Is it easier to evade detection and blocking by controlling a Command and Control (C2) botnet using social media (SM) rather than a traditional C2 server?

Introduction

Malicious attacks are becoming increasingly sophisticated. Command and Control (C2) enables attackers to remotely control infected devices and is a critical component in many of these attacks. Malware connects to C2 servers to receive commands and payloads, or upload logs or stolen files. Because C2 detection techniques are also constantly advancing, attackers are looking for ways to make C2 communication undetectable and resilient. Increasingly, they are hiding C2 communications in plain sight, often on social media (SM) and other cloud-based public services. In this essay, I am going to explore several emerging trends in the use of SM for C2 communications, and discuss methods being developed to counteract these trends.

C2 Messages

Relatively little text is needed to control a botnet, the actual command strings will most likely not be over 20 characters. All URLs will likely get shortened using a URL shortener such as bitly.com, and to accompany the shortened URL will be a command, that will likely have a short acronym that has been pre-defined in the bot's code to reduce the length of the commands. This results in no more than 20 bytes of data. This will vary depending on the botnet, but in general, this will be the system.

Uses of C2 Botnets

DDoS

Distributed Denial of Service (DDoS) attacks are one of the main uses of botnets. A DoS attack is when a server is flooded with more requests than it can handle – each one often requiring a lot of processing on the server side. This will make the server unable to handle any other requests from legitimate users, effectively taking down the server. The server may crash, which would also stop legitimate users using its services as it wouldn't be able to respond to any requests.

This can be prevented with a firewall that blocks users that are sending excessive requests. This is where DDoS attacks are used. The first "D" stands for distributed. When the users that are attempting to DoS the server are from many locations around the globe, it becomes much harder to block every user. This results in a higher success rate of the attack.

Install Secondary Malware

A botnet may also be used to distribute malware. If an attacker wants to install their malware on lots of machines without creating a delivery mechanism, they can use a pre-existing botnet to distribute the malware to the infected computers. Often this will be spyware, because obvious malware will lead the user to look for other malware on their device, and potentially discover the bot code and delete it from their device.

Activate Malware

If another piece of malware was installed along with the bot, or secondary malware has been sent to the infected device, it may lie dormant until the botmaster activates it. This would be done through the C2 botnet in order to activate every device at the same time.

Traditional C2

Controlling Traditional C2 Botnets

Internet Relay Protocol

What used to be one of the most common ways of controlling a traditional C2 server is to use the IRC (Internet Relay Chat) . This uses a client-server model to transfer text between clients. A client connects to a server – which contains channels in which messages can be sent. When a client sends a messages to a channel, every client connected to the channel receives it. This is ideal for controlling a C2 botnet (Puri, 2021).

Once the bot is installed on the target machine, it will generate a random “nickname” needed to join the IRC channel, and then uses the hard-coded password to join the botmaster’s private IRC channel. Whenever the botmaster wants to send out a command to their bots, all they have to do is type a message in their private channel, and all the bots will receive it.

Domain Name System Queries

A common technique is to use DNS (Domain Name System) queries to transmit C2 commands. This approach is effective because it doesn’t create any outbound traffic from the infected computer. In this scenario, the botmaster would register a domain, and identify their C2 servers as the authoritative name servers for the domain. When the infected computer wants to check for commands, it sends a DNS query within that domain. The answer sent back to the bot is the command. This is typically obfuscated so that it isn’t obvious data is being sent to the device (Brenton, 2022).

Hypertext Transfer Protocol

HTTP (Hypertext Transfer Protocol) is the protocol used for displaying webpages. But it can also be used to control botnets. Unlike IRC where the bots stay connected to the server, the bots will periodically visit a web server to get commands.

Within the response from the web server, there is a section for the requested object from the server. This is where the malicious message will be hidden.

This is a very broad method, and it has many of the same benefits as using SM, as they both use the HTTP protocol.

Detecting Traditional C2 Botnets

Network Monitoring

One of the most successful ways to detect traditional C2 servers is by monitoring network traffic. They can be identified by mapping the connection frequency of IP pairs (one on the network, one external) over an extended period of time. The infected machine will be constantly checking in with the C2 server, and while this may be hard to recognise in a brief capture, by analysing the data over a longer period of time, it becomes clear that the two IP addresses are consistently communicating (Brenton, 2022).

The problem with using IRC, is that it is a very unusual protocol. It has fallen out of use since it was created in 1988, and so cyber security professionals often monitor networks for the IRC protocol, as it is known for being used to control botnets. It is not always used controlling botnets, and there are still chatrooms that are still used, but it is nevertheless a suspicious protocol to be detecting on a network.

HTTP is not a suspicious protocol, but what will alert network admins to the presence of a bot will be the sudden constant communication between a machine and a web server.

Detecting DNS C2 Servers

To ensure that a local DNS forwarder does not hand back cached answers, the domain names that the bots requests for have to constantly change. This means that on a network where an infected computer is present, the amount of unique queries will increase greatly. By tracking the number of unique queries per domain, network admins can quite quickly and accurately identify when an infected computer is present on the network.

