Konstantinos Kokkinis, 123RF

A guided tour to someone else's network

BREAKING IN

You need to think like an attacker to keep your network safe. We asked security columnist Kurt Seifried for an inside look at the art of intrusion.

BY KURT SEIFRIED

In June 2009, a virtualization product aimed at web servers was found to have a few security vulnerabilities. The end result was that about 100,000 web sites got hacked and deleted at a number of different providers. (It's not clear how many were recovered.) Also in June, the security-related website astalavista.com was hacked, and a variety of files and databases, as well as the remote backups, were deleted. These were only the "big" hacks that were newsworthy, the actual number of websites and servers compromised is much higher.

The techniques for network attacks keep evolving. In this article, I take a look at some favorite strategies for the latest generation of intruders.

A Quick Legal Disclaimer

Please note that engaging in the kinds of activities described in this article can potentially get you into trouble, ranging from a stern talking to by your network administrator to a less-than-enjoyable, all-expenses-paid vacation courtesy of whichever law enforcement agency you manage to annoy the most. So why am I writing this? If you want to build and maintain secure systems, you need to

understand how to make them fail. If you want to buy a good lock, you either have to buy a bunch of locks and learn how they work or find someone who has [1]. My advice is to get a cheap quad-core machine with lots of RAM, put VirtualBox or VMware on it, and build systems and networks you can attack without disturbing anyone else.

A Brief History

Life used to be pretty simple. You had a server, and on it you ran a couple of services (mail, file, DNS, etc.). If users wanted an application, you installed it on their machines. If users wanted to edit or upload content remotely to the web, you gave them FTP access. Email was just text, PDF files didn't include JavaScript, and image files were just image files – they weren't executable content. To secure your network, you simply kept things up to date, firewalled access, and ran as many services as possible without root access.

Results **1** - **10** of about **86,700,000** for corporate phone directory. (**0.25** seconds)
Results **1** - **10** of about **18,600,000** for corporate phone listing. (**0.37** seconds)
Results **1** - **10** of about **46,600,000** for employee listing. (**0.30** seconds)
Results **1** - **10** of about **2,510,000** for executive biographies. (**0.26** seconds)
Results **1** - **10** of about **2,510,000** for executive biographies. (**0.26** seconds)

Figure 2: Output of some interesting Google searches.

On Brute Force Attacks

Some automated tools simply hammer away, attempting a variety of common exploits against any server they can connect to, giving up speed and sophistication for brute force. This often works because of the sheer number of web servers and applications and, more importantly, because of the number of out-of-date applications with well-known security flaws (witness Adobe taking several weeks to months to fix various vulnerabilities in their Reader product). Some studies put the percentage of abandoned web logs at 95%, and, if no one is updating them with content, the chances are that no one is updating them for security fixes [2].

Step 1: Reconnaissance

Strictly speaking, reconnaissance isn't always necessary, but learning about the company's network layout, organizational structure, and personnel can aid in other attacks. Finding the domain name for a company or organization usually just means sticking *.com* (or a top-level national domain) on the end of the company name, and if that doesn't work, just ask Google. Once you have the domain name, you can learn a lot about the company or organization with tools like *whois*. Intruders look for technical details, like where the company's domain name servers are located and whether or not the technical management of the company is competent. (If an organization can't set up DNS competently, chances are their security isn't very good.) A perfect example of this is *seifried.org*, which I host on a VPS server. Unfortunately, the control panel is so brain dead it sets up the DNS server to allow remote zone transfers. Therefore, the *dig -t axfr* command lets you download entire DNS zones, finding all the hosts in seconds (Figure 1).

```
# dig -t axfr seifried.org @208.109.127.68
; <<>> DiG 9.3.4-P1 <<>> -t axfr seifried.org @208.109.127.68
; (1 server found)
;; global options:  printcmd
seifried.org.           3600    IN      SOA     ns1.seifried.org.
hostmaster.seifried.org. 2009052602 3600 3600 1209600 3600
seifried.org.           3600    IN      A       208.109.127.68
seifried.org.           3600    IN      NS      ns1.seifried.org.
seifried.org.           3600    IN      NS      ns2.seifried.org.
seifried.org.           3600    IN      MX      10 aspmx.l.google.com.
seifried.org.           3600    IN      MX      20 alt1.aspmx.l.google.com.
seifried.org.           3600    IN      MX      20 alt2.aspmx.l.google.com.
ns1.seifried.org.       3600    IN      A       208.109.127.68
ns2.seifried.org.       3600    IN      A       208.109.127.68
www.seifried.org.       3600    IN      A       208.109.127.68
seifried.org.           3600    IN      SOA     ns1.seifried.org.
hostmaster.seifried.org. 2009052602 3600 3600 1209600 3600
```

