be written in Ruby, rely on the Metasploit libraries, and take advantage of cross-platform code locations in Metasploit for persistence. Metasploit users can protect themselves from these attacks by using securely coded scripts or virtualization and other defense in depth techniques. Despite these releases, Metasploit still has one of the best track records from a vulnerability perspective; it is coded in a memory-safe language and thus avoids a constant stream of memory corruption vulnerabilities that other programs suffer from, and its source is open to critique.

Introduction

Research on counterattacking, also known as aggressive self-defense, active defense or strike-back, has taken place for many years. Ranging from passive approaches to full remote exploitation, counterattacks have nevertheless often been considered ethically dubious or technically impractical. Although this paper will not discuss the ethical or legal issues behind counterattacks, it will show how launching attacks often exposes the attacker to intrusion. This paper will first present previous work followed by a description of attacks against specific tools and finally previously unreleased attacks against Metasploit.

Previous work

Popular tools to block, deceive, or merely gain information about attackers include the ubiquitous intrusion detection and firewall systems. Other passive approaches led to the development of honeypots and honeynets. Many antivirus firms and other researchers have run large honeynets to collect malware and attack signatures. Active defenses to exploit vulnerabilities of attack tools have also been demonstrated before. Weaknesses in the popular Linux security distribution Backtrack have led to a number of attacks, such as the sequence described by Rob DeGulielmo in his DefCon 17 presentation. Many security researchers have used vulnerabilities in exploit packs to gain information on attacks, botnets, and even the botmasters themselves. For example, work done by Paul Royal targeted operators of the exploit kits LuckySploit and UniquePack through XSS vulnerabilities in those kits. Billy (BK) Rios has demonstrated a counterattack against a Zeus botnet controller.

Generic counterattacks

Some techniques for counterattack are independent of the particular tool used. For example, if a honeypot hosted a Windows network share, an attacker that connected to the share would transmit information in the connection. An attacker using Windows would reveal in the SMB request his system name, domain name, and username. A counterattacker can force or at least prompt such a connection by embedding a link with a UNC path into a honeypot web page. Especially effective when the attacker is using Windows, the counterattacker can run the Metasploit smb_capture or smb_relay modules to get a password hash of the attacker to crack the attacker's password or even take over the attacker's system.

In some situations, a counterattack can successfully mirror the initial attack. For example, a network security administrator may want to stage a counterattack against a worm infected system that is launching exploits to spread the worm. In a targeted attack, the attacker will probably not allow his system to be vulnerable to the exploit he is using, however in a worm outbreak, an infected system will probably be vulnerable, since it was infected by the worm in the first place. In this situation, the counterattacker will need the ability to use the worm's exploit. For example, against a worm spreading using the MS06-040 vulnerability, a counterattacker could use Metasploit to exploit the same vulnerability the worm initially used to gain control of the offending system and shut down the worm.

Security tools

Many of the most widely used security tools have had vulnerabilities opening them up to attack. Examples can be found in Nmap, Firesheep, Nessus, Cain & Abel, Wireshark, and Metasploit.

Nmap

Nmap, possibly the most popular network exploration tool or port scanner, is a classic tool of both network administrators and hackers alike. Hackers commonly run an Nmap scan against a host to gain information about operating system and running services before they can prepare an attack. The Nmap developers have a good track record with no vulnerabilities reported in the NVD (National Vulnerability Database) or OSVDB (Open Source Vulnerability Database) despite the fact that Nmap is a large, popular network program written in C++.

Counterattacks against Nmap without a 0-day will be limited to deception and denial of service. One option is to use port knocking or similar mechanisms to hide a port so that Nmap does not detect it. Another option is to run a tar pit to slow down Nmap scans once detected. A tar pit is a firewall setting that will accept a connection, then advertise a zero-byte TCP receive window. A tar pit will have little effect on the basic SYN scan other than making the port appear open when it has nothing meaningful running on it. However, when Nmap or other scanners open a connection to run version detection scripts, the connection will be frozen in an endless series of keepalive packets. The attacker will only be wasting time and system resources until he gives up or the connection times out. An aggressive scan that usually completes in seconds or a few minutes may instead take hours to complete and will have almost no useful information.