Another method is to check for further requests being sent to the DNS query's responses. An infected computer will request a domain, and won't then use the resource sent back from the DNS server. The internal hosts will never access the queried resources. This is another very effective way to detect C2 botnets over DNS.

Taking down Traditional C2

The main issue with traditional C2 servers is that they are vulnerable once they are discovered – as was the case with the Mumblehard Linux botnet in 2015, when researchers "[took] over the IP address, sinkhole[d] the server, and in essence shut down the botnet" (Bisson, 2016). And also in 2012 when Hlux/Kelihos was taken down when researchers "successfully disabled the dangerous Hlux/Kelihos botnet by sinkholing" (Ortloff, 2012).

Sinkholing is when researchers or ISPs change DNS records for malicious servers. When they have discovered a malicious server (such as C2) they change the DNS records for the URL to point to their own server. This prevents communication between the bots and the botmaster, essentially shutting down the botnet (ENISA, 2022).

SM C2

Controlling SM C2 Botnets

Posting to SM will typically involve using the SM platform's API (Application Programming Interface) which allows the botmaster to write a script that has the ability to retrieve and create posts and comments on behalf of an account. This is relatively easy to do, and when discussing the difficulties of each method, the difficulties of implementing the API will be disregarded, as this will be required for every method.

Plain-text

Plain-text is used to control C2 botnets, an account known to the bots will post IP addresses or URLs on SM with an instruction for the bots. This is simple, effective, but is incredibly easy to detect. The vast majority of social media sites will have algorithms that will check each new post for suspicious activity, and this will flag up very quickly, and the account will be shut down. This is why botmasters have come up with methods to hide the instructions within posts.

This is incredibly easy to implement for a botmaster, as the plain-text commands can be preset, and all that is needed is a URL or IP address to post, and these commands can be passed straight to the API to get posted.

Base64/Hex encoding

A basic way of obfuscating the commands is to encode the text into a different base, most commonly base64 or hexadecimal. This would circumvent the most basic of algorithms used to check for suspicious SM posts/DMs but is not by any means a secure method. Kartaltepe, E.J. et al. created an algorithm that would check any SM post for suspicious content with very high success rates.

This would also be very suspicious to anyone viewing the account or its posts. No normal account would be posting base64 or hex-encoded text. This would likely get reported as spam or suspicious content and get removed relatively quickly.

This is quite an easy method to implement. Base64 and Hex encoding is built into most programming languages, and if not, there will be libraries and examples of code online that can be used very easily. This would only require one line of code to encode the text, and then that can be passed to the API.

If the botmaster wants to use an alternative encoding, they may have to write it themselves. However, this is not a large task, and would not require more than around 15 lines of code.

Image Steganography

Image steganography is a technique that involves editing the data within an image to hide data within it. This could include textual commands, URLs, or even binary data for a malicious program.

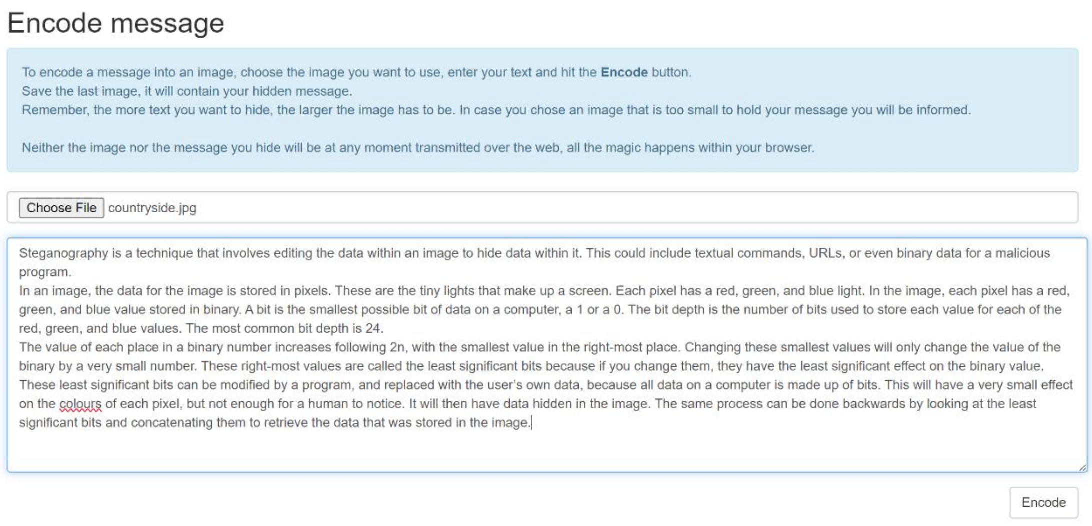
In an image, the data for the image is stored in pixels. These are the tiny lights that make up a screen. Each screen pixel has a red, green, and blue light. In the image, each pixel has a red, green, and blue value stored in binary. A bit is the smallest possible bit of data on a computer, a 1 or a 0. The bit depth is the number of bits used to store each value for each of the red, green, and blue values. The most common bit depth is 24.

