A Steganography-based Covert Keylogger

Megan Thomas m.thomas121381@gmail.com
University of Birmingham
School of Computer Science,
B15 2TT, Birmingham, UK

Panagiotis Yialouris panagiotis.y@gmail.com
University of Birmingham
School of Computer Science,
B15 2TT, Birmingham, UK

Thomas Yorkshire tjyorkshire@gmail.com
University of Birmingham
School of Computer Science,
B15 2TT, Birmingham, UK

Abstract

Identity theft through keyloggers has become very popular the last years. One of the most common ways to intercept and steal victim's data are to use a keylogger that transfers data back to the attacker. Covert keyloggers exist either as hardware or software. In the former case they are introduced as devices that can be attached to a computer (e.g. USB sticks), while in the latter case they try to stay invisible and undetectable as a software in the operating system. Writing a static keylogger which operates locally in victim's machine is not very complex. In contrast, the creation of covert communication between the attacker and the victim, and still remain undetectable is more sophisticated. In such a scenario we have to define how data can be delivered to the attacker and how we can make an efficient use of the channel that transfers the information over the network in order to stay undetectable. In this paper we propose a system based on Steganography that takes advantage of a seemingly innocuous Social Network (Tumblr) in order to avoid direct communication between the victim and the attacker. A core part of this study is the security analysis which is also discussed by presenting experimental results of the system and describing issues regarding surveillance resistance of the system as well as limitations

Keywords: Network Security, Covert Channels, Steganography, Keylogger, Social Networks

1. INTRODUCTION

Malware (stands for MALicious softWARE) is a category of programs that are designed to compromise, damage or perform any illegitimate behaviour to a victim's computer. There are many different types of malware, such as viruses, worms, bots and many more. In this study we present a scenario based on keylogging techniques that takes advantage of the network communication, to transfer data from victim's computer back to the attacker by leveraging an existing network infrastructure in a way that could be compared to a Botnet channel.

Keyloggers are programs that exploit system's capability to intercept keystrokes of the user in different applications. Their purpose is to identify useful credentials or user's inputs that would be useful for the attacker (e.g. credit card numbers, CVVs etc.). There are various types of keyloggers but only two of them would be suitable for our project. The kernel-based and the API-based approach. The former is the most reliable (from a attacker's perspective) solution as it is very difficult to be detected. Sometimes this kind of keylogger is integrated as rootkit and has direct access to the hardware by subverting the operating system's kernel of the victim. In contrast, the latter has a different implementation. This kind of software uses hooks (and related APIs) and the operating system itself notifies the malicious keylogger every time a specific key is pressed in order to be recorded. Apparently writing API-based keyloggers is easier than writing kernel-based ones. As it was out of the scope of the study to focus on the keylogger itself we chose the faster implementation of an API-based keylogger for proper keystroke logging. More details about the keylogger implementation are provided later in the study.

Many keyloggers like those we described integrate a lot of different components in order to transmit the stolen information to the attacker. A very good practice to achieve a successful transfer is by using covert communication and more specifically covert channels. The initial idea of covert communication was identified and described by [1] where the author described covert channels as channels that are not intended for information transfer at all. Nowadays there are many ways to encapsulate data into other transferable entities (e.g. fields of network protocols - IP, TCP). In this project we used Steganography in order to hide information by embedding it into images. Steganography is supported by different algorithms and techniques and allows one to attach data into a particular picture. In this way we can have a covert and unobservable channel.

One of the most reliable and efficient techniques to achieve communication between nodes (users) is to take advantage of Social Networks. A potential attacker is able to infect a social network with illegitimate material and distribute it widely. In our project we tried to exploit user's sociability to distribute the stolen data, and make them reachable to the attacker. More specifically we chose to upload user's images to Tumblr. The most important advantage of using any Social Network for covert communication is that the vast majority of them seem innocuous and harmless. As a result, network traffic to Tumblr is not suspicious to an antivirus program or a third party observer.

In the next sections of the study we will present a high-level architecture to provide a bird's eye view of the system and a low level design to describe the functionality and the operation of each component shown in the high-level design. Furthermore we present a detailed network simulation and also some statistics that we observed from different scenarios. We also describe the way the system is able to resist against the threat model that we assume. Moreover we indicate the limitations of the system and which are the possible improvements we can make. Finally we provide some directions for future research on similar systems.

2. MOTIVATION

We decided to implement this work in order to investigate a wide range of technologies and capabilities that a network system could have. In addition this work was also of great interest to us, because there was a plethora of security requirements to meet and that made it even more challenging. Finally such a project combines knowledge from different fields of Computer Science (e.g. Network Security, Network Programming, System Programming, Operating Systems, Cryptography/Steganography) and that gave us the opportunity to achieve a good team product based on our backgrounds.

3. THREAT MODEL AND IDENTIFIED ASSETS

In this study we assume the threat model of the external passive attacker. To justify this assumption we have to mention that as we use covert channels for communication and low level key interception the whole system stays undercover both from the administrator/victim of the computer and of course the Internet Service Provider. As a result we do not consider internal attacks. Additionally it was assumed that infecting Tumblr with images containing hidden data cannot be detected by the Social Network itself. Finally as cited by Nagaraja et al. [2], Albert et.al [3] showed that scale-free graphs are very robust to the removal of different nodes when they are selected randomly. Therefore we presume that our system is also robust to a reasonable rate of node dropping as Social Networks are often scale-free graphs [2]. Further discussion will be provided in the relevant section to present simulation scenarios.