A counterattacker can also make large numbers of ports appear to be open. If every port is open or if most of them are tar pits, an Nmap scan will consume a large amount of time and resources on an attacker's system, and it may crash Nmap. The current development tree of Nmap has a patch for this crash, limiting the number of concurrently running scripts, but it had not been incorporated into a stable release when this paper was written.

Firesheep

Firesheep is a recently developed specialized Firefox addon to perform session hijacking by sniffing cookies of specific websites, such as Facebook. Many previous tools have been able to do this, but Firesheep's complete ease of use and wide press coverage have pushed it to the forefront. Other tools are completely passive, but when Firesheep sees a new cookie, which generally represents a new logged-on user, it will test that cookie. Firesheep makes requests to Facebook to see whether the token is a valid session ID and extract the picture and name of the user it can hijack. It is therefore possible to detect a running Firesheep instance on the same open or WEP encrypted wireless network by sending out a request with a bogus cookie and listening for further requests with the same cookie. If another system sends out such a request, it can be identified as attempting to hijack sessions, probably with Firesheep. The Blacksheep Firefox addon performs this. Another program, Fireshepherd, sends out a request with a long string of random data as the cookie, which will cause Firesheep to crash.

Nessus

Nessus is a commercial vulnerability scanner that is one of the most widely used security tools. It can run thousands of tests to identify vulnerabilities from the network. Although it has a good track record, it also has suffered a number of vulnerabilities, such as CVE-2010-2914, XSS in the Nessus Web Server, or CVE-2007-4061, directory traversal.

Cain and Abel

Cain and Abel is a popular windows freeware tool that performs many attacks on “security aspects/weakness present in protocol's standards, authentication methods and caching mechanisms” that include many different password recovery techniques, ARP poison routing, and service manipulation. It is a closed-source application and has suffered a number of vulnerabilities, for example CVE-2005-0807: Multiple buffer overflows in Cain & Abel before 2.67 allow remote attackers to cause a denial of service (application crash) and possibly execute arbitrary code via (1) an IKE packet with a large ID field that is not properly handled by the PSK sniffer filter, (2) the HTTP sniffer filter, or the (3) POP3, (4) SMTP, (5) IMAP, (6) NNTP, or (7) TDS sniffer filters; and more recently CVE-2008-5405: Stack-based buffer overflow in the RDP protocol password decoder in Cain & Abel 4.9.23 and 4.9.24, and possibly earlier, allows remote attackers to execute arbitrary code via an RDP file containing a long string. A counterattacker could send exploits for these vulnerabilities at an attacker who had Cain & Abel's sniffer turned on.

Wireshark

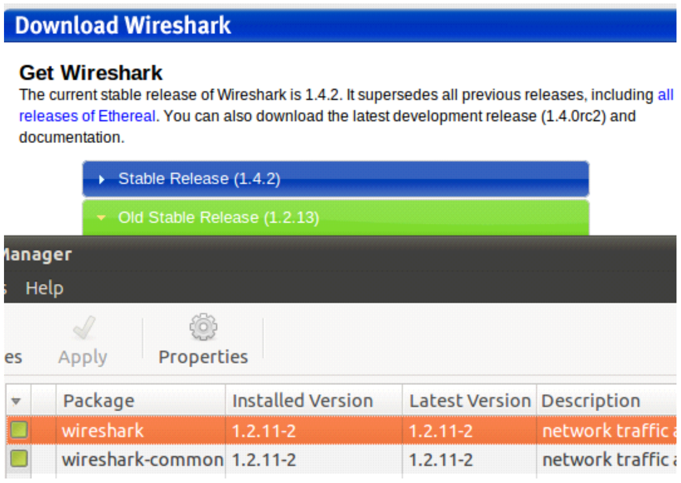
Wireshark is a packet-sniffing protocol analyzer that is useful for debugging network issues, but can also be used to sniff and view network traffic of others, listening to VoIP calls, and viewing passwords or any other sensitive content that is not properly encrypted. Wireshark has had a rough time with vulnerabilities. Dozens of security flaws have been documented. To be fair, it has faced possibly the most difficult job of the listed programs, as the Wireshark Security Wiki explains:

> Recent automated code inspections showed a much lower defect rate compared to other known open source programs (the defect rate of closed source programs is not known but may be even higher). Unfortunately most bugs found in the Wireshark code are security related so they are mentioned in the security bulletins.
>
> In most programs, only small sections of code work directly with "outside" data (e.g. from a file or network). By focusing on these small sections during code reviews, developers can eliminate most security problems.
>
> Wireshark is different. The vast majority of its code base deals directly with data from the "outside", so a code review on the relevant parts would cover most if not all of the complete Wireshark code. Running "wc -l epan/dissectors/*.[ch]" returns over 1,900,000 lines of code that's expected to handle fresh-off-the-wire data! Auditing all of this would be a **huge** effort, and may not guarantee success.
>
> Wireshark is implemented in ANSI C, which is vulnerable to security problems like

buffer overflows (compared to more securely designed languages like Java or C#).