The value of each place in a binary number increases following 2^n, with the smallest value in the right-most place. Changing these smallest values will only change the value of the binary by a very small number. These right-most values are called the least significant bits because if you change them, they have the least significant effect on the binary value.

These least significant bits can be modified by a program, and replaced with the user's own data, because all data on a computer is made up of bits. This will have a very small effect on the colours of each pixel, but not enough for a human to notice. It will then have data hidden in the image. The same process can be done backwards by looking at the least significant bits and concatenating them to retrieve the data that was stored in the image.

By using an online steganography tool (L, 2024) I hid this description of steganography in an image.

This is the text that I chose to hide in the image:


Encode message

To encode a message into an image, choose the image you want to use, enter your text and hit the **Encode** button.
Save the last image, it will contain your hidden message.
Remember, the more text you want to hide, the larger the image has to be. In case you chose an image that is too small to hold your message you will be informed.

Neither the image nor the message you hide will be at any moment transmitted over the web, all the magic happens within your browser.

Choose File countryside.jpg

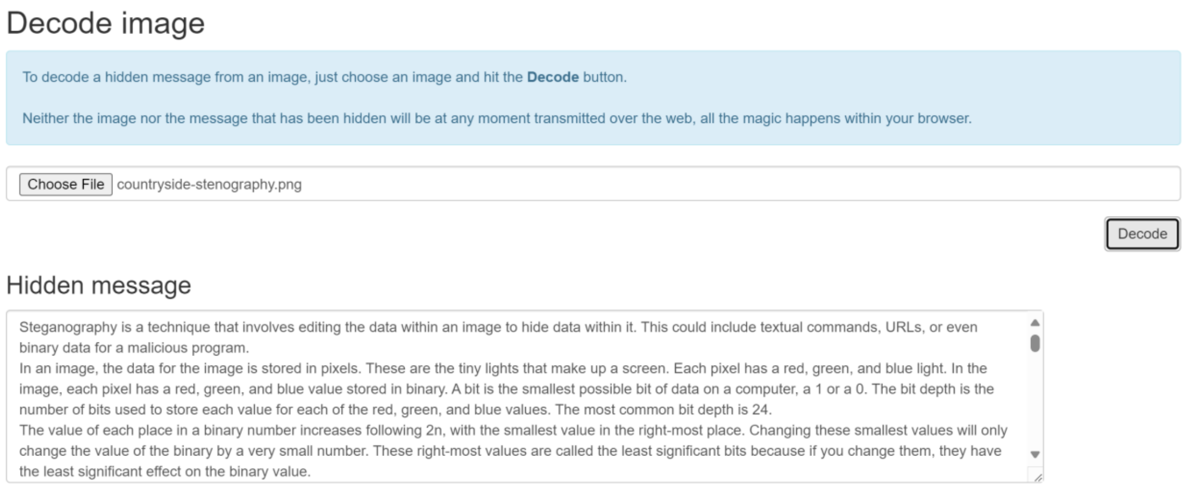
Steganography is a technique that involves editing the data within an image to hide data within it. This could include textual commands, URLs, or even binary data for a malicious program.
In an image, the data for the image is stored in pixels. These are the tiny lights that make up a screen. Each pixel has a red, green, and blue light. In the image, each pixel has a red, green, and blue value stored in binary. A bit is the smallest possible bit of data on a computer, a 1 or a 0. The bit depth is the number of bits used to store each value for each of the red, green, and blue values. The most common bit depth is 24.
The value of each place in a binary number increases following 2n, with the smallest value in the right-most place. Changing these smallest values will only change the value of the binary by a very small number. These right-most values are called the least significant bits because if you change them, they have the least significant effect on the binary value.
These least significant bits can be modified by a program, and replaced with the user's own data, because all data on a computer is made up of bits. This will have a very small effect on the colours of each pixel, but not enough for a human to notice. It will then have data hidden in the image. The same process can be done backwards by looking at the least significant bits and concatenating them to retrieve the data that was stored in the image.

ORIGINAL IMAGE

STEGANOGRAPHY OUTPUT

As shown above, there is no discernible difference between the two images, however the least significant bits of the right image have been edited to contain the binary representation of the text that was put into the steganography tool.

Using the decoder tool, the hidden message can be retrieved:



Decode image

To decode a hidden message from an image, just choose an image and hit the **Decode** button.

Neither the image nor the message that has been hidden will be at any moment transmitted over the web, all the magic happens within your browser.

Hidden message

Steganography is a technique that involves editing the data within an image to hide data within it. This could include textual commands, URLs, or even binary data for a malicious program.
In an image, the data for the image is stored in pixels. These are the tiny lights that make up a screen. Each pixel has a red, green, and blue light. In the image, each pixel has a red, green, and blue value stored in binary. A bit is the smallest possible bit of data on a computer, a 1 or a 0. The bit depth is the number of bits used to store each value for each of the red, green, and blue values. The most common bit depth is 24.
The value of each place in a binary number increases following 2n, with the smallest value in the right-most place. Changing these smallest values will only change the value of the binary by a very small number. These right-most values are called the least significant bits because if you change them, they have the least significant effect on the binary value.