In this part of the study, it is important to highlight the assets that we consider for protection in the Botnet. The major asset transmitted in the network is the logs of the user. In other words the information acquired by the victim and considered as useful by our filtering system. Apparently we can use the same channel to transfer any kind of information. However, for this case study we consider data from logs as the only asset, as the images themselves that are transferred through the Botnet, do not have any substantial value for the attacker.

4. BACKGROUND AND RELATED WORK

Infecting Social Networks with malware (such as bots) or other illegitimate software has become really popular. As it is stated in [4], Social Networks are exploited in many different ways as an attacker is able to exploit user's needs and expectations. In terms of security, we can say that social media is the best place to use social engineering. This may happen because many people are very "social" and they are tend to trust someone who will pretend to be their online friend. This behaviour led us to infer that distributing an image in such a network, will help us make the illegally acquired data available to the attacker easier. There is a lot of research over the last years regarding covert channels of communication, scalability and performance of decentralised Botnets and also security and privacy of file hosting services that could be compromised.

More specifically related work includes Stegobot that was proposed by Nagaraja et al. [2], which is a new generation Botnet communicating over probabilistically covert channels. It is implemented to spread via social malware attack and steals personal information from users. Similarly to our work, Stegobot uses steganography to hide the presence communication in the image. Porras et al. [5] [6] analyse the robustness and scalability of two different large scale Botnets (Conficker and Storm) that are based on a peer-to-peer and decentralised architecture. A very interesting example of covert communication was proposed by Nappa et al. [7]. They suggested a way to exploit a network such as Skype in order to build a parasitic overlay. The use of Skype was making really difficult for security researchers to find the botmaster and track the Botnet as this would cause serious damage to legitimate Skype users. Our work also includes interaction with Social Networks and file hosting exploiting as downloading occurs automatically. Related work as a solution to the problem of automatic downloads and crawling of file hosting was proposed by Nikiforakis et al. [8] who suggested a very interesting client-side solution for File Hosting Systems that were vulnerable to crawling attacks. That solution was based on encryption and steganography in order to protect private files and information from malicious disclosure. Finally the future of keyloggers, regarding logging keystrokes on smartphones, was proposed by Damopoulos et al. [9], where they suggested a fully operational touchlogger iOS platform.

5. HIGH LEVEL DESIGN

In this section we will provide the general architecture of our system and we will explain what the role of each component is in the whole process. A bird's eye view of the architecture is illustrated in Figure 1.

The whole process starts with the keystrokes interception. The keylogger is responsible for capturing not only the keystrokes but the title of the window where the keys are pressed. Each key is transmitted to a logfile where all the keystrokes are stored for later use. It is important to clarify that the keylogger itself operates as an isolated program, while the rest of the process (including networking activity and capabilities) are integrated in a semi-automated system. The system is considered as semi-automated because we assume that the process in initiated by the user.

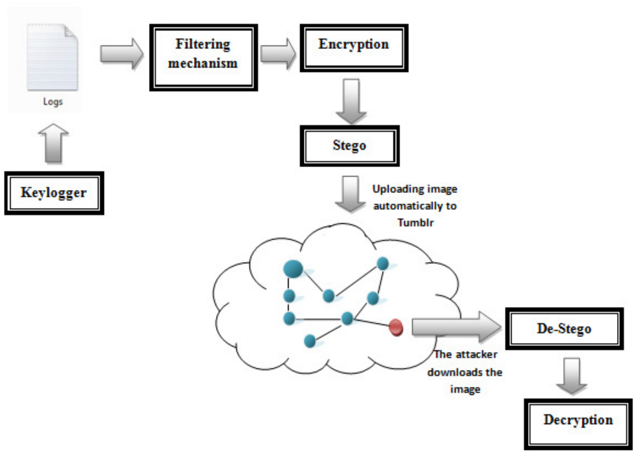
Logs
Keylogger
Filtering mechanism
Encryption
Stego
Uploading image automatically to Tumblr
The attacker downloads the image
De-Stego
Decryption

FIGURE 1: High Level Architecture of the system.

In the next stage the system performs a data filtering. The file where logs are stored is used to extract useful data. This means that the system searches for specific information in the file. This type of information includes e-banking accounts, e-mails, passwords, credit card numbers or CVVs. Apparently, not all the keystrokes are useful to the attacker. In contrast, the vast majority of them are totally useless to someone who wants to make money out of that process. As a result, we need this mechanism to find potentially critical information in the file. We consider that this stage is important to our general design in order to reduce the amount of information which is transferred through the covert channel and increase the level of unobservability as we create less network traffic.

After collecting all the information which is considered useful by the system we encrypt it using AES algorithm to add an extra layer of security. As we will describe later, the technique that is used for Steganography has particular limitations. Therefore there was a need to protect data from a third-party observation. By using encryption we do not allow a third-party user to find the owner of the data when the image is publicly exposed. For example, if one observes data embedded to an image and achieves to extract data, he will not be able to understand who is the real owner (e.g. by noticing a username or a password) as data are protected. In addition, nobody (except of the attacker) will be able to know to real owner as the image could have pass from many different computers and networks.

Once the data has been filtered and encrypted it is then ready to be embedded in images via the method of Steganography. In this system, the images used to embed the data are PNG images. Although using JPEG images would have made the data harder to detect within the image, PNG images are uncompressed and therefore it is much easier to embed data within them, so it was decided to use these instead. The steganography process in this system is based on the **org.steganography** library [10]. When doing the steganography, the data are placed into the image sequentially (i.e. the pixels of the image are accessed sequentially, beginning at the top left-hand corner, and ending at the bottom right-hand corner). The data are not scrambled before being embedded, as it has already been encrypted, and therefore is unreadable to begin with. The scheme used in this system is the Least Significant Bit (LSB) method of Steganography. This method simply replaces the least significant bits of some of the bytes in the image with the bits of the data to be embedded. In particular, this system uses a dual LSB method, which means that each byte of data to be embedded is encoded using 2 pixels of 32 bits each, the first pixel encodes 6 bits of the byte, while the second encodes the remaining 2 bits.