Figure 1: Output of dig -t axfr for seifried.org.

If zone transfers aren't allowed, another way to find web hosts is to use the *site:* search keyword in Google, filtering out *www.example.org* and so on in the search terms. This, of course, can be combined with a variety of interesting search terms such as *user name:*, *login:*, *password:*, *reset password*, and so on to find login screens. Google (and other search engines) can also provide information such as corporate phone numbers, lists of executives, corporate structure diagrams, and employee listings (Figure 2).

So the attacker has figured out where you live (metaphorically or literally), now what happens?

Step 2: Load Balancers, IPSs, and Firewalls

One of the first problems attackers will often run into is sites using load balancers, firewalls, intrusion prevention systems (IPSs), and web application firewalls (WAFs). If you try attacking a site behind a load balancer, the first part of your attack might go to server A, and the second part of your attack ends up at server B, resulting in a failed attack. Likewise, if a site is using a firewall, IPS, or WAF, it might detect and block the attacks (assuming it works).

How can you detect these devices and bypass their protective measures? Load

balancers are generally not built or deployed with stealth in mind. If a site is using DNS to load balance, tools such as *dig* will show them easily. If you get more than one IP or the IP changes, they're probably using DNS-based load balancing. Alternatively, tools such as Nmap can identify load balancers by their TCP-IP fingerprints (since almost no two TCP-IP–capable devices behave the exact same way). Detecting Firewalls and WAFs is also simple; just send a well-known attack using a remote host or an anonymous proxy such as Tor and see if the connection is terminated or if future connection attempts are blocked. If you really want to learn how the professionals do this, check out the DojoSec presentation by Joseph McCray [3][4].

Bypassing the devices blocking entry to the network is certainly possible. In the case of a DNS-based load balancer, simply using the IP address rather than the DNS name in an attack will ensure that all the attacks go to the same system. Bypassing firewalls is also relatively trivial because pretty much all firewalls allow incoming email and web traffic. The same goes for IPS and WAF systems; most sites are terrified of blocking legitimate web traffic and email, so they typically reduce the sensitivity of these systems, which reduces their effectiveness.

Web 2.0 and Modern Email

In the past, web pages and email were much the same: static text with minimal formatting and not much in the way of executable content. The pendulum has now swung the other way; most email clients support text and HTML, as well as file attachments. HTML, as you know, supports any number of executable technologies, the most popular being JavaScript. Almost everything now supports JavaScript (web browsers, email clients, even Adobe Reader), which means not only do you have to worry about buffer overflows and integer overflows in images, you often have a fully fledged Turing machine embedded in many applications. Because almost nobody blocks JavaScript or disables it, there is no better way to attack applications reliably.

Step 3: The Attack

A few common attack methods work really well against modern networks and users. The first is attacking exposed servers and services (like DNS), the second is attacking web servers (which are basically application servers now), and the last is attacking through email (which is also the de facto file sharing application for many people).

The first method is pretty well understood; generally speaking, the attacker will scan for vulnerable servers with a tool such as Nmap [5][6] or Nessus [7] and then attack them using exploit code or toolkits like Metasploit [8]. Exploiting these vulnerabilities will generally allow the attacker to run hostile code, like a root shell, on the machine.