The Wireshark team has taken security vulnerabilities very seriously, and responded by implementing many improvements. One of the improvements is an automated build process that includes fuzz testing after each checkin. However, the system can have negative effects, as it broadcasts security issues before they are fixed. When a fuzz test causes a crash, a file containing the packets that caused the crash are posted online, emailed to a public distribution list, and can serve as a proof of concept for an attack. In contrast, other popular open source projects like Firefox will mask files that can serve as the basis for an attack at least until a patch or a workaround is ready.

Another issue with Wireshark in practice is the difficulty of keeping Wireshark updated. Wireshark does not automatically update or check for updates, and installs generally become outdated in a matter of weeks or months. Furthermore, on Linux, when Wireshark is installed via a distribution software center, so that it can be automatically updated, the distribution often does not have the latest version. For example, Ubuntu, the most popular Linux distribution, generally lags behind on updates.



The current stable release of Wireshark is 1.4.2. It supersedes all previous releases, including all releases of Ethereal. You can also download the latest development release (1.4.0rc2) and documentation.

	Package	Installed Version	Latest Version	Description
🟩	wireshark	1.2.11-2	1.2.11-2	network traffic
🟩	wireshark-common	1.2.11-2	1.2.11-2	network traffic

Wireshark download page behind Ubuntu's Synaptic package manager showing an old, vulnerable version of Wireshark

The underlying problem, as admitted, is the use of a non-memory safe language like C that makes these attacks possible. If Wireshark were programmed in a memory-safe language such as, for example, Java, it would not suffer from these memory corruption vulnerabilities. The real elephant in the room of application security is that a huge majority of the vulnerabilities of not only Wireshark, but also more commonly exploited applications such as browsers and PDF viewers, could have been eliminated with the use of a memory-safe language.

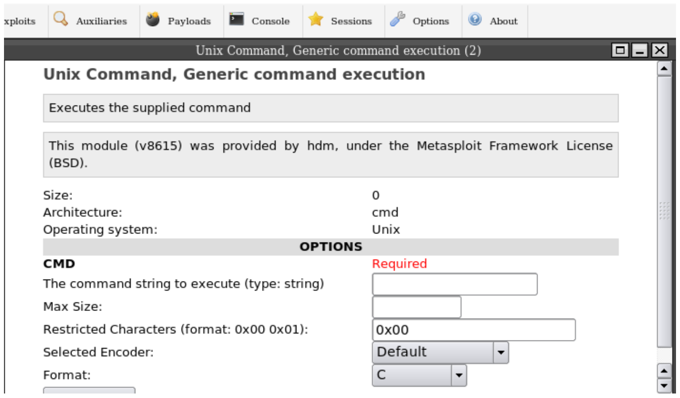
Metasploit

Metasploit was programmed in Ruby, which does not suffer from the memory corruption issues of C. It exposes very little memory corruption attack surface, and almost none of it lies in the actual Metasploit code, residing instead in libraries like OpenSSL. Vulnerability discovery is therefore unlikely with standard fuzzers. Instead, finding vulnerabilities requires manual code analysis, although a counterattacker is not completely on his own. For example, web vulnerability scanners can be used. (although none were used in this research) The vulnerabilities released in this paper are examples of common vulnerability classes found against many other applications, especially webapps.

Msfweb vulnerabilities

The first two vulnerabilities released here are found in the msfweb interface. This interface, the open-source web interface, has been included in some form in Metasploit since about 2005. However, in late 2009 support for the interface was dropped.

The first vulnerability is a cross-site scripting vulnerability. Msfweb provides a payload generating capability that will generate payloads based on options that a user provides.