After that stage, the image is ready and is uploaded to Tumblr. At this point the image is available to the attacker. Having downloaded an image from Tumblr, the system then performs the reverse of the Steganography scheme described above in order to extract any hidden data present in the downloaded image. Encrypted data are then decrypted to give access to the plain-text data hidden in the image. Any C&C messages contained within the data can then be acted upon by the system, while key-logging data extracted will be amalgamated with the data stored on the infected machine, ready to be encrypted and embedded within another image.

6. LOW LEVEL DESIGN

In this section an elaborate analysis is presented concerning all the components of the system as illustrated in the high level design in Figure 1. Explaining each subsystem in detail is important to understand its significance and role in the covert channel. Moreover it is helpful for clarifying issues in the following section, regarding the security analysis of the system.

6.1 The Keylogger

The base of our system is a keylogger program. The keylogger prepares the text file that will be used from other subsystems in order to provide the stolen data to the covert channel.

While a keylogger is a system-dependent program we had to decide about the appropriate API that we would use. Since it was out of the scope of the study we didn't implement the keylogger itself to be compatible with every Operating System. As a result we decided to make a keylogger that would be compatible with different versions of the same Operating System. The optimum choice was Windows and of course Windows API [2]. Further analysis of the low level functionality of the keylogger is out of the scope of this study.

In order to add an extra layer of unobservability we added some useful components on the keylogger to make it more powerful. The first one was related to file management. The file was able to be created in a path that we were able to define (e.g. in System32 which exists in every Windows system or anywhere else) as usual. In addition to this we made the file treated as a hidden one meaning that was not obvious to the user. The file was created and appended as a hidden. The second problem that we identified and solved was the problem of usability of the keylogger. The keylogger should be covert and we had to find a way to do not lose any keystroke after a possible restart of the computer. Therefore we designed the keylogger to be executed as a startup right from the source code. To do this we used a registry function to open a specified registry key to the system and another one to handle the value and the type of information of a particular registry key. As a result we managed to create a new startup program for the user (which is the keylogger) that would start each time the user starts his machine. More specifically we used the HKEY_LOCAL_MACHINE registry hive that in order to create this startup for all the users of the computer in contrast to HKEY_CURRENT_USER that would create the startup program only for one particular user. Thus, we achieve to have the keylogger as a startup for any user in the same computer and store their keystrokes in the same file.

In order to evaluate the initial requirement of system compatibility and functionality of this program we tested it in a Windows XP (32-bit), a Windows 7 Starter (netbook version, 32-bit) and a Windows 7 (64-bit) machine and it worked properly in all these versions. So, we managed to have a broad range of machines that could be infected by the same program. Finally in order to test the unobservability from antivirus systems we used three different Antivirus to test whether they we were able to detect it as malware. All of them failed to detect it and no alert was triggered. This is probably because they use a database to compare it with fingerprints of already reported malware and as a result they are not able to find anything.

6.2 The Filtering Mechanism

The filtering mechanism assists in finding useful data in the hidden text file created (and appended) by the keylogger as we described in the previous section. Apparently only a small percentage of the contents of the file has real value to an attacker. So, we need to isolate the amount of information that should be transferred through our channel.

The design and functionality includes information retrieval based on patterns. We imparted to the program an efficient structure that would let it be maintainable in order to add or remove patterns from the retrieval and make the system understand easily what we want to isolate in each change of our preferences. Patterns are always a good way to look for, but combining this search with tokens gives our system a model-driven approach which allows us to introduce specific changes and improvements efficiently.

As we described in the high level design, we look for particular information such as credit card numbers, CVVs, e-mail accounts or e-banking accounts. For example if we want to find a bank account we have defined three different patterns for banks, meaning that we look for bank names as substring (e.g. Barclays, Lloyds, Hsbc). Similarly we look for credit card numbers in the text file by searching for elements with the following formats:

- AAAA.BBBB.CCCC.DDDD or
- AAAABBBBCCCCDDDD

Where A,B,C,D identified as integers from their ASCII value. In the first case we also look for 3 dots that separate this word Similarly we look for a pattern like the following:

- " "ABC" "

Where ABC integers again. That means we look for a word of only 3 integers where a space exists before and after the word. We use the double quotes here for convenience to describe the example.

We could create as much patterns we want but continuing this way we couldn't face the problem of maintainability. In that way we decided to use to tokens to decide what our system should each a pattern recognition was successful. In each case of the above our recognition function returns a result (token). This result defines the type of info that we want to retrieve and store. For example we decided that each time where a bank name substring is found we will retrieve this word and the next two words found in the file (because it would be very likely to be a user name and a password from a bank account). Similarly when the symbol "@" is found in a word, we want to retrieve that word and the following one (probably the password for an e-mail account). In contrast when we find credit card numbers or CVVs (by the pattern described above) we want just those words to be retrieved from the file. As a result we consider three models of retrieval.

1. Retrieve only the recognised word.
2. Retrieve the recognised word and the next one.
3. Retrieve the recognised word and the next two words.

In order to define in which retrieval family where the recognized word belongs, we give the token the appropriate value. So for example, if the function returns "2" as a result, we assume that we have a situation of an e-mail and we also get the next word which is probably the password.

As a result we managed to model our retrieval and make an efficient use of the patterns. In that way, we can add patterns easily, and just give a token to describe the retrieval family. Of course we can add more retrieval families based on our preferences that could be identified by a new value of the token. Finally the same mechanism stores the useful information that was acquired to a data structure in order to be used later for further processing.