Finding All the Attacks

So how do you track down all these individual attacks? Given a specific software package (e.g., Sendmail, WordPress, DokuWiki, or MediaWiki), how do you track down the vulnerabilities affecting it? Your best bets are to check out the CVE [9] and OSVDB [10] databases, which have links to resources in each security report, and, for exploit code, Milw0rm [11] (Figure 3) and PacketStorm Security [12] (Figure 4). The Metasploit framework actually includes surprisingly few exploits – around 300 at last count. PacketStorm Security carries about 300–400 exploits a month. Chances are that if the site is running out-of-date software, you can find something on Milw0rm or PacketStorm Security that will let you attack it, and if not, the CVE and OSVDB databases often contain enough information to point you in the right direction.

Attacking Web Servers

Web servers are basically application servers now, and where you have applications, you have security flaws. One of the biggest problems is the complexity of these programs. At a minimum, a "basic" application will often include: the application itself, a web server, an operating system, and a back-end database. All of these components can be attacked through flaws in the application, and in many cases, a number of small flaws can be combined to allow for code execution that lets an attacker onto the server.

If you're feeling lazy, you can also just download a web application scanner and point it at your target. Automated tools

Figure 3: MilwOrm.com - search results for SQL injection attacks.

such as Nessus or like Nikto, which looks for more than 3,500 potentially dangerous files and CGI scripts, can scan a server for vulnerable applications. If these tools don't find anything with known vulnerabilities, the attacker can always use tools like WebScarab to examine and attack web applications directly. Poking around randomly often exposes interesting problems faster than you would think [13].

/// File Name:	sugarcrm-exec.txt
Description:	SugarCRM versions 5.2.0e and below suffer from a remote code execution vulnerability.
Author:	Antonio Parata,Francesco Ongaro,Giovanni Pellerano
Homepage:	http://www.ush.it/
File Size:	7103
Last Modified:	Jun 15 16:04:40 2009
MD5 Checksum:	f81ce65d75a4b29de7ebbf23b6cb8179

Figure 4: PacketStorm Security - description of exploit code.

SQL Injection

Most web applications need to store data locally on the server, and rather than dealing with local files or SQLite, applications commonly use a back-end database such as MySQL or PostgreSQL. Unfortunately, many applications do not safely sanitize data before passing it to an SQL query, and many do not construct queries safely. Some common attacks include:

```
name' or '1'='1
productid' and '1'='2
```

The first attack is commonly used against authentication fields (e.g., username or password); if not properly sanitized the query will look for *name* or *'1' = '1'*; because 1 = 1 evaluates to *TRUE* in SQL, this would potentially let the attacker authenticate even without a valid password or user name. The second attack will help the intruder find out if a query is vulnerable to SQL injection; it will look for *productid*, but because 1 does not equal 2 (and thus the statement evaluates as *FALSE*), the *and* statement will force the SQL query to be false, which will most likely result in an error. In the case of a product page that should display the results for *productid*, it won't because the SQL query is invalid or causes an error.

Attackers love SQL injection because they can bypass authentication altogether, and depending on the database and configuration in question, it can also let them attack the local database server directly. Additionally, if the web application gets content from the database (such as product descriptions), an attacker can modify the product description to include PHP code (or whatever language the web application is built in), which is then executed the next time that page is requested, allowing an attacker to execute code on the server. For a really complete list of tips and tricks, check out the "SQL Injection Cheat Sheet" [14]. For a great SQL injection tool, check out SQLMap (now available as an official Debian package!) [15].

Cross-Site Scripting (XSS)

Cross-site scripting is a perfect example of why one should never mix data and code in a single object. HTML was once just a simple mark-up language that didn't involve any content executed on the client side. Of course, this made for a really boring web, and it wasn't too long before we got JavaScript from Sun and a variety of options from Microsoft (ActiveScript, ActiveX, etc.). Now the problem is that the web browser has no way to know what JavaScript should and shouldn't be present in a web page (and whether or not the JavaScript is "bad"), so an attacker who can insert content can just as easily insert JavaScript, which (usually) is executed on the client side, unless the client user is paranoid and has disabled JavaScript or installed the NoScript extension.