Steganography is slightly harder to implement. The input image has to be broken down into individual pixels and the bytes within the pixels have to be modified. The online tool that I used was 82 lines of code written in JavaScript including whitespace and comments. This is a substantial chunk of code and would take some planning to work out the code needed. But again, there are lots of examples of steganography code online that can be copied and pasted into a botmaster's program.

Linguistic Steganography

Linguistic Steganography is the process of hiding data within text rather than in an image. It doesn't use least significant bits, but rather clever methods to hide meaning behind seemingly innocuous words.

SM Metadata

Pantic and Husain came up with a method that takes it one step further. They hid the data in the metadata of a tweet. Metadata is data about data – in this case, the length of the tweet. At the time of writing, the maximum tweet length was 140 characters. This is just over 7 bits (128). Each tweet would use the length of itself to store a character to be received by the bots.

It is not practical to store 7 bits in each tweet, as the tweet lengths would be suspicious: "Certain length tweets rarely appear on Twitter so seeing, for example, many tweets of length one or two on a single account would be suspicious. To solve these problems, we can use a one-to-many encoding technique to hide information in the tweet lengths." (Pantic and Husain, 2015)

Each character in the alphabet is mapped to a number from 1-140. These numbers are then removed from the set of numbers from 1-140. Next, a character and a number are chosen "probabilistically based on the weights" until there are no more numbers left. This results in a table with each character having many numbers that correspond to them.

Now each character has multiple numbers that correspond to them, the message is separated into characters, and one of the numbers belonging to the character is chosen. A tweet is then generated of the length of the number chosen for that character. The tweet generator used machine learning to convincingly create tweets that look like they are a series of tweets from a real person and use "twitter language" i.e. retweets and hashtags. Essentially, they had to pass a simple Turing test to avoid detection.

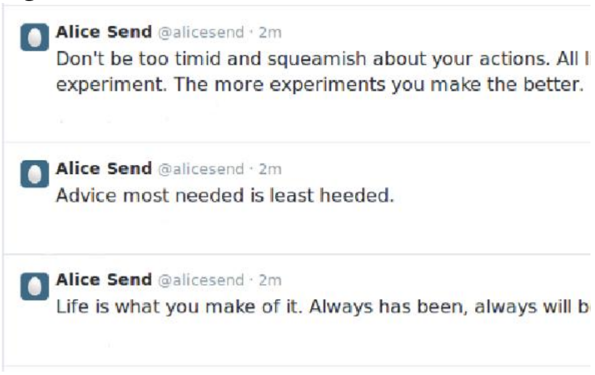
The tweets are then posted using the twitter API. The bots then each use the API to retrieve the tweets. They then do the same process backwards to retrieve the letter for each tweet.

The example given by Pantic and Husain:

Table 1

Symbol	Weight	Encoding
A	14810	32, 16, 19, 131, 84, 37, 106, 140,
B	2715	105, 138, 67
C	4943	75, 36, 125, 46, 62
F	4200	17, 122, 61, 87
...	...	...
O	14003	35, 121, 43, 107, 92, 12
...	...	...

Figure 5



Alice Send @alicesend · 2m
Don't be too timid and squeamish about your actions. All l
experiment. The more experiments you make the better.

Alice Send @alicesend · 2m
Advice most needed is least heeded.

Alice Send @alicesend · 2m
Life is what you make of it. Always has been, always will b

"Suppose we want to send the message FOO. First, the message is separated in to the sequence of symbols FOO. Each is passed to the Encode function, which chooses appropriate lengths, e.g. 61, 35, 121. The Generate function then generates tweets and they are posted to Twitter, as shown in figure 5. The figure should be read from bottom up, because the newer messages are posted on top of the older messages. [...] The recipient then reads these tweets,

obtains the lengths, then uses Decode with the same table as was used for the Encode process to get the original message.”

(Pantic and Husain, 2015)

This is an extremely complicated method to create. Botmasters would probably have to write methods such as this one themselves, as there are not many examples online of similar methods. To write this code, the botmaster would have to be a very advanced developer as it has some very complicated functions, including machine learning. Once the code is written, it would not be very hard to implement it, as it could be reduced down to one function call.

SM Comments

Another approach is to run a traditional C2 server (not on SM) but to change the location of the server regularly. This would involve changing the domain name of the server, and also changing the physical location of the server. This would mean that the server would have a continuously changing URL and IP address. The issue with this is that the bots don’t know where the server has moved to. This is resolved through linguistic steganography, one such method employed by the cyber-espionage group known as Turla (believed to be the cyber-arm of Russian intelligence (Cimpanu, 2017)) was through the comments of an Instagram post of Britney Spears.

