A Mobile Botnet That Meets Up at Twitter

Yulong Dong, Jun Dai(✉), and Xiaoyan Sun

California State University, Sacramento, 6000 J Street, Sacramento, CA 95819, USA
{dong,jun.dai,xiaoyan.sun}@csus.edu

Abstract. Nowadays online social networking is becoming one of the options for botnet command and control (C&C) communication, and QR codes have been widely used in the area of software automation. In this paper, we orchestrate QR codes, Twitter, Tor network, and domain generation algorithm to build a new generation of botnet with high recovery capability and stealthiness. Unlike the traditional centralized botnet, our design achieves dynamic C&C communication channels with no single point of failure. In our design, no cryptographic key is hard-coded on bots. Instead, we exploit domain generation algorithm to produce dynamic symmetric keys and QR codes as medium to transport dynamic asymmetric keys. By using this approach, botnet C&C communication payload can be ensured in terms of randomization and confidentiality. We implement our design via Twitter and real-world Tor network. According to the experiment results, our design is capable to do C&C communication with low data and minimal CPU usage. The goal of our work is to draw defenders' attention for the cyber abuse of online social networking and Tor network; especially, the searching feature in online social networks provides a covert meet-up channel, and needs to be investigated as soon as possible. Finally, we discuss several potential countermeasures to defeat our botnet design.

Keywords: Mobile botnet · Online social networking · QR code

1 Introduction

With the fast development of mobile industry and technology, the number of mobile users has dramatically increased. As the most popular open-source mobile platform in the world, over 2 billion monthly Android devices were found active by May, 2017 [1]. To turn the huge number of mobile devices into an army to perform attacks like Distributed Denial of Service (DDoS), SMS interception, and spamming, attackers started to build mobile botnets [2,3].

Two common command and control (C&C) topologies are found in traditional PC-based botnets: centralized and Peer-to-Peer (P2P)-based structures. In centralized botnets, the C&C communication latency is short and the botmaster can monitor the number of available bots using single or limited amount

R. Beyah et al. (Eds.): SecureComm 2018, LNICST 255, pp. 3–21, 2018.
https://doi.org/10.1007/978-3-030-01704-0_1

of C&C servers [4]. However, centralized botnets suffer from single point of failure, i.e. the botnet can be easily disabled by the defenders via shutting down the C&C channels. Moreover, the botmaster is directly exposed to defenders when the C&C channel is monitored.

On the other hand, P2P-based botnets have no single point of failure and achieve better stealthiness for the botmaster. However, P2P-based botnets suffer from the looseness of network structure, lack message transmission guarantee, and tend to have longer latency for message delivery [4–6]. Also, P2P-based botnets require higher network overhead to keep the botnets robust [7,8].

Compared with traditional PC-based botnets, mobile botnets are inherently restricted by the features of mobile platforms: low CPU capacity, small network bandwidth, limited battery and expensive data usage. Given the above comparative study about botnet topology, a centralized botnet design is more desirable for the mobile environment. However, the drawbacks of centralized design need to be addressed for robustness and stealthiness, especially the single point of failure problem.

Our paper is an effort to solve these issues by exploiting the automation feature in QR codes and the Twitter search engine to build dynamic C&C channels with high recovery abilities. The following paragraphs introduce the techniques that are essential for our solution, as well as rationales to exploit them.

Quick Response (QR) code: QR codes have been widely used in mobile software automation in the past few years. Compared with traditional masquerading techniques, QR codes are more stealthy since it has been widely used in daily life for other purposes and cannot be distinguished by human beings. Researchers [9–11] report that QR codes have been used in several attacking methods such as phishing and social engineering attacks. However, the automatic detection and removal of malicious QR codes for security is still a fairly new topic in the area.

Online Social Networking (OSN): As one of the most popular public networking services, OSN draws attention from researchers [12–14] to use it for building C&C channels. Compared with other botnet C&C communication mediums, OSN has several advantages, such as the simplicity in implementation, the portability over multi-platform environments, and the stealthiness. Nowadays, some OSN platforms like Twitter and Sina Blog (a popular Chinese OSN platform) provide searching interface to allow users to find interested posts by keywords. We find that the search feature provides a possibility to build dynamic C&C channels instead of static (i.e. hard-coded) ones. The dynamic C&C channels can help avoid single point of failure, and thus deliver better robustness and high recovery capabilities.

Domain Generation Algorithm (DGA): DGA from Conficker [15] in 2008 is a solution to avoid single point of failure in the centralized topology. In general, DGA takes one or multiple seeds as inputs to produce random domain names. Based on the actual implementation, DGA may hugely increase the difficulty of predicting the next generated domain name, and make it computationally impossible to stop the attack through banning all possible DGA outputs [16]. In

our botnet design, we vary DGA to produce random strings instead of domain names. The DGA generation results play as countersigns (left by the botmaster) to help bots find the meetup place at Twitter.

Tor network: Internet was born as a public network, while the second generation onion router (Tor) [17] provisions an ideal technique to achieve anonymity. Tor was invented with ready-to-use client proxy and web browser. It is natural to think of Tor network to keep botmaster's C&C communication anonymous. Today, Tor has been integrated with the mobile platform, such as Orbot [18] for Android. With the appearance of Orbot, the implementation complexity of Tor-based network applications on Android is dramatically decreased, and the botmaster can easily use mobile Android devices to issue commands to botnet with fair stealthiness.

By creatively orchestrating QR codes, OSN, DGA and Tor network, our botnet design successfully enables the following features to overcome the natural limitation in traditional centralized and P2P-based botnets.

- Constructing a new OSN-based mobile botnet with no single point of failure.
- Building dynamic C&C channels with high recovery capability based on the Twitter search engine and QR codes.
- Using dynamic asymmetric key pairs and DGA with random seeds to keep the confidentiality of C&C communication traffic.
- Using Tor network to hide the identity of botmaster.
- Simple implementation and huge potential threats to all OSNs that include searching features.