6.3 Encryption

After storing the useful information, the system makes the appropriate processing to prepare the data to be used in the covert channel. The first step is to encrypt the data in order not to be embedded as plaintext in the file. As a result in a case where a passive adversary manages to extract the stolen information from the image, he will not be able to see the information as plaintext. This is useful because no third-party is able to steal from us the information transferred through the covert channel. In addition, as we described earlier, as the information stays encrypted we provide a level of anonymity for the user and repudiation to the attacker. That happens in a case where a legitimate third-party wants to find who is the attacker by suspecting potential enemies of the victim (in a real life scenario), is not able to recognize the real owner of the data (for example by investigating a credit card number) as encryption protects the sensitive information transferred.

To achieve encryption we use AES algorithm and cipher-block chaining (CBC) as a mode of operation. There is also used an unique Initialisation Vector (IV) for encryption and decryption. We use AES-128bit version (128-bit is the length of the key). We chose this length because there is no reason to use a longer key as 128-bit keys are secure and longer ones (192-bit and 256-bit keys) are preferred for purposes where security is very important and for legislation compliance (e.g. military services etc.). In addition we assume that the encryption occurs in the victims computer, so we need this process to be as lightweight as possible, because longer keys introduce extra rounds in the key scheduling and make the process more heavyweight which is something that we don't want. Furthermore we use PKCS5Padding schema in order to pad plaintext to be multiples of 8-byte blocks. The reverse applied for decryption as well when the attacker extracts the hidden data from the images at the end of the process. More about the security offered by the encryption will be discussed later in the section for external passive attack resistance.

6.4 Steganography

Once images have been placed in the “downloaded” folder, the deSteganographise method is then called to begin the Steganography process. This method goes through each image in the folder and reverses the Steganography process described in the high-level design. Any key-logging data contained within these images is then placed in an array list ready to be added to the data to be uploaded. When all data has been retrieved, this method then calls the Steganographise method in order to complete the second part of the process. The Steganographise method takes in an array list of all data retrieved via the deSteganographise method and amalgamates this data with any key-logging data collected on the infected machine. It then begins the process of embedding this data in images stored on the infected machine. To do this it takes images from the “stock” folder and performs the Steganography process described in the high-level design in order to embed the data available in these images. For each image, it also ensures that the amount of data to be embedded does not exceed the maximum amount of data able to be contained in this image [10]. Each Steganographised image is placed in the “upload” folder, ready to be uploaded to Tumblr. This process will carry on until either there is no more data available, or there are no more stock images available.

6.5 Routing

In order for the Botnet system to be effective, a number of considerations must be made that are specific to the type and scope of the network system we are attempting to design. A few key points must be kept to in the design. The routing algorithm must satisfy:

Incorporation of redundancy. This is because the actual data transfer is initiated by the victims themselves (in the uploading and viewing of images on Tumblr). The routing algorithm needs to be designed in such a way that changes in frequency and size of images user's upload, doesn't affect the overall transfer of the network too much.

As part of the extreme covert nature of the network, the routing algorithm has to be able to effectively route data to the destination without actually knowing which node is a master node. All nodes will behave in the same way and the bot master can just observe at any node and collect data in a passive manner.

Taking the above point into consideration we also have to ensure that as much data as possible gets to the master node in a reasonable amount of time. Some losses are acceptable as the volume of unique data are more important than losses of portions of some of the data.

The routing algorithm used in this system is based on a type of restrictive flooding. Since each node in the system can see all images posted to Tumblr by its 'followers', it is not possible to restrict which nodes to send images to, as in conventional restricted flooding algorithms. Instead, the algorithm restricts the flow of data in two different ways. Firstly, each node keeps a record of all data has previously seen, only choosing to re-upload data which have not seen before. This keeps the amount of data currently flowing around the network at a minimum, and also helps to ensure that portions of the network do not get stuck in a loop of sending and receiving the same data. Secondly, when a node downloads images from nodes it is following, instead of simply downloading all images from all nodes, it chooses a random sub-set of nodes to download from, and downloads all images from these. Since a node can only upload any data when the user of the infected machine decides to upload an image to Tumblr, the amount of data that can be uploaded by any one node is severely restricted (on average, a Tumblr user uploads less than one image per day [11], [12]). Therefore this second limitation is necessary in order to restrict the in-flow of data to a node to an amount which can reasonably then be re-uploaded. This ensures that the flow of data in the system is not strangled as a result of flooding the nodes with too much incoming data. These mechanisms also help to ensure data redundancy. Although nodes that have already seen unique data will not bother replicating it again, many different nodes can still have this data available from them. If a portion of nodes get taken down, it is still highly likely that a number of other nodes that had previous interactions with down nodes are able to continue to serve the data to other nodes.

7. ANALYSIS

In this section the security analysis of the system is presented, and the simulation scenarios that took place during the study are also described. In addition some statistics regarding performance and efficiency of the system are presented and discussed. Finally we will present some limitations of the system and explain why is important to make some improvements in future work.

7.1 Simulation and Experimental Results

As part of the methodology used to determine the usefulness of this Botnet system to an attacker, a large scale simulation program was created. This simulation gave us the opportunity to determine the viability of using the system in a real world setting as a financially beneficial resource to the attacker/bot master.