In this case, the malicious code was a Firefox extension. However, the code for a Firefox extension is readily available for the user to view, so the attackers could not hard code the link/IP address of the server into the extension, otherwise researchers would easily be able to find it and shut it down. Instead, the extension checked the comments of a post by Britney Spears and computed the custom hash value for each comment. If the value equalled 183, it would then run this regular expression on the comment: “`(?:\\u200d(?:#|@)(\\w)`” This expression looks for the characters “@”, “|”, “#”, or the Unicode character “`\200d`” – a non-printable character called “Zero Width Joiner.”

The only comment that matched these conditions read so: “`#2hot make loved to her, uupss #Hot #X`” When the Zero Width Joiner characters are displayed, it actually read so: “`<200d>#2hot ma<200d>ke love<200d>d to <200d>her, <200d>uupss <200d>#Hot <200d>#X`”

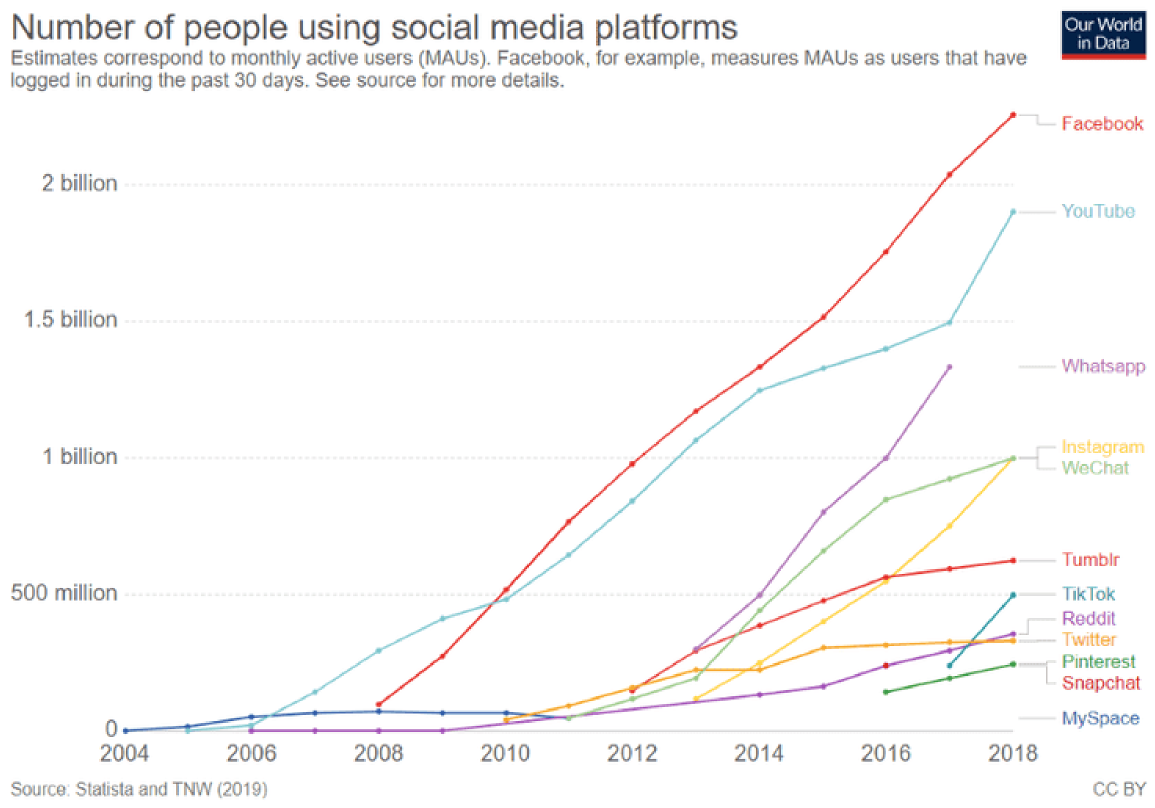
Running this through the regular expression returns this: “2kdhuHX”. This was the end of a bit.ly URL. The full link was “`http://bit.ly/2kdhuHX`” This was the location of the C2 server. This could be changed to point to any URL, and in so making the C2 server disappear. (Boutin, 2017)

If researchers were to find and remove the comment and delete the account, the botmaster could always create a new account to post the message again as many times as necessary.

Again, this is quite a complicated method to create, as the botmaster would have to create a custom hashing function that would return the same value for different strings so that the server could be moved around with multiple comments. But the bot side of the program would not be very hard to make, as all that is needed is a basic understanding of regular expressions (commonly known as regex).

Detecting SM C2 Botnets

There are a few methods of detecting C2 botnets on SM, but many of them are resource intensive and would require lots of storage space on servers to run them. Building records for each account to look for patterns would be nigh impossible, as most mainstream SM platforms have at least 500 million users, some going up to as high as 2 billion.



This graph shows the trends in users of SM platforms, and if they continue to follow this trend, many of these platforms will all have billions of users in the coming years.