Unix Command, Generic command execution (2)

Unix Command, Generic command execution

Executes the supplied command

This module (v8615) was provided by hdm, under the Metasploit Framework License (BSD).

Size:	0
Architecture:	cmd
Operating system:	Unix
OPTIONS	
CMD	Required
The command string to execute (type: string)	
Max Size:	
Restricted Characters (format: 0x00 0x01):	0x00
Selected Encoder:	Default
Format:	C

The msfweb payload generation page

After the form is submitted, msfweb will generate, encode, and display the payload in the format given inside a HTML textarea tag. Although the only options for format under the drop box are C, Ruby, Perl, Javascript, and Java, the msfweb server also accepts raw as a format parameter. A counterattacker can reflect any content into the output page from the CMD option of the generic command execution payload combined with the generic/none encoder and raw output format. Cross-site scripting can be obtained by sending a closing textarea tag followed by a script tag; i.e. “</textarea><script>alert(1)</script>”. The parsing logic will not echo content following a comma or equals sign or a semicolon. So to execute useful code, this exploit concatenates String.fromCharCode calls to decode and execute a stager that downloads and executes the main Javascript stage from the attacker's server.

This XSS vulnerability provides control over the msfweb instance. To turn this control into remote code execution, an exploit can use a command injection technique or a console-based technique that will execute Ruby code from within the msfweb process. The exploit presented here uses the second. The main stage will first obtain a new console by making a request to /console/. This request will create a new console, give it a number, and redirect to the URL /console/index/N where N is the index of the new console; for example /console/index/3. The script will then issue console commands to run a handler for a bind meterpreter, verifying that these commands complete correctly. Once the meterpreter session is established, it will download a file to a temporary directory and use the loadpath command to execute the Ruby code in the file.

This code execution technique may seem overly roundabout, but it functions cross-platform in Linux and Windows systems without relying on other system commands or functions. It will also work even when the attacker is behind a firewall and when the attacker does not have privileges to run a TFTP server.

The second vulnerability in msfweb is CSRF. Msfweb does not verify that requests originated from itself via any kind of CSRF token or other check. If a counterattacker can convince the attacker to view a page, he can force the attacker to make any request to the msfweb instance. Even though the process of launching an exploit or other module from the user's perspective is a multi-step approach of sending options to a series of forms, until the Launch Exploit (or Launch Auxiliary) command is given, the server is stateless. Each form post includes all options already specified. The final form post to launch the module holds all the information needed to launch the module. Furthermore, msfweb accepts parameters in either the POST body or the URL. Therefore a single CSRF request can launch a module, and it can do that without a form, only using an image tag easily embedded in many different places, such as web forums and emails.

A number of Metasploit modules call on external utilities to gain information or interact with other tools and place options that the user provides into the command line. Examples include the wifi fuzzers and the sqlmap auxiliary module. The sqlmap module simply creates a command line to launch the sqlmap program and runs it through a shell. It allows the user to add arbitrary arguments, which it simply appends to the command. This allows the module to run any other commands by inserting a pipe (|) ampersand (&) or semicolon (;) before the other command. In the exploit released here, a CSRF request launches the sqlmap module to create and run an arbitrary command line following a semicolon. To be effective, a counterattacker must use a command that will function on the attacker's OS.

Directory Traversal Vulnerabilities

Many scripts in the framework extract various items of information from the exploited host and save it in a subdirectory of the user's home directory. To organize the data across different hosts, the directory name often includes the computer name. Unfortunately, until recently, these were not checked for directory traversal sequences. Therefore, if an attacker ran one of these scripts, the counterattacker could control both the location and, depending on the script, the contents of a file to be written to the attacker's machine by using a computer name including sequences like "../../". Many scripts perform or can perform this type of logging, such as arp_scanner, domain_list_gen, dumplinks, enum_chrome, enum_firefox, event_manager, get_filezilla_creds, get_pidgin_creds, packetrecorder, persistence, search_dwld, and winenum.

Payloads

Many options exist for payloads on a Metasploit exploit, although care must be taken if the OS and architecture of an attacker cannot be reliably determined. Metasploit runs on everything from most Windows versions to Linux, OS X, and the iPhone. A counterattacker can only assume the basic dependencies of Metasploit and Metasploit itself are present. Knowing this, counterattackers can write cross-platform payloads in Ruby. Alternatively, after gaining code execution, shellcode can extract native or Java payloads depending on the target OS and architecture. Metasploit already generates Ruby bind and reverse shells for command injection bugs, but these tend to be unreliable as they are optimized for a short command line. For cases where an exploit can run larger Ruby shellcode, more reliable shells have been provided. In addition to catching errors, and re-listening for connections in case a shell process dies, the payloads will run in a new thread so that the exploited process does not freeze waiting for it to complete.