7.1.1 Simulation Architecture

The system simulated primarily the networking traffic part of the system on a closed computer system. The simulation did not use any Tumblr accounts. In order to simulate a bot network of sufficient size, a node number of about 1000 nodes was chosen. These nodes represent each Tumblr user with their own account. Within the simulation each node had two primary tasks in order to emulate an active Tumblr user going about their Tumbling. Firstly, each node is assumed to be infected with a keylogger and will thus generate a new, unique, key logged data to be sent via the bot network. Secondly, each node has to transfer data to other nodes according to the routing method specifically developed for this botnet. The nodes also have to run according to how the victim will use them. Because this system is heavily reliant on the user interacting with and actively uploading and viewing photos (as one of the mechanisms of covert communication), the nodes also have to simulate this, and it's effect must be incorporated into the inefficiencies induced when routing the data via the network.

In order to automatically construct a network of 1000 nodes, the system randomly generates neighbouring connections according to a node to follower ratio that is similar to the average number of connections as used by Tumblr. Each node is then allowed to run and pass data through the network according to the average rates of available bandwidth. This is determined by how often a user uploads an image and the size of the images that are uploaded. In order to make the simulation as accurate a possible, typical values for these figures were used. For example, it was found that an average of 0.8 posts a day are attributed to one user. This was found using average of the number of Tumblr users [12] and posts per day [13]. The amount of unique data generated per day by the keylogger per node used was around 2.7kb. This was determined from the combination of several studies on real world botnet and key logging systems in terms of useful (i.e. credit card numbers, passwords etc.) key logged data [14]. Each node within the system then allowed an amount of information to be 'downloaded' by each other node based on the last 15 images uploaded. (This figure was come to by the fact that users are probably only likely to view the last page of images of a user at one time). This also means that each node only has to process the last 15 images to gain new data. Any other data after these last 15 images should already have been read, and passed on via the network by some other node. Each node however does keep a list of unique data it has seen so it does not pass on already seen data.

In order to gain relevant information from this system, a random node was chosen as the bot master node, and statistics were collected from it. The main piece of information that was obtained by the node was amount of unique information it had seen. Unique information in this simulation is defined by information that hasn't been seen by a node out of the pool of information from all nodes in existence. This allows us to calculate how much of the total data a botnet is generating actually gets to the bot master.

7.1.2 Results

Figure 2 shows the rate of data transfer around the network for a period of time after unique data has stopped being generated. This is used to demonstrate the time taken for unique data to be moved around the network until it is eventually seen by a master node. In this example, all nodes generate data up to 5 days upon which they stop but continue to route data around then network.

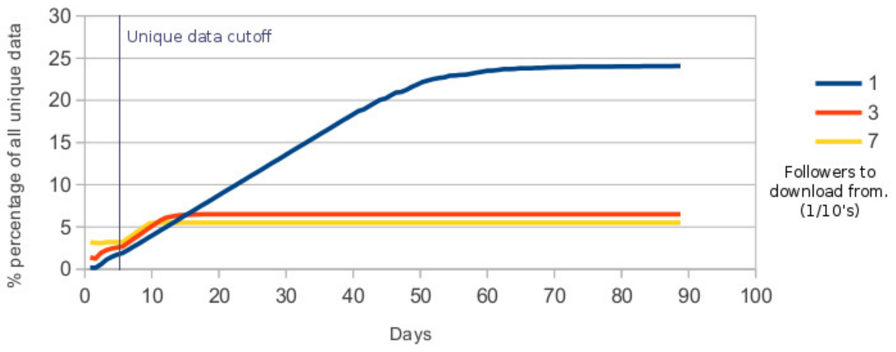

FIGURE 2: The relationship between the number of days active and the total data as seen by an attacking node as a percentage of all unique data created within the network.

From Figure 2 we can see that within this network after about 70 days we will have collected about 25% of the total unique generated data in 5 days. In comparison to the last graph, this shows that at least a third of all data generated in the botnet will reach the bot master at some point in time.

Figure 3 shows a simulation run on the system where all nodes are generating unique data daily according to average rates. The two different series on the graphs relate to the number of random followers to forward data from (as is specified in the routing algorithm).

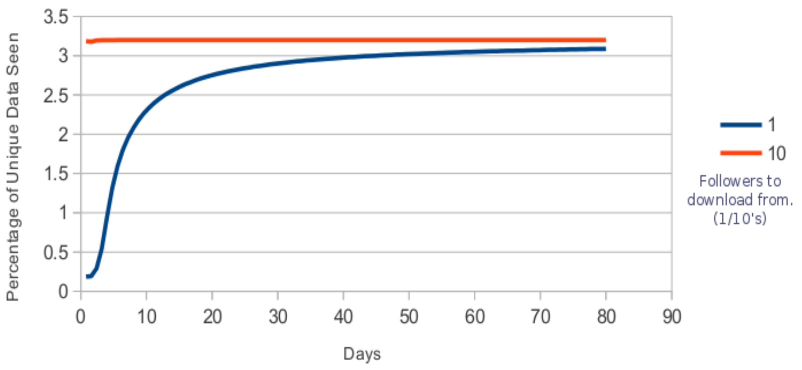

FIGURE 3: Stabilisation of seen unique data within the network.

From Figure 3 we can see that after about 60 days the system reaches a flat line peak in which a master node has received 3% of ALL unique data within the system. This may not sound impressive, however, it is important to note that this figure is the figure for all unique information ever generated by the botnet. The fact that the figure stabilises at 3%, while new data are also continually being generated and added to the mass of data currently being routed means that the network is continually and reliably forwarding data that the master node will eventually acquire.