Base46/Hex encoding detection algorithm

As previously mentioned, Kartaltepe, E.J. et al. created an algorithm using machine learning to determine if a SM post is being used to control a C2 botnet. It is lightweight and doesn't need to keep records about every user on the site, which is very useful for SM services that have a large number of users. It also had extremely good results:

"For standard Base64 and Hexadecimal encoding schemes, our classifier was able to quickly distinguish between our "normal" and "suspicious" text samples in an account-agnostic way, no false positives and no false negatives, for both Base64 and Hexadecimal encoding. [...]Our profiler was able to distinguish between them in an account-agnostic way, with a false positive rate of 0.0% and false negative rate of 1.25% for Alternate Base64 encodings, and a false positive rate of 3.25% and false negative rate of 12.5% for Alternate Hexadecimal encodings" – (Kartaltepe et al.*, 2010)*

As shows that algorithms can be made with lightweight machine learning that very accurately determine if a SM post is being used to control a C2 botnet. The accuracy dropped when using different encodings but was still very accurate in its results. This would be a very accurate and cheap way for SM companies to ensure that their services are not complicit with illegal activities. This means that botmasters have to come up with more advanced solutions to circumvent these algorithms.

This algorithm was “account-agnostic,” meaning that it doesn’t need to keep a record of the suspiciousness of each account, rather it would work on any SM post without the need to know what account it is from. This saves huge amounts of storage space on servers, as the amount of space needed to store extra data on (minimum) 500 million users would be enormous.

Steganography

Detecting steganography is a matter of statistical analysis, looking at the image and trying to determine if it is being used to hide data or not. This is quite complicated and would be very resource-intensive if it were to be used on mainstream SM platforms. There is however, a 100% effective method to prevent steganography. If SM platforms were to randomise the least significant bits of each image uploaded to the platform, then no image could be used for steganography. This would only change the colours of the image slightly and would generally not be visible to users.

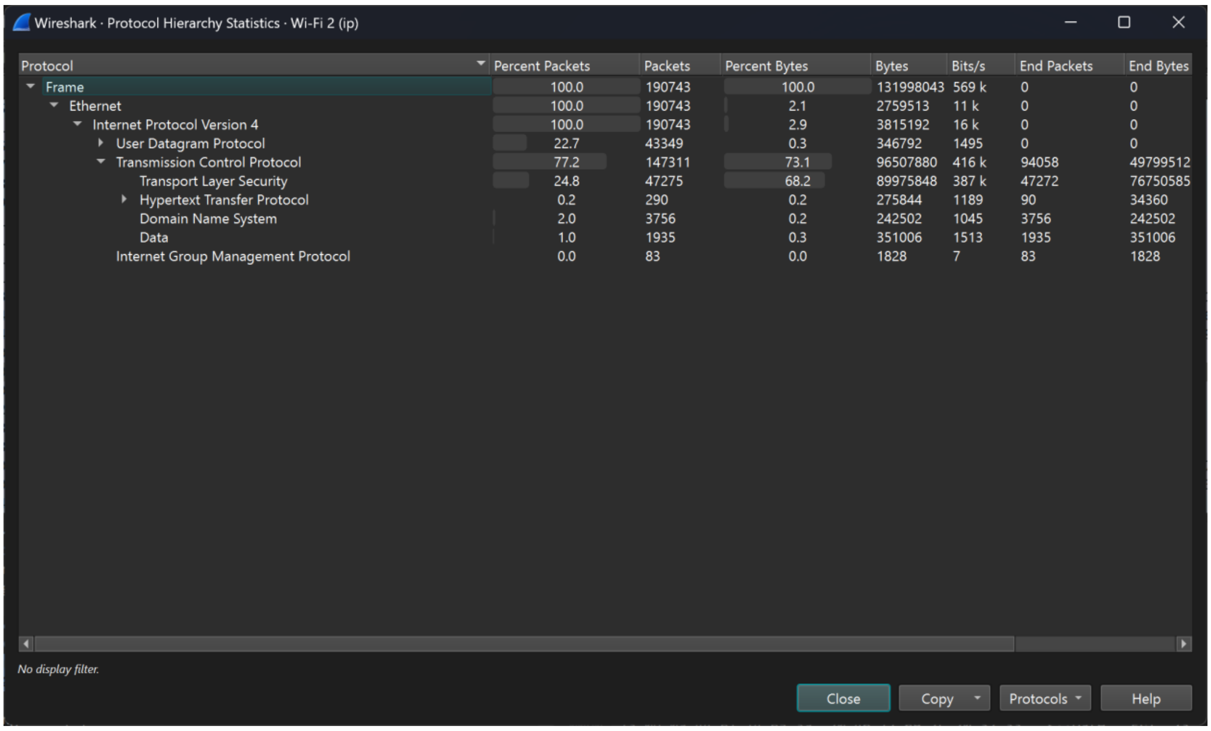
Linguistic Steganography

Pantic and Husain claimed that their method of linguistic steganography was unobservable. This would be an incredible method if that were to be the case and would pose large problems for cyber security professionals and SM platforms. However, Pantic and Husain wrote their paper in 2015, and Radunovic et al. wrote their paper in 2020. Radunovic et al. stated that the method was unobservable, but that was 4 years ago. In recent years, machine learning technology and AI has advanced incredibly rapidly. It may be that with the right training, an AI model would be able to detect patterns in SM posts’ metadata and suggest if a post is being used to control a C2 botnet or not. However, this is just speculation and there is no evidence to suggest that this would be the case for sure.