Counterattackers can obtain persistent access to an attacker in any OS by placing code in a subfolder of the .msf3/modules/ subdirectory of the user's home directory that will be automatically loaded when the framework starts. Metasploit will load all Ruby code inside the subfolders named exploits, auxiliary, encoders, nops, and payloads to support personally developed modules without affecting the main module tree. Another place persistence can be achieved is by placing commands or Ruby code in the file ~/.msf3/msfconsole.rc which Metasploit runs as a resource file when the framework is started. If the exploited process has privileges to change the main Metasploit code, it can add a back door, or even relocate the subversion root if the counterattacker wants to maintain access by subverting the update process. And of course, counterattackers can use any platform-specific mechanism to maintain access.

Payloads for use against Wireshark encounter some of the same difficulties as payloads for use against Metasploit. Wireshark compiles and runs on many different operating sytems and architectures. Cross-platform exploits are extremely difficult to create due to differences in system calls, and even across different versions of the same OS, structures like heap structures differ widely as well as general memory layout.

Defenses

The preferred method of defending against any of the exploits discussed in this paper is to patch the vulnerabilities discussed. Of course 0-day vulnerabilities may still remain. Using a dedicated laptop or VM for penetration testing tasks provides a degree of defense in depth, however some tools such as wifi injection modules may not function from a VM. Likewise, limiting privileges can also be effective although running client-side exploits may require more privileges. For example, Metasploit can run HTTP, DNS, DHCP, FTP, TFTP, SMB, and RPC servers, but most of these modules require root privileges on Linux or UNIX systems to open privileged ports. Running most Nmap scans also requires root privileges. A counterattacker should isolate vulnerable code used in counterattacks and attackers' sessions in dedicated honeypot VMs, but the preferred method is using bogus services and emulated meterpreter sessions in honeypots to avoid giving a real shell to an attacker.

Conclusion

Counterattacks to deceive, crash, exploit, or just get information on attackers have been created for use against many different attacks and tools, such as exploit packs, Nmap, Firesheep, Nessus, and now Metasploit. Tools that are coded in a memory-safe language have some of the best track records from a vulnerability perspective, avoiding a the memory corruption vulnerabilities that other programs suffer from. Tools like Metasploit also benefit from being open-source. Although, as the example of Wireshark shows, open-source does not imply error-free, it provides an atmosphere more conducive to external aid in finding and fixing vulnerabilities. Metasploit's large library set allows a counterattacker to create reliable, cross-platform payloads and maintain persistent access on the disk.

Works Consulted

Aggressive Network Self-Defense. Neil Wyler, Bruce Potter, Chris Hurley. 26 February 2005.

A very large malware honeynet. Panda Research Blog. 19 December 2006. Accessed 10 December 2010. http://research.pandasecurity.com/a-very-large-malware-honeynet/

Con Kung-Fu – Defending Yourself @ DefCon. Rob DeGulielmo . 30 July 2009.

National Vulnerability Database. NIST. Accessed 8 December 2010. http://web.nvd.nist.gov/view/vuln/search

Nmap Changelog. David Fifield. Accessed 11 December 2010. http://nmap.org/changelog.html

One-in-four hackers runs Opera to ward off other criminals - Security firm bamboozles hacker toolkit operators into divulging info. Gregg Keizer. 20 August 2009. http://www.computerworld.com/s/article/9136920/One_in_four_hackers_runs_Opera_to_ward_off_other_criminals

Oxid.it – Cain & Abel. Massimiliano Montoro. Accessed 15 December 2010. http://www.oxid.it/cain.html

Security – The Wireshark Wiki. Accessed 8 December 2010. http://wiki.wireshark.org/Security

Turning the Tables. Billy (BK) Rios. Accessed 2 January 2011. http://xs-sniper.com/blog/2010/09/27/turning-the-tables/

Open Source Vulnerability Database. Accessed December 2010. http://osvdb.org/