**NAME: Eric Somogyi – CSEC 488 – 11/17/2024**

# **Command and Control (Lab #9 - Bonus) - 75 Points**

## Listeners

1. **Include screenshots from above. Investigate one of the other listeners available. How does it differ from the http listener we used? Why might this be useful? (15 pts)**

**Answer:**

**The other listener I used was the listener dbx listener which stand for the Dropbox (DBX) protocol. It allows for command and control (C2) communication over the dropbox API. It basically establishes a C2 channel using Dropbox as the transport mechanism. In the options menu after running “uselistener dbx” instead of only setting the Port , Bind IP you set the Dropbox *Developer* account credentials which is done in the first category called API Token, which you get from being logged in your Dropbox account on Dropbox’s website. There is also no “set Port” option in the dbx listener like there is in the normal http listener.find There are steps you have to take from your Dropbox login from a browser first by creating an app. This Dropbox listener is extremely helpful because it makes it even harder to be detected by cybersecurity software. It is also useful because it can be used by different Dropbox account and once the C2 is running you can privilege escalation and move laterally within a network.**

**Source of data:** [**https://bc-security.gitbook.io/empire-wiki/listeners/dropbox#empire-setup**](https://bc-security.gitbook.io/empire-wiki/listeners/dropbox#empire-setup)

**See Screenshots below:**

**A screenshot of a computer

Description automatically generated**

**The 2nd screenshot is of my options. (it would not fit all on one screenshot)**

**A screenshot of a computer

Description automatically generated**

## Stagers

1. **Include screenshots from above. Investigate one of the other stagers available. How does it differ from the batch stager we used? Why might this be useful? (15 pts)**

**Answer:**

**I investigated the multi\_bash stager and this one is basically a self-deleting bash script written in python to execute the stage0 launcher. The stage0 launcher is the initial stage of the Empire payload. The payload connect to the Empire listener through HTTP/HTTPS and after its executed it downloads the script to get the script from the Empire server and then it deletes itself and exits to minimize forensic artifacts. It’s useful because if you want to employ an agent on a unix based system with minimal footprint. The script is open source on github found here:** [**https://github.com/BC-SECURITY/Empire/blob/main/empire/server/stagers/multi/bash.py**](https://github.com/BC-SECURITY/Empire/blob/main/empire/server/stagers/multi/bash.py) **.**

**See Screenshot below:**

**A screenshot of a computer

Description automatically generated**

## Agents

1. Run agents again to see your new agent checking in. Take a screenshot of your new agent.

**Answer:**

**See Screenshots below**

A screen shot of a computer

Description automatically generated

*A screen shot of a computer

Description automatically generated*

## Modules

1. Lets dump passwords using mimikatz. Issue the command usemodule powershell/credentials/mimikatz/logonpasswords\*. Then run the info command to learn more and options to see variables that need to be set. Notice that this module does not require any options to be set (the agent name is prepopulated). Execute the command.Was your module successful? Why do you think that is?

**Answer:**

**I ran the command execute after running options which just shows the current record. Info shows the settings that can’t be changed from default because this module is a preset for a specific purpose. The module was not successful, and I received the error “module needs to run in an elevated context”. Mimikatz needs administrative access to interact with system processes (exe) and those processes have the login credentials stored somewhere in the Window’s memory. It seems that being logged in as student there is not full escalated privileges available on the current Windows machine in this environment so then using a different module first like PowerShell/privilege/escalate first to get to that correct administrative privileged level and then running the MimiKatz/logon passwords module should be able to dump the credentials from the target hypothetically.**

1. **Include screenshots from above. Investigate one of the other modules available. What does it do and why might this be useful? (15 pts)**

**Answer:**

**I investigated the module called “/powershell\_credentials\_mimikatz\_command”. All this module does it starts the Mimikatz with a custom command. There are only two options which are True (required) and they are Agent and Command. So you are able to set the Agent (which is our “5D2VYLP4” from the .bat executed earlier) and then it invoked a command from the Empire client after you execute the module. You type set Command (which is the only other option besides Agent) and there are 7 options sekurlsa::logonpasswords, sekurlsa::minidump, privilege::debug sekurlsa :: logonpasswords, token::elevate, sekrulsa::krbtgt, lsadump::sam, and Kerberos::list. This module is very helpful in it automatically connects to the agent and launches the mimikatz command to what you set it as which results in a successful response for remotely performing this function through the Agent that was connected earlier.**

**See screenshots from above (there was no screenshot instruction in steps 1 and 2\*\*)**

**A screenshot of a computer program

Description automatically generated**

**A computer screen shot of a program

Description automatically generated**

1. **What does the sleep command do that makes it different from say a meterpreter payload? Why might this be useful? (1 pt)**

**Answer:**

**The sleep command as noted in the information response “Tasks specified agent to update delay (s) and jitter (0.0. -1.0). “ Also after a Google search, the sleep command in this context of adjusting the sleep time in powershell-empire client on a successfully connected agent sets the interval at which the target machine (where the .bat script was ran) will contact the attacker (in our case the linux machine) every X amount of seconds to check in. So this setting is set first before exporting the Agent script because this is written in the code. It is more stealthy basically. It is different from a meterpreter payload because these are more like an active session in real time so there is no delay for when commands are executed where as if there is a sleep delay seconds set in the agent, this is visible from the powershell-empire server (attacker) which is listening which is where you would see the history being executed.**

Lastly, figure out a way to get the mimikatz module to work (sekurlsa::logonpasswords) by using empire features to help you. Take a screenshot of successful execution.

1. **Include screenshots from above. Document the steps you took including a** screenshot **of any module options you used and a successful mimikatz output. (15 pts)**

**Answer:**

1. **First I executed the command “usemodule powershell\_credentials\_mimikatz\_command”.**
2. **I went to “help” to see the options for commands then used options to see what parameters have not been set. There were only 2, Agent and Command.**
3. **I then set Agent to the only option which was my “5D2VYLP4”**
4. **I then set Command to one of the 7 options which was “sekurlsa::logonpasswords”**
5. **Then I used the command “execute” and and it ran “Task 1”.**
6. **Then to prove my successful mimikatz output for the screenshot and to confirm my module was successful I went to my powershell-empire server tab to confirm that the “Task 1” ran and there in the screenshot below is the confirmation that “Agent 5D2VYLP4 got results” and “Agent 5D2VYLP4 returned results”**
7. **To further prove my steps above were SUCCESSFUL was I enterer the command “back” to go back into the agent domain. Then I went back to view options (I scrolled up) and saw relevant commands I could use to such as “jobs” and “history”.**
8. **Lastly, I executed “jobs” which confirms the Task was received . Then I executed “history” and this shows the mimikatz sekurlsa::logonpasswords results job.**

**See screenshots below:**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

**A computer screen shot of a program

Description automatically generated**