Our design is generic for both mobile and PC platforms, while our proof of concept and corresponding analysis is conducted on Android platform. **To the best of our knowledge, we are the first to use QR codes as C&C communication medium in OSN-based botnet**.

The rest of the paper is constructed as follows: in Sect. 2, we introduce the related work. In Sect. 3, we elaborate our botnet design. In Sect. 4, we present the proof of concept, including a walkthrough to demonstrate our botnet workflow. In Sect. 5, we evaluate our work. In Sect. 6, we discuss potential countermeasures to our botnet design. In Sect. 7, we conclude this paper. More design rationales and implementation details are presented in [19], and the prototype code can be provided upon request for research purposes.

2 Related Work

Researchers have proposed a variety of approaches to build botnets on both PC and mobile platforms, either in traditional centralized or P2P-based topologies. We introduce related botnet research and designs in Sect. 2.1, and summarize our literature review in Table 1. The related QR code research is introduced in Sect. 2.2.

Table 1. List of related botnet research & Designs

Research	Year	Platform	Botnet topology	C&C channel	Masquerade technique
Hua et al. [20]	2011	Mobile	P2P	SMS	N/A
Zeng et al. [21]	2012	Mobile	P2P	SMS	Plain encrypted text
Faghani et al. [22]	2012	Mobile	Centralized	SMS/OSNs	N/A
Nagaraja et al. [23]	2011	PC	Centralized	OSNs	Steganography (JPEG)
Cui et al. [12]	2011	PC	Centralized	OSNs	Steganography (JPG)
Singh et al. [13]	2013	PC	Centralized	OSNs	N/A
Yin et al. [14]	2014	PC	Centralized	OSNs	Plain encrypted Text
Compagno et al. [24]	2015	PC	Centralized	OSNs	Unicode stenography
Koobface [25–27]	2010	PC	Centralized	OSNs/web server	Plain encrypted text
Elirks [28]	2012	PC	Centralized	OSNs	Plain encrypted text

2.1 Botnet Research and Design

Since Short Message Service (SMS) is a common technology in mobile environments, several researches have addressed SMS as C&C channel in botnet design. In 2011, Hua et al. [20] built a botnet with SMS and flooding algorithm. Hua's design successfully spreads one command to 90% of 20,000 bots in 20 min with each bot sending less than 4 messages. However, Hua's design suffers from the natural limitation of flooding algorithm, i.e. if defenders shut down a bot that is very close to the first one, the botmaster loses the rest of bots in the flood.

In 2012, Zeng et al. [21] proposed a SMS and P2P-based botnet design. Their research concludes that the ubiquitousness of SMS, the simplicity of accommodating offline bots, and the capability of hiding C&C commands make SMS suitable for C&C communications in mobile environment. However the malicious text messages in their botnet design is directly exposed to phone owners, and the monetary cost of SMS may attract the owners' attention even without anti-malware alerts.

On the other hand, Faghani et al. [22] designed Socellbot which compares SMS and OSNs as communication medium in mobile environment. Based on their experiment results, OSNs excel in lower network traffic load and faster propagation speed, and hence are more suitable for mobile environment than SMS.

Nagaraja et al. [23] introduced a botnet design by combining steganography and OSNs. In Nagarajia's research, all the C&C communication commands are hidden in JPEG images. The botmaster and bots use two hard-coded OSN accounts as C&C channels. In Nagaraja's design, the C&C traffic is stealthy, but the botnet suffers from single point of failure. If the hard-coded OSN accounts are detected and banned by defenders, the botmaster loses control of the whole botnet.

Similar to Nagaraja's idea, Cui et al. [12] designed an OSN-based botnet called Andbot. Andbot combines URL-flux (a variation of IP-flux), steganography, and Microsoft blog to decrease the threat of single point of failure and increase the stealthiness of C&C communication. In their design, the blog works

as a C&C channel. Bots use a hard-coded DGA algorithm to assemble an URL to find the blog built by the botmaster. After the blog is connected, bots download steganographic images to receive the botmaster's commands.

Following up with Cui's research, Yin et al. [14] reported a newer generation of botnet design combining Sina blog and Nickname Generation Algorithm (a variation of DGA) to build dynamic C&C channels. In Yin's design, the botnet has no single point of failure and high resistance to destruction. However, there is still a bottleneck that the network load capacity of each C&C channel is limited. In a large group of bots, botmaster needs to build multiple C&C channels in order to allow all the bots to retrieve the commands. Also, the identity of botmaster is directly exposed to the blog website without any protection.

Singh et al. [13] applied Twitter as the C&C communication platform for a centralized botnet. In their design, they take advantage of the OAuth mechanism provided by Twitter to ensure the origin of C&C commands. The botmaster posts C&C commands through its Twitter account. The drawback of this design is that it suffers from single point of failure. On the other hand, a list of commands are hard-coded on each bot which may be prone to the detection of any anti-virus systems.

In addition to image stenography, Compagno et al. [24] found and proved Unicode encoding can be used as a masquerading technique. In their research, the botmaster takes advantage of Unicode and hides the C&C communication using invisible Unicode characters. Compagno's botnet design is able to survive from the traditional botnet detection and defense strategies, but may be captured by character filtering and statistical analysis on the OSN posts.

Beside the above botnet designs from research, OSNs are already practically observed in real-world malware. For example, in 2010 Koobface was detected and investigated by researchers [25–27]. As a network worm, OSNs are used by Koobface to download different pieces of malicious content, and do C&C communication through several hard-coded OSN accounts and web servers. The specific OSN accounts can be banned, which causes Koobface suffer from single point of failure.