In order to put these figures into perspective at a rate of 3%, which we can assume will be maintained for a year without any node drop-outs. And at a daily keylog generation rate of 2.7 Kb per day. After a year, the total amount unique data generated by all the keyloggers (assuming 1000 nodes) will be 985.5 MB. At 3% this is 29.565Mb for 1000 nodes. If we assume that the simulation can be scaled up (which should be the case as all node interactions will remain the same, but data generation should be higher) then for a system such as the real key logging botnet analysed in [14] with ~160,000 nodes, after a year, our system has the potential to acquire around 4.85Gb of unique data. This data will all be either credit card numbers, passwords, accounts etc. and based upon the current black markets rates for this information, the system could be worth up to USD $2.2million (GBP £1.45million) a year [4].

From figures 2 and 3, it can be seen that the random follower parameter can have a deep impact on the routing of information. Higher values allow data to be forwarded more quickly at the loss of other data however. Lower values slow the process down, however much more data (~25%) will eventually reach a master node. In a system where long term commitment and volume of unique data are most important, a lower value is more suited.

7.1.3 Node Dropping

In order to ensure reliability in cases where a proportion of the botnet fails or is taken down, it is of importance that the system and the network routing design allows for dropped nodes whilst trying to maintain performance of the system. There are two ways in which efficiency could be avoided in this system. We can either attempt to make the network resilient in terms of speed of data transfer and bandwidth, or we can make the network resilient in terms of as little data loss as possible when nodes drop. Within this system, it has been decided that volume of unique data are more highly regarded than the transit time to the bot master.

The simulation of the original bandwidth test (in which unique data was no longer generated after 5 days) was used as a base for this simulation. In this simulation however, after 10 days a certain percentage of random nodes was terminated. The unique data collected at the master node was then recorded.

Figure 4 shows how the network reacts to a sudden drop in nodes. The data received is measured in percentage of total unique data created as seen by the master node.

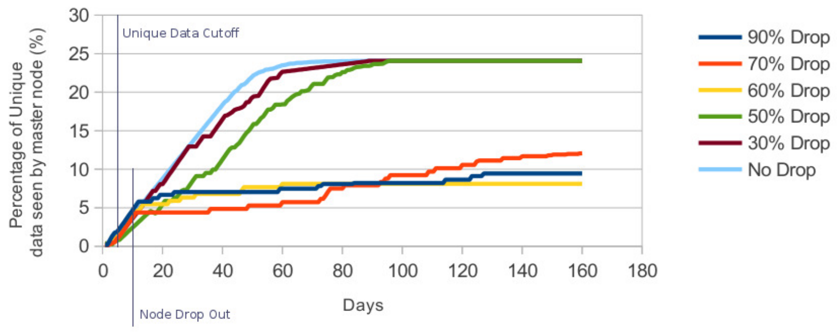

FIGURE 4: The effect of node drop out on the total unique data as seen by the master node.

It is clear that after a significant number of nodes have been dropped the time taken for the level of data that the master node received is on a time scale much slower than the original fully functioning network. However due to the huge redundancy of data within the network (i.e. many nodes retain copies of the same data for some time until it has been forwarded by multiple other nodes) most of the data that is within the network is still available. Although the volume of data are now higher than the original bandwidth capacity of the network it so well distributed among many different nodes that much of it still eventually reaches the master node.

It appears that any number of nodes under 50% of the current total, has a minimal effect in terms of data lost within the botnet system. The data eventually arrives at the master node, but in a longer time frame. For any node drop out above 50% data loss starts to occur as each node does not retain data for long enough in comparison to its ability to transfer data to other nodes in the network. The network however does not reach a point of critical fatality, its performance merely degrades (drastically, at very high drop outs) as more nodes are lost.

All these results are representative of having used only one master node. An attacker could quite simply set up a number of master nodes at random points around the network of infected computer and easily increase unique data uptake, making this system as demonstrated by these simulations an effective but still highly covert botnet.

7.2 External Attack Surveillance Resistance

Despite the fact that this is a black-hat project and we perform as an attacker, we can present how this system is resistant to external passive attacks from other adversaries. Apparently we cannot consider internal attacks, because as we described in the threat model that we assumed a covert keylogger is hidden from the user or the ISP. So, given the fact that we achieve unobservability from the user we eliminate internal attacks. Therefore it is interesting to see how we can achieve resistance against external attacks focusing on passive ones that we assumed from the threat model.

In an external passive attack on a network system the attacker is only able to eavesdrop and perform traffic analysis in contrast to active attacks where the attacker is able to be to interfere actively to the communication (e.g. create, reroute, or modify messages). We consider that Cryptanalysis is a basic category of potential passive attacks. There are different attacks related to Cryptanalysis. An example of such a passive attack that our system is able to resist is the classic brute force attack as the 128bit-AES is not vulnerable to this attack. The same applies for algebraic attack. In such an attack the malicious attacker has to represent the cipher as a system of equations and try to find the key through this system. Moreover our system is secure against code book attacks. Code book attacks are effective when the same key is used for producing ciphertext many times. This allows the attacker to collect ciphertext from different encryptions and try to analyze it in order to find the key. Recent algorithms (such as AES-128bit implementation) are able to prevent this and make it practically impossible. Therefore a potential attacker who wants to use a code book attack will not be able to use this attack even if he receives a large number of ciphertext encrypted with the same key. Therefore is probabilistically impossible to use code book against this system. In addition, in contrast to older DES, AES provides resistance for linear and differential cryptanalysis [15] (basically provided by the non-linearity which can be added by having bigger S-boxes than DES) [16],. As a result we can say that we can ensure data integrity based on the cipher. Finally as we described earlier, encryption also allows to provide a level of anonymity as in a case where data are extracted nobody is going to be sure about the real owner. In some cases this could provide a level of repudiation to the attacker. This may happen when a third-party is unable to find where the encrypted data belongs (as also described in the encryption section). So, when is so difficult to find the owner, is much more difficult to blame even the real attacker.