The real beauty of XSS attacks is that they are extremely common. You can put hostile content and code into the website the user is using, allowing you to steal cookies (which usually contain authentication credentials), keystrokes, form data, and so on – all of which can ultimately lead to administrative access on the application and more detailed attacks. Also, when XSS is combined with targeted email attacks and personal information gleaned from the web, the chances of successfully tricking a user into clicking on a bad link go up. But what happens when the website has a good web application firewall (or perhaps *mod_security*), which blocks XSS attacks? In that case, you go with encoding (UTF-8), embedding white space (which the browser will generally ignore), and adding strange characters into the script tag (such as "/"), to name a few methods.

Cross-Site Request Forgery (CSRF)

I covered cross-site request forgery in my monthly column in December 2008 [16]. To recap briefly: If a web application doesn't ensure that commands sent to it are valid, an attacker can potentially trick a victim's web browser into making a request that contains commands that the web application will execute (such as adding a new user or changing the permissions on an existing user). The bad news is that many applications still do not provide effective CSRF protection, the simplest reason being that there really aren't any widely accepted libraries to deal with this problem; the fact that people are forced to keep reinventing the wheel has led to many broken implementations.

Http Parameter Pollution

Http Parameter Pollution (HPP) is a new attack technique that was publicly announced only a few months ago. HPP is so simple, I'm amazed no one caught it sooner. When an application submits data to a server in the form of parameters, the server might not handle the situation gracefully when a parameter occurs more than once. In other words, if you put *productid* into the *GET* string twice, you might blow up the application (Figure 5). I love this class of vulnerability for one simple reason: it involves interactions between web servers, which all appear to behave in strange and often unexpected ways, and web applications (your guess as to how they react to mangled parameters is as good as mine). As far as I can see, there is no "correct" way to handle multiple parameters. Good arguments can be made for only taking the

Tech/Web Server	Parsing result	Example
PHP/Apache	Last occurrence	par1=val2
mod_perl,libapreq2/Apache	First occurrence	par1=val1
Perl CGI/Apache	First occurrence	par1=val1
mod_perl,unknown lib/Apache	Becomes an array	ARRAY(foo)
For the URL parameters: ?par1=val1&par2=val2&par1=val3		

Figure 5: Http parameter pollution with the Apache web server.

first one, the last one, concatenating them together, and even passing them as an array instead of the expected string type. Therefore, it is unlikely that the world will see a good long-term solution [17].

Local File Inclusion

Most applications make use of include files (CSS files, libraries, etc.), and oftentimes they can be tricked into including files elsewhere on the system. Browsing the file system, especially remotely, gives an attacker detailed information, for example, by checking the */etc/* directory (for configuration files), the man page directories (for installed software and version information), and the binary and library directories. Because most web applications run when requested, any configuration information they need must be available to them; configuration files such as *wp-config.php*, *config.inc.php*, or *LocalSettings.php* allow attackers to retrieve the database credentials remotely.

```
From: big.boss@example.org

To: support.guy@example.org

Subject: Can't open the attached file - need it urgently!

Attachment: media-campaign.zip

The attached file is some kind of zipper file, they told
me I could unzip it but it doesn't work. The file is supposed
to be our new media campaign,    and I need to get it to the
printers ASAP. I need you to unzip the file and make sure
the file is ok and then copy it my network directory. I need
this done before lunch ends.

Signed:

Big Boss
```

Figure 6: An example email sent 15 minutes before lunch time.

Creating Files with MySQL

If an attacker is unable to execute code on the system but has access to the database (either through SQL injection or by harvesting credentials from a configuration file), what's the worst that can happen? Well assuming they don't simply wipe the database or delete large chunks of it, they can create files on the system by using the *INTO OUTFILE* SQL command. Fortunately, an attacker cannot overwrite existing files; otherwise, they'd be able to modify or destroy every database on the system!