Researchers [28] also captured and investigated wild botnets with their C&C communication methods. An OSN-based botnet called Eliriks was detected based on their observation. In Eliriks, the botmaster posts the information of the C&C web server on a microblogging service called Plurk. For the C&C server's information, the botmaster uses a modified Tiny Encryption Algorithm (TEA) and modified Base-64 encoding to further masquerade the C&C server's sensitive information. The defenders were able to successfully extract the Plurk accounts used by Elriks by the time that the corresponding paper was published. In other words, Eliriks did not survive from the threat of single point of failure.

2.2 QR Code Research

In 2010, Kieseberg et al. [10] did a security research on QR codes. Based on their research, the automation feature in QR codes is vulnerable to SQL injection, command injection, fraud, phishing, and social engineering attacks. Krombholz

et al. [9] lists several experimental examples to prove that the threats from Kieseberg can actually be implemented in real-world environment.

On the other hand, Kharraz et al. [11] investigated 94,770 QR codes from 14.7 million unique web pages in 2014. In their report, they found 145 real-world malicious QR codes were used in phishing and malware distribution.

Although there is already proof of the existence of malicious QR codes, the research about QR code security is still falling behind. Yao et al. [29] did a security investigation on 31 commercial QR code scanners. Based on their evaluation, only two of them include security warnings after users scan a QR code. This is due to lack of research on the detection of suspicious QR codes.

Our botnet design has four fundamental advantages in contrast with the above related work. First, compared with other OSN-based botnets, our botnet design leverages the OSN searching feature to further randomize the locations that the C&C account appears. Theoretically, the botmaster could use any account in OSNs to publish the C&C commands. Second, beyond other masquerading techniques, we use QR codes to disguise the botnet C&C posts to ensure their stealthiness. Third, instead of hard-coded keys, we use dynamic symmetric and asymmetric keys to ensure the confidentiality of botnet C&C communication. Fourth, Tor network is successfully integrated in our botnet design to hide the identity of botmaster.

3 Methodology

We exploit the Twitter search engine, DGA, and QR codes to ensure the botnet robustness and stealthiness. Specifically, we leverage Tor, RSA [30], and Advanced Encryption Standard (AES) [31] cryptographic algorithm respectively to achieve anonymity, integrity, and confidentiality for the botnet C&C communication.

No matter how the botnet topologies evolve, pushing, pulling, and listening are the common options for bots to do C&C communications. In this section, we first give a overview of our botnet design. After that, four major parts of our botnet design are introduced in the following subsections: initialization, command pulling, information collection, and command pushing.

In this paper, the terminologies are denoted as follows: the DGA is denoted as $DGA()$, the DGA seeds as $Seed_1$, $Seed_2$, ... to $Seed_n$, the generated results from DGA as S_1, S_2, ... to S_n, the RSA key pair as Key_{pub} and Key_{priv}, and a special token as $Token$. The DGA algorithm is hard-coded on bots, with the DGA seeds derived from timestamps, for synchronization with the botmaster. The special token is a random string concatenated (with a delimiter) with its digital signature signed by the botmaster, i.e. encrypted by Key_{priv}. The token is hard-coded on all bots for authentication purpose. No RSA key pairs are hard-coded.

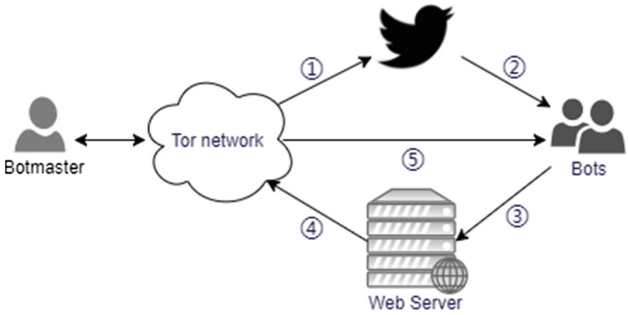

Fig. 1. Botnet design overview

3.1 Botnet Design Overview

As shown in Fig. 1, our botnet design involves five major parts: botmaster, Tor network, Twitter, bots, and a movable web server built by botmaster. The botmaster's duty is to set up the web server, prepare and publish Twitter posts, and send commands to bots. Twitter's role is to hold a Twitter post as a temporary C&C channel that allows bots to pull a QR code image from the botmaster. The Twitter post contains two major sections: a keyword generated from DGA and a QR code image. The web server is set up by the botmaster to collect information from each bot, such as IP address and device ID. The IP of the web server is propagated to bots as part of the QR code, and thus could be dynamic. The combination of the web server and Twitter posts works as C&C channels which allow bots to do command pulling and information uploading. In addition, every bot sets up a TCP server on its own device. Through information uploading, bots encrypt and upload their identify-sensitive information to the web server. When the botmaster wants to send commands to bots, it downloads and decrypts the bot uploaded data from the web server, and then sends commands to bots via Tor network.

In our botnet design, only the DGA algorithm and a special token are hard-coded on both botmaster and bot sides. The current date (i.e. timestamp) is used as a key factor to produce exactly the same DGA seeds on both botmaster and bot sides. Based on the actual implementation, DGA can take any format of seeds that is generated by the current date. All botnet C&C communication is encrypted by AES or RSA, and no key is hard-coded on bots. The symmetric keys used by AES are generated from DGA, and the public key used by RSA is spread as part of the QR code from botmaster's Twitter post. The special token is used to verify the identity of botmaster and validate the data source after decryption.

The communication between the botmaster and outside networks is via Tor. As Fig. 1 illustrates, there are five steps in our botnet design. Step ① serves as the initialization process for the botmaster to prepare and publish the Twitter post. Step ② is used by bots to download data from the botmaster's Twitter post. Then, bots upload their IP and device information to the web server in Step ③.