All the above are considered only in the case where a passive attacker is able to detect data in the hidden pictures. We remind that we achieve a good level of unobservability of the actual data transferred due to Steganography.

Other passive attacks that could result from eavesdropping and packet sniffing may include revealing the location of the victim and finding the frequency of this communication. In the former case we do not consider it as a vulnerability to our system as revealing the identity of the victim does not affect the attacker at all. In the latter case we assume that the communication is initiated by the user. Therefore this data transfer occurs randomly (e.g. when the user uploads a picture) and not automatically as part of our system.

7.3 Limitations of the System

There are also some limitations on our system. One limitation is presented in the steganography scheme as this particular algorithm (LSB) is considered detectable and vulnerable to Steganalysis attacks. Raphael et al. [17] proposed a Steganalysis technique based on discrete logarithms. In addition Mitra et al. [18] suggested a steganalytic tool that is able detect hidden data in LSB Steganography based close colour pair analysis. Another limitation is that we might have is data injection due to our AES implementation. Moreover despite the fact that the keylogger itself was tested on different systems by real antivirus and was not identified as malware, it does not run as a kernel process therefore could be detectable from an experienced user. Finally a limitation that we can avoid is image exposure. Images are exposed to Tumblr and that makes them public material. This is something that could lead to more adversaries and probably violate security and privacy in some cases.

8. ETHICAL CONSIDERATIONS

As part of our work involved transferring personal data and interaction with social media we have to clarify ethical issues that may occur. In particular we transmitted personal data from local computer to Tumblr by embedding them to images. Personal data transferred are experimental data produced by our personal use and they do not reflect real passwords, credit card numbers or CVVs. In addition we did not intercept keystrokes and data during the use of our computers by users other than the authors. Finally we used private accounts for Tumblr to avoid exposing images with hidden data to third parties or other followers.

9. CONCLUSION

We proposed a system that infects a Social Network and creates a botnet based on nodes (users) and edges (connections between followers). The system is able to take advantage of the created botnet and transmit user's data stolen by a keylogger back to the attacker. We were able to show that using steganography to embed the stolen data in images is a viable way of relaying the data from an infected machine to the botmaster without external observers even knowing that any data has been transmitted. In addition, our simulation of the system shows that the routing algorithm proposed here provides a reasonably efficient way of transferring data from infected machine to botmaster, while providing resilience to node failure, and enabling a botmaster to collect data from any node in the network.

In contrast to the paper by Nagaraja et al [8], our system uses a different method of steganography, and also employs a more sophisticated routing algorithm. However, the conclusions reached by both parties are similar, in that both papers demonstrate that a system utilising social networks and steganographised images is not only covert, but also a viable system for propagating stolen data from an infected machine to a botmaster. There is also more research focusing on social network attacks and security. Athanasopoulos et al [19] proposed Antisocial Networks. Similarly to our work, the researchers of this study describe ways of compromising social networks and transform them to an attack platform. One similarity of this study and our research is the user manipulation that could lead to an particular attacks. Another similarity is that the social network platform is the stepping stone of achieving that black hat activity. However main difference of this study and our study is that Athanasopoulos et al [19] attempt to compromise user in the level of web by manipulating client software through HTML tags, JavaScript instructions etc. In contrast our study focuses on a complicated version of an attack that combines both system level exploitation and web content violation. Another one point that we would like to highlight is that there are many studies focusing on malicious applications created via a particular API of the social network. In that direction many research has been conducted such as Makridakis et al [20]. The interesting feature in our study is that the attack does not rely on a particular application built inside the social network, as it achieves the attack begins locally in the victim's computer and spreads globally via the social network. To conclude this discussion we can say that our idea is an extended version of the idea proposed by Backstrom et al [21], where a family of attacks is described. These attacks were able to de-anonymize users based on edges inspection and algorithms for statistical and probabilistic investigation. In addition structural steganography as a key feature of the social structure is also introduced in [21]. In a similar way to evolve these attacks, Wondracek et al [22] proposed a theoretical analysis and empirical measurements to demonstrate the potential of their experiment as an advanced way to de-anonymize users of social networks. As a result, we performed this extended simulation to confirm the validity of our experiment and provide financial and statistical results of the scale of our attack.

It was a first attempt to work on a "black hat" project and we can conclude that implementing such a project leads to different requirements and challenges. In order to provide an advanced level of security, the researcher has to think as an attacker and come up with the appropriate defence mechanisms that would be suitable for a legitimate system. However design and implementation of a system which uses cover channels and tries to stay hidden is much more challenging if one wants to stay really undercover. Another observation is that using social networks for illegitimate operations has become very popular and effective. There is a lot of illegitimate material in Social Networks that will not probably become observable by users or system administrators. As a result we can say that we should not trust everything that we see or download through Social Networks.

10. FUTURE WORK

As we described earlier there are some limitations on our system that could be avoided with further improvements. First of all we can use a better Steganography technique for embedding data to images, as LSB insertion is vulnerable to a variety of attacks. Another future improvement could include an improvement on the keylogger itself. For example a kernel implementation would be able to make it even more powerful and make it totally covert. Also to avoid data injection due to AES algorithm, a different implementation scenario could include RSA encryption. Also as the keylogger system is different than the communication system, integration of those two could make a really synchronized and fully-automated system.

Finally future research could focus on touchlogger systems that became really popular on smartphones the last years. In addition it could be useful to investigate possible capabilities of this system regarding cover communication and mobile phone's operating system exploitation.

11. REFERENCES

[1] W.B. Lampson. "A note on the confinement problem". *Communication of the ACM*. Volume 16, Issue 10, pp.613-615, Oct. 1973.