However, with the use of *INTO OUTFILE* with custom-made database content, an attacker can create script files in the */tmp* directory – for example:

```
CREATE TABLE `database`.➥
`scripts` ➥
(`contents` TEXT NOT NULL) ➥
ENGINE = MYISAM
INSERT INTO `database`.➥
`scripts` (`contents`) VALUES ➥
('#!/bin/bash\necho ➥
"hello world"');
SELECT * INTO OUTFILE ➥
"/tmp/bad.sh" from `scripts` ➥
WHERE 1
```

An attacker can even create binary files by using the *DUMPFILE* command, letting the attack leverage local file inclusion bugs via image files or documents containing buffer overflows.

Email-Based Attacks

So what happens if you hit a dead end and can't find any vulnerable services to attack? What if the network is properly segmented and there is no path from the web server you have compromised to the internal network? Go with email. Because virtually all email clients now handle HTML, multimedia content, and so on, they rely on the underlying system libraries to parse this content.

The bad news is that virtually every HTML rendering engine (WebKit, Gecko, Microsoft HTML Rendering Engine, Microsoft Word, etc.) has exploitable flaws, and most image and multimedia files also have exploitable flaws. If you can sneak a malicious email past the scanners, you can probably cause code execution on the victim's machine.

To make things even easier, you also have the option of attaching a file that targets any number of local programs, currently the more popular ones are Adobe Reader (with many JBIG2-related vulnerabilities), Open Office, and of course, Microsoft Office.

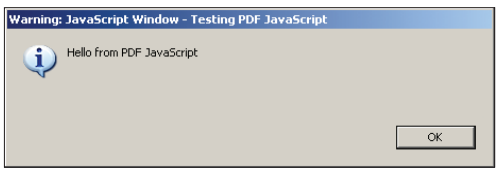

Figure 7: An example of a PDF file with a JavaScript pop-up message.

But don't all sites have virus scanning of incoming email and blocking of executable attachments? Well, this is where the information harvested about the target really comes in handy. If you can find a list of the executives, or a company phone directory (which will sometimes even lists the department someone is in), you can craft email messages that look something like the message shown in Figure 6.

Creating Malicious PDF Files

The only reason I am picking on PDFs and not some other file format (such as TIFF, AVI, DOC, and ODT) is that, in the last few months, a lot of easy-to-use tools and exploits for Adobe Reader have been released, and Reader is one of the few applications that is almost guaranteed to be on a system. (If it isn't there, the system probably has an equally vulnerable program, such as Foxit). Oh, and you can embed JavaScript into PDF files (Figure 7) that is executed by default, although you can disable JavaScript support in Acrobat Reader [18].

Didier Stevens has released a tool called *make-pdf-javascript.py* that allows you to embed arbitrary JavaScript into a PDF file [19]. Fortunately, this tool doesn't do any obfuscation or other tricks to hide the JavaScript, although other tools do. However, I will leave finding them as an exercise for the reader).

One note: You might have to run the script through dos2unix to fix the line breaks, and depending on your version of Python, there is a *finally:* clause in line 63 that you might need to remove. Just be sure to remove one tab from the line that follows as well and it will run fine.

Bringing It All Together for the Win

Individually, most of these attacks won't get you very far. You might gain access to a web application, read someone's email, or view a file on the server. But by combining techniques, such as writing arbitrary contents to a file and then including that file so that the PHP code within it is executed (Figure 8), an attacker can launch local attacks, of which there are plenty. In the first half of 2009 alone, the Linux kernel has suffered be-

cause of several locally exploitable vulnerabilities (*ptrace_attach*, *udev*, *netlink*, and *exit_notify*) for which exploit code exists publicly (just search Milw0rm for "Linux Kernel").

Exploiting a system via the kernel is particularly effective because a) you know it's installed and b) upgrading a Linux kernel on many web hosts is either a complete pain or simply not possible. Once attackers have the ability to exploit code locally, it's only a matter of time before they can execute code as the root user.