The botmaster downloads bot data from the web server in Step ④, and uses the downloaded data to send commands to bots via Tor network in Step ⑤. In our design, using TCP as the communication protocol in Step ⑤ has two advantages. First, it is a reliable communication protocol which guarantees message delivery. Second, it makes the design compatible with Tor network, which only supports TCP or HTTP communication.

A walkthrough is presented in Sect. 4 to demonstrate the above botnet workflow.

3.2 C&C Communication

Initialization. The botmaster needs to perform a few initialization procedures before the C&C communication starts. First, the botmaster sets up a web server. Second, the botmaster generates two random strings S_1 and S_2 from $DGA(Seed_1)$ and $DGA(Seed_2)$. After S_1 and S_2 are generated, the botmaster collects the web server's address information, as well as a pre-generated RSA key pair Key_{pub} and Key_{priv}. Third, botmaster combines the current web server address, the hard-coded token $Token$, and Key_{pub} as a command, and then encrypts the combined command with S_2. Fourth, the botmaster encodes the combined command into a QR code image. Finally, the botmaster uses a random Twitter account to publish a post, which contains S_1 and the QR code image.

Command Pulling. Bots regularly conducts command pulling, and succeed whenever the botmaster's Twitter post is available. First, similar to the botmaster, bots generate two strings S_3 and S_4 from $DGA(Seed_3)$, and $DGA(Seed_4)$. In our design, in order to ensure the synchronization between the bots and botmaster, S_3 must be equal to S_1 and S_4 must be equal to S_2. This is ensured by applying the same algorithms to timestamps for getting equivalent seeds at both bot and botmaster sides, and then using the same seeds for DGA algorithms. Bots use S_3 as the **keyword to query the Twitter search engine** to find the post from the botmaster. Bots download the QR code image based on the query response. After the QR code image is downloaded, bots first decode the QR code to retrieve raw data and use S_4 to do decryption. The botmaster's public key Key_{pub} and the special $Token$ are contained in the decrypted data, and the bots can use Key_{pub} to verify whether $Token$ is generated by the real botmaster using the paired Key_{priv}.

Information Collection. The web server is important to maintain the robustness of the botnet, as all bots are instructed to upload their real IP and device ID to the web server after IP spoofing. IP spoofing helps disguise the bot identities during C&C communications, in case they are tracked down. In order to keep their uploaded data safe, all data from bots are encrypted by Key_{pub}, and remains ciphered in database storage at the web server. This way, the botmaster can monitor and get the information of available bots in the botnet via the web server safely.

Command Pushing. The botmaster can send commands to bots at any time. For command pushing, all available bots have their current IP addresses stored on the web server. When command pushing is needed, the botmaster first generated S_5 from $DGA(Seed_5)$. Then, the botmaster combines the command for bots and $Token$ as one string, and encrypts the combined string with S_5. The botmaster queries the web server to collect each bot's IP address and decrypt the result using Key_{priv}. After that, the botmaster extracts the IP addresses of bots and broadcasts the encrypted command via TCP-based Tor network. After TCP packages are received, bots generate S_6 from $DGA(Seed_6)$. Similar to Step ① and ②, $Seed_6$ is set (via synchronized timestamps) equal to $Seed_5$ to ensure the bots can decrypt the TCP payload. After checking the existence of $Token$ for validity of the command origin as botmaster or not, bots perform tasks included in the command.

Throughout the above communications, **only the DGA algorithm and the special token are hard-coded** on both botmaster and bot sides. The rationale for hard-coding the special token will be elaborated in Sect. 3.3. The various seeds for DGA are synchronized between the botmaster and bots by applying the same computing algorithms towards the date/timestamp information. To avoid single point of failure, our **Twitter post account and the web server are dynamic**. Each time the botmaster publishes a new Twitter post, the QR code contains the information to redirect bots to the new address of the web server. If the Twitter post is banned by defenders, the botmaster can publish another post from a different and unpredictable account. If the web server is banned, the botmaster sets it up in a new address and generates a new QR code image which contains the new server address. Thus, no matter how defenders destroy the C&C channels, the botmaster always has a way to reconstruct the botnet.

3.3 Cryptography and Botnet Robustness

As we mentioned earilier, all the C&C communication in our botnet design is encrypted either by AES or RSA. Step ①, ② and ⑤ are protected by AES. Step ③ and ④ are protected by RSA. It's fully understood that RSA requires more resources than AES for computing. However, AES will require hard coding of symmetric keys for Step ③ and ④, while RSA can avoid this. Taking an extreme example, when a bot falls into a honeynet, defenders may easily track down the web server address once the bot is detected. If defenders further manage to get all the encrypted data from the web server and decipher them with cryptanalysis, other bots in the same botnet may get their IP addresses and device IDs directly exposed to defenders. Using the RSA as the encryption algorithm can dramatically decrease the risk of such situation.

In addition to strengthening the data encryption in communication and storage, using the RSA algorithm helps the bots authenticate the connections and commands from the botmaster. For example, if the defense side succeeds in reverse engineering the bot samples and obtaining the hard-coded token with botnet workflow information, the botmaster identity may be faked to hijack

the ownership of the botnet. This can be defeated by using the asymmetric encryption, i.e. the RSA algorithm. Specifically, the special $Token$ includes the digital signature generated by using the Key_{priv}, which is only owned by the real botmaster. Hence, only the real botmaster could initiate the authentic C&C communication, as nobody else could provide the corresponding Key_{pub} to verify the signatures associated within $Token$. It's possible that the reverse engineer defenders use an intercepted Key_{pub} to fake as a botmaster to issue a bogus web server address to bots. But again, thanks to RSA algorithm used in Step ③ to encrypt all upload data, the faked botmaster will not be able to decipher the bot connection information for further actions. Defenders could not track down the bot addresses as well, as Step ③ enforces bots to upload data based on IP spoofing. Deciphering the uploaded data on the web server is the only chance to push commands to bots, and only the authentic botmaster can achieve that.