As for the method of commenting under a particular post, this is extremely hard to block, as botmasters would be able to continually create new accounts to comment, and there wouldn’t be any way to stop them. The SM platform would be able to scan every comment, but that could be resource intensive, and may not remove all comments before the bots have seen them.

Network Monitoring

One of the biggest advantages of using SM to control C2 botnets is that so much traffic in the modern day is that to the world wide web (using the http protocol). I did my own research to find out what percentage of a home network's traffic is http/https. I did this using Wireshark, capturing packets and analysing the data recorded. This was the result over a period of 30 minutes at a time when there were only two active users in the house – one of which was on a zoom call, so generating lots of traffic – but not HTTP.



Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes
Frame	100.0	190743	100.0	131998043	569 k	0	0
Ethernet	100.0	190743	2.1	2759513	11 k	0	0
Internet Protocol Version 4	100.0	190743	2.9	3815192	16 k	0	0
User Datagram Protocol	22.7	43349	0.3	346792	1495	0	0
Transmission Control Protocol	77.2	147311	73.1	96507880	416 k	94058	49799512
Transport Layer Security	24.8	47275	68.2	89975848	387 k	47272	76750585
Hypertext Transfer Protocol	0.2	290	0.2	275844	1189	90	34360
Domain Name System	2.0	3756	0.2	242502	1045	3756	242502
Data	1.0	1935	0.3	351006	1513	1935	351006
Internet Group Management Protocol	0.0	83	0.0	1828	7	83	1828

Here, we can see that the Transmission Control Protocol (TCP) accounted for 77.2% of all network traffic. Of which, Transport Layer Security (TLS) accounted for 24.8% of traffic, and Hypertext Transfer Protocol (HTTP) accounted for 0.2% of traffic. TLS is the protocol used to encrypt HTTP traffic into HTTPS, so the sum of TLS traffic and HTTP traffic gives the percentage of all traffic going to websites, which is 25%. This may not seem like a high percentage, but that was 90,251,692 bytes – 90MB. The maximum twitter post size is 560 bytes (Charles, 2011), (this is disregarding the new 10,000 character limit introduced in 2023 exclusively for paying users in the US.) This size tweet would only be achievable using special characters, as most characters only require 1 or 2 bytes.

This means that if a bot were to check for commands every 30 minutes, it would only account for 0.0006% of network traffic. If it were to check for commands every minute, it would still only account for 0.019% of network traffic.

Because the bots only account for an extremely small percentage of the total network traffic and they use very common protocols and ports, they are extremely hard to detect using traditional network scanning techniques.

Taking down SM C2 Botnets

Sinkholing

Sinkholing would not work with SM botnets because the servers that they use are those of the SM platform, and they have many servers around the globe, and the servers provide a publicly accessible service to millions (and sometimes billions) of people. To successfully sinkhole a SM botnet, the whole SM platform that is being used would have to be sinkholed as well.

Sinkholing only works on traditional C2 servers where the server's whole purpose is to control the botnet as sinkholing takes down the whole server and all services it provides.

Account Deletion

After an account has been detected to be being used to control a C2 botnet, the SM platform will often delete the account. This means that all the commands are taken down, and the bots have no commands anymore. But it's extremely easy for the crooks to re-direct communications to another freshly created account and therefore carry on undisrupted (Bisson, 2016).

This isn't a problem in the example given earlier controlled by Turla (using SM comments) as the botmaster could create as many accounts as needed and keep posting the messages.

Image Steganography

Image steganography is extremely hard to detect without the use of complex statistical analysis. This makes it impractical to check every image uploaded to SM for possible C2 commands. However, since modifying the least significant bits of an image is undetectable to the human eye, SM platforms can use this to their advantage.

When an image is uploaded to the SM platform, the least significant bits could be randomised. This wouldn't have a visible effect, but would destroy any data that could have been potentially stored inside the image. This would be a very effective way of stopping a botmaster using image steganography to control the bots.

Linguistic Steganography

A similar approach can be taken in an attempt to stop linguistic steganography. SM platforms could check every post and comment for non-visible characters and remove them. This would make it a lot more complicated to use methods such as those employed by Turla in the example above, but would be ineffective against methods such as that devised by Pantic and Husain.

Machine Learning

With the rise of AI and machine learning, it is possible that models could be trained specifically to detect possible C2 commands in SM posts and comments. This may be very effective, and could help prevent botmasters using SM to control their botnets. But as of yet, this doesn't exist and is just speculation.