Step 4: What to Do Once You're In

So you've successfully compromised a host, executed a local attack, and gained root access. Now what? For most attackers, the answer is simple: Install a rootkit [20] to maintain access and then keep going. With access to internal systems (such as file servers), an attacker can create links to shared files, which on Windows, for example, will be executed with "Intranet" if it is within the same network, thus bypassing many of the security protections.

Even if the attacker only has access to a limited web server within your domain, the assailant will be able to attack the network infrastructure (such as routers and switches) directly and spoof email more easily. Alternatively, an attacker might simply use your systems as part of a botnet to attack other hosts and networks, send spam, and harvest personal information. The possibilities are endless. ■

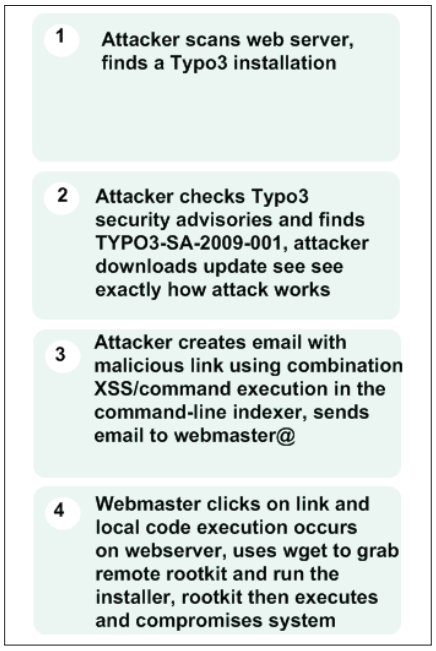

Figure 8: An example of a combination XSS/ remote code execution.

INFO

[1] "Ten Things Everyone Should Know About Lockpicking & Physical Security" by Deviant Ollam: *http://www.blackhat.com/presentations/bh-europe-08/Deviant_Ollam/Whitepaper/bh-eu-08-deviant_ollam-WP.pdf*

[2] "Blogs Falling in an Empty Forest": *http://www.nytimes.com/2009/06/07/fashion/07blogs.html*

[3] DojoSec: *http://www.dojosec.com/*

[4] DojoSec monthly briefings, April 2009, Joseph McCray: *http://vimeo.com/4109188*

[5] "Sysadmin: Nmap Scripting" by Eric Amberg, *Linux Magazine*, February 2008, pg. 68

[6] "Sysadmin: Nmap Methods" by Christian Ney, *Linux Magazine*, January 2006, pg. 62

[7] Nessus: *http://nessus.org/nessus/*

[8] "Metasploit: How Hacking Got Easy" by Kurt Seifried, *Linux Magazine*, November 2008, pg. 62

[9] Common Vulnerabilities and Exposures: *http://cve.mitre.org/cve/*

[10] Open Source Vulnerability Database: *http://osvdb.org/*

[11] Milw0rm: *http://www.milw0rm.com/*

[12] PacketStorm Security: *http://packetstormsecurity.com/exploits100.html*

[13] Top 10 web vulnerability scanners: *http://sectools.org/web-scanners.html*

[14] SQL Injection Cheat Sheet: *http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/*

[15] SQLMap: *http://sqlmap.sourceforge.net/*

[16] "Attack of the CSRF" by Kurt Seifried, *Linux Magazine*, Febraury 2009, pg. 66

[17] Http parameter pollution: *http://www.owasp.org/images/b/ba/AppsecEU09_CarettoniDiPaola_v0.8.pdf*

[18] Disabling JavaScript in Adobe Reader: *http://blogs.adobe.com/psirt/2009/04/update_on_adobe_reader_issue.html*

[19] Didier Stevens' blog: "PDF Tools" *http://blog.didierstevens.com/programs/pdf-tools/*

[20] "Fourth-Generation Rootkits" by Kurt Seifried, *Linux Magazine*, December 2008, pg. 64