4 Proof of Concept

To further illustrate our botnet design, in this section we present a quick walk-through with essential implementation details of our botnet prototype. To demonstrate our botnet design in Sect. 3, we use and run 10 Genymotion emulators on our workstation, one Google Nexus 6 phone, the Tor network, one randomly generated Twitter account, one apache server, and one MySQL database in an orchestrated way. In our demonstration, each emulator acts as one infected bot and the Nexus phone acts as the botmaster.

In order to emulate an attack, we build a victim website to let bots to perform DDoS attack after Step ⑤. As Fig. 2 shows, after each bot processes all the steps for command pulling, information collection, and command pushing, the botmaster pushes a command to bots to coordinate them to conduct a DDoS attack against a victim web server. The TCP payload in Step ⑤ contains an encrypted command which includes the information of $Token$, the length of the attack, the frequency of the attack, and the IP address of the victim website. We present some command construction details in Sect. 4.2. In our demonstration, the botmaster's web server for information collection and the victim website are both running based on apache. The emulators in Fig. 2 are performing a DDoS attack to the victim website. A full video demonstration is available at [33].

4.1 Botnet Workflow

Initialization. In our botnet implementation, we configure the botmaster to process the initialization on daily base. All the botmaster's communications with the public network (i.e. the communications with Twitter and Internet) are through Tor network. In our experiments, the botmaster generates new QR codes and DGA strings every single day. Depending on the botmaster's choice, it can also be hourly or monthly to update the QR codes and Twitter them accordingly. No matter how often the Twitter posts are published, the current timestamp is used as a key factor for DGA synchronization across the bots and

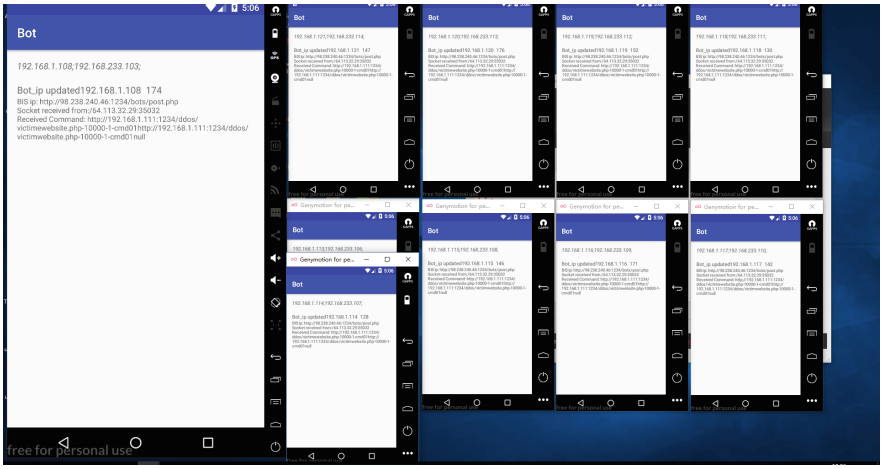

Fig. 2. Demonstration of botnet DDoS attack

botmaster. Whenever needed, the botmaster can choose to renew the address of the web server and let the bots know the change through a new QR code Twitter post. The botmaster also has the option to choose whether to use a standard or a modified QR code encoding and decoding library. After the QR code and DGA strings are ready, as Fig. 3 illustrates, the botmaster could post them with any (unpredictable) Twitter account. In our experiments, the Twitter post stays public until the next one is published by botmaster. The Twitter accounts used to post the QR code can vary everyday.

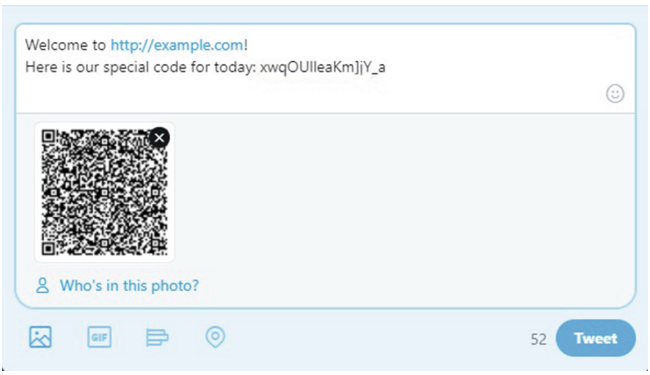

Fig. 3. Publishing a Twitter post

Command Pulling and Information Collection. In our botnet prototype, there is no initialization process like Koobface downloading different malware pieces from different locations. Because our Twitter posts are dynamic and unpredictable, there is no substantial difference between newly added bots and the bots that were previously infected. When an infected bot is invoked, as we discussed in Sect. 3.2, each bot first generates two DGA strings: one DGA strings works as the keyword to query the Twitter search engine, and the other one works as the decryption key to decipher the data stored in QR codes. By using the decrypted data from QR codes, bots submit their individual device and connection information to the web server.

Command Pushing. In our botnet prototype, instead of storing commands directly in QR codes, the botmaster pushes commands to bots whenever an attack (such as DDoS attack in our demonstration) is desired against some specific target victim. This is more dynamic and more resistant to anti-virus detection. Instead of a straightforward DDoS attack to the web server, our bots launch the reflection DDoS attack to take down the MySQL server at the backend of the victim web site. As the DDoS attack is conducted, the available connections for the MySQL server are exhausted quickly and normal users ultimately receive the MySQL connection error while loading the web page.

4.2 Command Structure

Our botnet implementation uses two type of command structures, and the summary of the command structures is given in Table 2. Command structure No. 1 is used in Step ① and ②, and command structure No. 2 is used in Step ⑤ . In command structure No. 1, commands use "]][[" as delimiters to differentiate different sections. Specifically, Key_{pub}, the current IP address or the URL of the web server, and $Token$ are the sections for command structure No. 1.