Conclusion

Traditional C2 botnets controlled with C2 servers are very vulnerable, as they have a single point of failure. The methods used to communicate with the server are also vulnerable as they can be detected by network administrators. This is why botmasters have moved to controlling botnets over SM.

There are so many different ways to control C2 botnets using SM, these include regular and linguistic steganography, plaintext commands, posting URLs to online file storage, etc. These can all happen on public posts, direct messages, or comments on others' posts. The wide variety of ways that commands can be issued to the botnet on SM make it that much harder to block them reaching the infected computers.

The very basic methods of controlling botnets over SM can be detected very easily and will get taken down very quickly, but there are some very complex ways to hide commands in SM posts and comments, which in themselves are undetectable. Maybe, with the rise of artificial intelligence, models could be trained to spot otherwise undetectable C2, but this is just speculation.

The traditional method of taking down botnets – sinkholing – doesn't work because the servers being used to host the C2 are those of the SM platform. These cannot be sinkholed because they are public servers that provide a service to millions (or sometimes billions) of people around the world.

SM botnets are also harder to block because traditionally because if authorities take a C2 server offline, the botnet operation generally goes offline as well. With SM controleed C2 botnets, if the SM account is banned, the botnet can simply switch to another to receive its instructions. (Evans, 2016)

You can't ever fully block SM C2 botnets however, as you would have to block SM in a corporate environment. Such a ban obviously wouldn't be realistic in most companies. The botnet traffic would be difficult to distinguish from legitimate communications. (Higgins, 2015)

References

Bisson, D. (2016) *C&C Servers? Too Risky! Android Botnet Goes with Twitter Instead*, *BleepingComputer*. Available at: https://www.bleepingcomputer.com/news/security/candc-servers-too-risky-android-botnet-goes-with-twitter-instead/ (Accessed: 22 February 2024).

Boutin, J.-I. (2017) *Turla's watering hole campaign: An updated Firefox extension abusing Instagram*. Available at: https://www.welivesecurity.com/2017/06/06/turlas-watering-hole-campaign-updated-firefox-extension-abusing-instagram/ (Accessed: 24 February 2024).

Brenton, C. (2022) *Detecting Malware Command and Control Channels*. Available at: https://www.linkedin.com/pulse/detecting-malware-command-control-channels-chris-brenton/ (Accessed: 5 March 2024).

Charles (2011) 'Answer to "How many bytes of memory is a tweet?"', *Stack Overflow*. Available at: https://stackoverflow.com/a/5999852 (Accessed: 23 February 2024).

Cimpanu, C. (2017) *21 Years Later, Experts Connect the Dots on One of the First Cyber-Espionage Groups*. Available at: https://www.bleepingcomputer.com/news/security/21-years-later-experts-connect-the-dots-on-one-of-the-first-cyber-espionage-groups/ (Accessed: 24 February 2024).

ENISA (2022) *DNS Sinkhole*, *ENISA*. Available at: https://www.enisa.europa.eu/topics/incident-response/glossary/dns-sinkhole (Accessed: 8 March 2024).

Evans, S. (2016) *Twitter-Controlled Android Botnet Discovered*, *Infosecurity Magazine*. Available at: https://www.infosecurity-magazine.com/news/twittercontrolled-android-botnet/ (Accessed: 17 January 2024).

Higgins, K.J. (2015) *Tool Controls Botnet With Twitter Direct Messages*. Available at: https://www.darkreading.com/endpoint-security/tool-controls-botnet-with-twitter-direct-messages (Accessed: 17 January 2024).

Kartaltepe, E.J. *et al.* (2010) 'Social Network-Based Botnet Command-and-Control: Emerging Threats and Countermeasures', in J. Zhou and M. Yung (eds) *Applied Cryptography and Network Security*. Berlin, Heidelberg: Springer (Lecture Notes in Computer Science), pp. 511–528. Available at: https://doi.org/10.1007/978-3-642-13708-2_30.

L, C. (2024) 'stylesuxx/steganography'. Available at: https://github.com/stylesuxx/steganography (Accessed: 24 February 2024).

Ortloff, S. (2012) *Botnet Shutdown Success Story - again: Disabling the new Hlux/Kelihos Botnet - Securelist*. Available at: https://web.archive.org/web/20120330210044/http://www.securelist.com/en/blog/208193431/Botnet_Shutdown_Success_Story_again_Disabling_the_new_Hlux_Kelihos_Botnet (Accessed: 5 March 2024).

Pantic, N. and Husain, M.I. (2015) 'Covert Botnet Command and Control Using Twitter', in *Proceedings of the 31st Annual Computer Security Applications Conference*. *ACSAC 2015: 2015 Annual Computer Security Applications Conference*, Los Angeles CA USA: ACM, pp. 171–180. Available at: https://doi.org/10.1145/2818000.2818047.

Puri, R. (2021) *Bots and Botnet: An Overview*, *Egnyte*. Available at: https://sansorg.egnyte.com/dl/s9RHzYWkNe (Accessed: 23 February 2024).