[2] S. Nagaraja, A. Houmansadr, P. Piyawongwisal, V. Singh, P. Agarwal, N. Borisov. "Stegobot: a covert social network Botnet". In Proc. of the 13th international conference on Information hiding, 2011, pp. 299-313.

[3] R. Albert, H. Jeong, A.L. Barabasi. "Error and attack tolerance of complex networks". Nature, Volume 406, Issue 6794, pp. 378-382, Jul. 2000.

[4] Symantec. "Internet Security Threat Report - 2013 Trends", Volume 19, April 2014. Internet: http://www.symantec.com/content/en/us/enterprise/other_resources/b-istr_main_report_v19_21291018.en-us.pdf, Apr.19,2014 [Jul. 2, 2014].

[5] P.Porras, H. Saidi, and V.Yegneswaran. "A multi-perspective analysis of the Storm (Peacomm) worm". Internet: http://www.cyber-ta.org/pubs/StormWorm/SRITechnical-Report-10-01-Storm-Analysis.pdf. Oct. 2007 [Jul. 1, 2014].

[6] P.Porras, H. Saidi, and V.Yegneswaran. "A Foray into Conficker's Logic and Rendezvous Points". In Proc. 2nd USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET '09), 2009. Available: https://www.usenix.org/legacy/events/leet09/tech/full_papers/porras/porras.pdf, Apr. 2009, [Jul. 2, 2014].

[7] A. Nappa, A. Fattori, M. Balduzzi, M. Dell'Amico, L. Cavallaro. "Take a deep breath: a stealthy, resilient and cost-effective Botnet using Skype". In Proc. of the DIMVA'10, 7th International conference on Detection of intrusions and malware, and vulnerability assessment, 2010, pp. 81-100.

[8] N. Nikiforakis, M. Balduzzi, S. Van Acker, W. Joosen, D.Balzarotti. "Exposing the lack of privacy in file hosting services". In Proc. of the 4th USENIX conference on Large-scale exploit and emergent threats. 2011, pp 1-1. available: https://www.usenix.org/legacy/events/leet11/tech/full_papers/Nikiforakis.pdf?CFID=373638416&CFTOKEN=62886009

[9] D. Damopoulos, G. Kambourakis, S. Gritzalis. "From Keyloggers to Touchloggers: Take the Rough with the Smooth". Computers & Security, Volume 32, pp. 102-114, Feb. 2013.

[10] N. Cottin. "Steganography made easy using Hide and Reveal". Internet: http://hidereveal.ncottin.net/download/HideAndReveal.pdf, May 2010 [Jun. 2014].

[11] Tumblr Press Information, 2014. Tumblr Internet: http://www.tumblr.com/press, [Jul. 2,2014].

[12] Webbiquity. "79 Remarkable Social Media Marketing Facts and Statistics for 2012". Internet: http://webbiquity.com/social-media-marketing/79-remarkable-social-media-marketing-facts-and-statistics-for-2012/, Aug. 2012, [Jul. 2014].

[13] Tumblr Wikipedia Article, 2014. Internet: http://en.wikipedia.org/wiki/Tumblr [Jul. 2,2014].

[14] T. Holz, M. Engelberth, F. Freiling. "Learning more about the underground economy: a case-study of keyloggers and dropzones". In Proc. 14th European conference on Research in Computer Security (ESORICS'09), 2009, pp. 1-18.

[15] J. Lu, O. Dunkelman, N.Keller, J. Kim. "New Impossible Differential Attacks on AES". In Proc of the 9th Progress in Cryptology -INDOCRYPT 2008. International Conference on Cryptology in India, 2008, pp. 279-293.

[16] K., Sakamura, X. Dong Wang. "A Study on the Linear Cryptanalysis of AES Cipher". Journal of Faculty of Environmental Science and Technology, Vol.9, No.1, pp. 19-26, Feb. 2004.

[17] C. Raphael -W. Phan, Ling Huo-Chong. "Steganalysis of random LSB insertion using discrete logarithms proposed at CITA03" In Proc. MMU International Symposium on Information and Communication Technologies (M2USIC 2003), Petaling Jaya, Malaysia, 2003, pp. 56-59.

[18] Mitra S., Roy T., Mazumbar D., Saha A.B. "Steganalysis of LSB Encoding in Uncompressed Images by Close Color Pair Analysis" IIT Kanpur Hackers' Workshop 2004(IITK-HACK04), 23-24 Feb.2004. Internet: http://www.security.iitk.ac.in/contents/events/workshops/iitkhack04/papers/cp03.pdf, Feb. 24, 2004 [Jul 1. 2014].

[19] Athanasopoulos E., Makridakis A, Antonatos S., Antoniades D., Ioannidis S. Anagnostakis K. and Markatos E. “Antisocial networks: turning a social network into a Botnet“ In Proc of the 11th Information Security Conference, Taipei, Taiwan, 2008, pp. 146-160.

[20] Makridakis A., Athanasopoulos E., Antonatos S., Antoniades D., Ioannidis S., Markatos E. “Designing malicious applications in social networks”, IEEE Network Special Issue on Online Social Networks, 2010.

[21] Backstrom L., Dwork C., Kleinberg J. “Wherefore art thou r3579x? anonymized social networks, hidden patterns and structural steganography”, In Proc of the 16th international conference on World Wide Web, 2007, pp. 181-190.

[22] Wondracek G., Holz T., Kirda E., Kruegel C. “A Practical Attack to De-Anonymize Social Network Users”, Security and Privacy (SP) 2010 IEEE Symposium on, May 2010, pp. 223-238.