Table 2. C&C command structure

No.	Detailed command structure
1	Key_{pub} +"]][[" + Address of the web server +"]][[" + $Token$
2	IP_{victim} + "-" + Attack Length +"-" + Attack Frequency + "-" + $Token$

Compared with command structure No. 1, command structure No. 2 uses "-" as delimiters. Command structure No. 2 is used when the botmaster wants to issue commands to bots via the Tor network in Step ⑤. In command structure No. 2, the IP of victim site, the length of the attack, the frequency of the attack, and the hard-coded $Token$ are included. $Token$ is used to authenticate the source of the command. The attack frequency and attack length are specified with milliseconds as units.

4.3 Implementation Subtleties

Besides the fundamental design and implementation details of our botnet prototype, some specific subtleties can play to gain improved stealthiness and performance. Based on our observation, these tricks or their variants are applicable to the majority of the OSNs.

Fig. 4. Twitter post example

OSN Post Masquerading. Ideally, the Twitter post from the botmaster is accessible to all the public. Hence, it is necessary for the botmaster to disguise the post. A typical Twitter post in our botnet design includes a section of misleading information, a keyword, and a QR code image. As Fig. 4 shows, the random string started with *mrf* is the searching keyword. The other characters are misleading information to masquerade the post. Using this approach, the botmaster is able to fake the malicious Twitter post as a real-world commercial post which looks normal.

QR Code Fetching. After the botmaster's post is set up, bots can locate the URL of the QR code image from the Twitter search engine. Specifically, based on the REST API provided by Twitter, bots send GET requests by using S_3 as the query keyword. In our design, the bots retrieve real-world time together with time zone information from the mobile device. By using the real-world time from the device, the botmaster and bots are able to keep synchronized and ensure S_3 from bots is quivalent to S_1 from the botmaster.

Fig. 5. Twitter search response example

In the response data from the Twitter search engine, the actual URL of the QR code image can be extracted. Based on our experimental results, the images from Twitter posts are stored under the domain of https://pbs.twimg.com/media/, as shown in Fig. 5. In addition, a special unique ID is assigned by Twitter for each image, and the image format is specified in terms of suffix at the very end. By combing the domain name, the unique image ID, and its suffix, bots could identify the URL of the QR code image and download it without any authentication.

Data Usage Deduction. Data usage deduction is desired to minimize mobile device's resource consumption by both defenders and adversaries. Hence, we can leverage whatever facilitates supported by the OSNs. For example, Twitter provides Mobile Twitter Search [32] to adapt mobile data usage for bandwidth limitations. By taking advantage of Mobile Twitter Search, the data usage in Step ② can be minimized. Specifically, Twitter provides a service to shrink the size of an uploaded image. After the image URL is extracted, bots can add : *small* to the end as shown in Fig. 5, which can shrink a 770*770 QR code to 680*680 without hurt to the image quality.

5 Evaluation

In this section, we evaluate our botnet prototype from below perspectives: C&C data usage, CPU and memory usage, and the complexity of the DGA computation results.

C&C Data Usage. In our botnet design, bots use REST API to communicate with Mobile Twitter Search. Figure 6 shows our monitoring results with

Wireshark. 24 packages are found used for searching for the Twitter post and 45 packages for downloading the QR code image. In total, 2,595 bytes are used by bots in communication with Mobile Twitter Search, and 4,240 bytes are used to download the image.

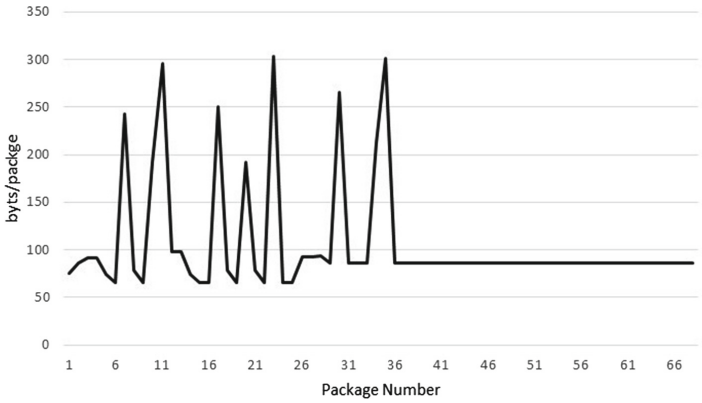

Fig. 6. Bot network traffic monitoring result

After bots complete the command pulling process from Twitter, bots upload information to the web server. In our implementation, bots send their IP addresses and device IDs to the web server. In this step, since the payload is encrypted by RSA, the size of package is larger than the original plain text. Based on Wireshark, each bot uploads a 731-byte HTTP packet to the web server. The last is the C&C communication between botmaster and bots in step ⑤. In our implementation, bots receive the TCP package from botmaster with a total length of 90 characters.

To conclude, bots use around 7-KB data to communicate with Twitter, around 700-byte data to upload their individual information to the web server, and around 400 bytes to receive the TCP package from the botmaster.

Bot CPU and Memory Usage. CPU usage has significant impact on battery life and user experience of a mobile device. Since a TCP server is created and dedicated to listen on each bot, it is necessary to estimate the CPU usage of the TCP server while bots are awaiting commands. Based on our experiment results, the TCP server thread consumes almost no CPU cycles and around 25 megabytes of memory after it is set up. The memory usage is caused by a simple Android application UI that helps track the actions of each bot, and thus we consider it as acceptable.

We also evaluated the CPU and memory usage for RSA encryption that takes place on bot side. It was found that this entire process only takes around 0.7 seconds, consuming up to 6.83% of CPU cycles and about 0.3 megabytes. In

contrast, the RSA decryption that occurs on botmaster side takes around 0.7 seconds as well, occupying up to 12.08% of CPU cycles and about 0.2 megabytes. Overall, the cipher and decipher operations are not burdens for both bot and botmaster sides.

DGA. DGA plays an important role for C&C communication and botnet robustness. The complexity of DGA output is critical to defend key cracking techniques such as dictionary attacks. In our design, we use DGA to generate random 16-character strings. To test the complexity of such strings, we ran our DGA implementation 10 million times taking three random Java positive integers as seeds. Based on our evaluation results, the DGA generation results are found made up of a total of 49 characters with no collision detected. Therefore, there are maximally 49^{16} possible DGA generation results. Based on Arora [34], the complexity of DGA algorithm is in-between of 64-bit and 128-bit AES algorithms. The evaluation proves the impossibility to exhaust all the DGA results for banning.

6 Countermeasures

Different countermeasures may be taken to defend our botnet. In this section, we discuss the countermeasures based on the perspectives of botnet detection and defense. Although the various approaches introduced below could be used against our botnet design, it is still a big challenge for defenders to fully kill a botnet like this.

6.1 Botnet Detection

As one of the most popular approaches in the field of malware detection and analysis, honeynet can be used to trap bot infections and collect information for botnet detection in real-world. If bots fall into any honeynet, the compromised devices can be detected by using behavior detection models. For instance, Gu et al. [35] proposed BotMiner to use clustering analysis of network traffic to detect suspicious behavior in a honeynet. Since the C&C protocol for our botnet is HTTP-based, monitoring the header and content of the HTTP packets may statistically reveal the botnet network traffic to defenders.

In addition to honeynet, local botnet abnormal behavior detection approaches may be leveraged to detect our botnet. The Android application store can leverage malware detection software to detect the bot-infected application before it passes their security test. For example, the tool from [36] can be used to detect and visualize traffic flowing to unexpected websites, based on monitoring application activities and building an app-specific attack-neutral activity graph.

Android users can also install and run anti-virus applications on individual devices to help detect the unexpected URL visits (such as Twitter visits in our botnet design). In addition to URL visits, monitoring the mobile application's

CPU usage and network usage can also help detect bot infections, given the fact that a bot will be commanded to launch attacks (like DDoS), which will consume CPU and network resources intensively in a burst. However, till today most customers are still reluctant to run extra software on their devices, due to the limited computing power and battery capacity of the mobile devices.

6.2 Botnet Defense

The most straightforward while efficient defense strategy specifically for our botnet design is to exhaust all the possible DGA-generated keywords and black-list them in Twitter search engine. The main issue of this method is the scale of DGA results. The aforementioned evaluation result of our implementation shows that the scale of the DGA results can be too large for defenders to completely ban them. A bigger challenge is that such a black-list requires, as the prerequisite, the success of reverse engineering the corresponding bot samples, as the prediction of DGA outputs needs to know the seeds fed to the DGA algorithm. Any variance in the DGA setting will cause the difference of DGA-generated keywords. Hence, it's almost an unfeasible task to black-list the query keywords in Twitter search engine.

To mitigate the threat of our botnet, we recommend all the OSN platforms to consider to leverage filter bubble [37] to prevent abnormal searching behavior. Filter bubble, as a technology that can be used by modern OSN design to consider peer relationships, could make the OSN post from the botmaster not searchable by others, i.e. bots, as they are not linked to botmaster in OSN. Based on our botnet design, this may prevent the bots to meet up in Twitter.

7 Conclusion

In this paper, we proposed a new OSN-based botnet design with strong robustness. In our botnet design, the botmaster orchestrates QR codes, Twitter search, and a movable web server to build dynamic C&C channels, which effectively avoid single point of failure for botnet's high recovery capability. Cryptography (DGA, AES, and RSA) is used to ensure the confidentiality of the C&C communications. Last but not least, our design takes advantage of the Tor network to achieve the botmaster's anonymity, i.e. to ensure that the botmaster is never directly exposed to the public network. To the best of our knowledge, this paper is the first to exploit QR codes as C&C communication medium in OSN-based mobile botnet.

References

1. Google announces over 2 billion monthly active devices on android. https://www.theverge.com/2017/5/17/15654454/android-reaches-2-billion-monthly-active-users
2. Eslahi, M., Rostami, M.R., Hashim, H., Tahir, N.M., Naseri, M.V.: A data collection approach for mobile botnet analysis and detection. In: The IEEE Symposium on Wireless Technology and Applications (ISWTA), pp. 199–204. IEEE, Kota Kinabalu (2014)
3. Felt, A.P., Finifter, M., Chin, E., Hanna, S., Wagner D.: A survey of mobile malware in the wild. In: The 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices, pp. 3–14. ACM, Chicago (2011)
4. Cooke, E., Jahanian, F., McPherson, D.: The zombie roundup: understanding, detecting, and disrupting botnets. In: The Steps to Reducing Unwanted Traffic on the Internet on Steps to Reducing Unwanted Traffic on the Internet Workshop, p. 6. USENIX, Cambridge (2005)
5. Bailey, M., Cooke, E., Jahanian, F., Xu, Y., Karir, M.,: A survey of botnet technology and defenses. In: The Conference for Homeland Security on Cybersecurity Applications & Technology (CATCH09), pp. 299–304. IEEE, Washington (2009)
6. Eslahi, M., Salleh, R., Anuar, N.B.: MoBots: a new generation of botnets on mobile devices and networks. In: IEEE Symposium on Computer Applications and Industrial Electronics (ISCAIE), pp. 262–266. IEEE, Kota Kinabalu (2012)
7. Malatras, A., Freyssinet, E., Beslay, L.: Mobile botnets taxonomy and challenges. In: European Intelligence and Security Informatics Conference, pp. 149–152. IEEE, Manchester (2015)
8. Dagon, D., Gu, G., Lee, C.P., Lee, W.: A taxonomy of botnet structures. In: 23rd Annual Computer Security Applications Conference, pp. 325–339. IEEE, Miami Beach (2007)
9. Krombholz, K., Frühwirt, P., Kieseberg, P., Kapsalis, I., Huber, M., Weippl, E.: QR code security: a survey of attacks and challenges for usable security. In: Tryfonas, T., Askoxylakis, I. (eds.) HAS 2014. LNCS, vol. 8533, pp. 79–90. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07620-1_8
10. Kieseberg, P., et al.: QR code security. In: 8th International Conference on Advances in Mobile Computing and Multimedia, pp. 430–435. ACM, Paris (2010)
11. Kharraz, A., Kirda, E., Robertson, W., Balzarotti, D., Francillon, A.: Optical delusions: a study of malicious QR codes in the wild. In: the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 192–203. IEEE, Atlanta (2014)
12. Cui, X., Fang, B., Yin, L., Liu, X., Zang, T.: Andbot: towards advanced mobile botnets. In: the 4th USENIX Conference on Large-scale Exploits and Emergent Threats, p. 11. USENIX, Boston (2011)
13. Singh, A., Toderici, A.H., Ross, K., Stamp, M.: Social networking for botnet command and control. Int. J. Comput. Netw. Inf. Secur. **5**, 11–17 (2013)
14. Yin, T., Zhang, Y., Li, S.: DR-SNBot: a social network-based botnet with strong destroy-resistance. In: 9th IEEE International Conference on Networking. Architecture, and Storage, pp. 191–199. IEEE, Tianjin (2014)
15. Shin, S., Gu, G.: Conficker and beyond: a large-scale empirical study. In: the 26th Annual Computer Security Applications Conference, pp. 151–160. ACM, Austin (2010)
16. Conficker's estimated economic cost? $9.1 billion. http://www.zdnet.com/article/confickers-estimated-economic-cost-9-1-billion/

17. Dingledine, R., Mathewson, N., Syverson, P.: Tor: the second- generation onion router. In: the 13th Conference on USENIX Security Symposium, p. 21. USENIX, San Diego (2004)
18. Orbot. https://www.torproject.org/docs/android.html.en
19. Dong, Y.: An Android botnet that meets up at Twitter. http://csus-dspace.calstate.edu/handle/10211.3/198844
20. Hua, J., Sakurai, K.: A SMS-based mobile botnet using flooding algorithm. In: Ardagna, C.A., Zhou, J. (eds.) WISTP 2011. LNCS, vol. 6633, pp. 264–279. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21040-2_19
21. Zeng, Y., Shin, K.G., Hu, X.: Design of SMS commanded-and- controlled and P2P-structured mobile botnets. In: The 5th ACM Conference on Security and Privacy in Wireless and Mobile Networks, pp. 137–148. ACM, Tucson (2012)
22. Faghani, M. R., Nguyen, U. T.: Socellbot: A new botnet design to infect smartphones via online social networking. In: 25th IEEE Canadian Conference on Electrical and Computer Engineering, pp. 1–5. IEEE, Montreal (2012)
23. Nagaraja, S., Houmansadr, A., Piyawongwisal, P., Singh, V., Agarwal, P., Borisov, N.: Stegobot: a covert social network botnet. In: Filler, T., Pevný, T., Craver, S., Ker, A. (eds.) IH 2011. LNCS, vol. 6958, pp. 299–313. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24178-9_21
24. Compagno, A., Conti, M., Lain, D., Lovisotto, G., Mancini, L.V.: Boten ELISA: A novel approach for botnet C&C in online social networks. In: IEEE Conference on Communications and Network Security, pp. 74–82. IEEE, Florence (2015)
25. Koobface: inside a crimeware network. https://www.nartv.org/2010/11/12/koobface-inside-a-crimeware-network/
26. Thomas, K., Nicol, D.M.: The Koobface botnet and the rise of social malware. In: The 5th International Conference on Malicious and Unwanted Software, pp. 63–70. IEEE, Nancy (2010)
27. Web 2.0 Botnet Evolution Koobface Revisited. https://www.trendmicro.de/cloud-content/us/pdfs/security-intelligence/white-papers/wp_web-2-0-botnet-evolution-koobface.pdf
28. Chasing Advanced Persistent Threats (APT). https://www.secureworks.com/research/chasing_apt
29. Yao, H., Shin, D.: Towards preventing QR code based attacks on android phone using security warnings. In: The 8th ACM SIGSAC Symposium on Information, Computer and Communications Security, pp. 341–346. ACM, Hangzhou (2013)
30. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM **21**, 1–5 (2012)
31. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Springer, New York (2013). https://doi.org/10.1007/978-3-662-04722-4
32. Mobile twitter search. https://mobile.twitter.com/search
33. Botnet prototype demonstration. https://youtu.be/LkfYa4OgvYI
34. How secure is AES against brute force attacks. https://www.eetimes.com/document.asp?docid=1279619
35. Gu, G., Perdisci, R., Zhang, J., Lee, W.: BotMiner: clustering analysis of network traffic for protocol and structure-independent botnet detection. In: The 17th USENIX Security Symposium, pp. 1–5. USENIX, San Jose (2008)
36. Gopalan, S., Kulkarni, A., Shah, A., Dai, J., Ouyang, J., Muyan-Ozcelik, P., Sun, X.: Dont be surprised: i see your mobile app stealing your data. In: ICNC 2018-Mobile Computing & Vehicle Communications Symposium, to appear. ICNC, Hawaii (2018)
37. Filter bubble. https://www.techopedia.com/definition/28556/filter-bubble