Cloud-Based Push-Styled Mobile Botnets: A Case Study of Exploiting the Cloud to Device Messaging Service

Shuang Zhao[1,2], Patrick P. C. Lee[3], John C. S. Lui[3], Xiaohong Guan[1], Xiaobo Ma[1], Jing Tao[1]
[1]School of Electronic & Information Eng., Xi'an Jiatong University, China
[2]Institute of Information Engineering, Chinese Academy of Sciences, China
[3]Dept of Computer Science & Engineering, The Chinese University of Hong Kong, Hong Kong
dflower.zs@gmail.com, {pclee,cslui}@cse.cuhk.edu.hk,
xhguan@xjtu.edu.cn, superxiaoboma@gmail.com, jtao@xjtu.edu.cn

ABSTRACT

Given the popularity of smartphones and mobile devices, mobile botnets are becoming an emerging threat to users and network operators. We propose a *new* form of cloud-based push-styled mobile botnets that exploits today's push notification services as a means of command dissemination. To motivate its practicality, we present a new command and control (C&C) channel using Google's Cloud to Device Messaging (C2DM) service, and develop a *C2DM botnet* specifically for the Android platform. We present strategies to enhance its *scalability* to large botnet coverage and its *resilience* against service disruption. We prototype a C2DM botnet, and perform evaluation to show that the C2DM botnet is *stealthy* in generating heartbeat and command traffic, *resource-efficient* in bandwidth and power consumptions, and *controllable* in quickly delivering a command to all bots. We also discuss how one may deploy a C2DM botnet, and demonstrate its feasibility in launching an SMS-Spam-and-Click attack. Lastly, we discuss how to generalize the design to other platforms, such as iOS or Window-based systems, and recommend possible defense methods. Given the wide adoption of push notification services, we believe that this type of mobile botnets requires special attention from our community.

1. INTRODUCTION

With the advent of mobile Internet access, we have seen significant technological advancement of smartphones and mobile devices. This provides a fertile ground for hackers to realize *botnets*[1] in a mobile network. In recent years, we have seen many real-life examples of mobile botnets. In 2009, a mobile botnet SymbOS.Yxes [1] was discovered in the Symbian platform, which used the conventional HTTP-based C&C channel for communication. In December 2010, the first iPhone botnet Ikee.B [2] was found in the wild. It targeted jailbroken iPhones and pulled commands from a HTTP server. In the same year, the first Android botnet named GEINIMI [3] emerged. It also used a HTTP-based C&C channel for command dissemination. In 2011, a Short Messaging Service (SMS)-based mobile botnet named ZeuS [4] was found in the Blackberry, Symbian, and Windows Mobile platforms. It used an SMS-based C&C channel to communicate with the botmaster. In September 2011, an Android bot called *AnserverBot* was identified and it was the first Android bot that used public blogs as C&C servers [5]. In late January 2012, an Android botnet disguised as the game application "Madden NFL 12" [6] and used the Internet Relay Chat (IRC) as its C&C channel. All these incidents indicate that mobile botnets have become an emerging threat for users and network operators.

From a botmaster's perspective, how to deploy a stealthy and robust mobile botnet is an interesting issue; from an operator's perspective, understanding the deployment strategy of a mobile botnet is critical for defending against malicious attacks on an operational network. Our key observation is that many smartphone platforms provide the *push notification service*, which comprises a cloud of push-based messaging servers that are responsible for relaying messages from application servers to mobile applications. There are various deployments of this service in different platforms, such as Google's Cloud to Device Messaging (C2DM) service for Android [7], Apple's Push Notification Service (APNS) [8] for iOS, Microsoft's Push Notification Service (MPNS) for Windows Mobile [9], Blackberry's Push Service (BPS) [10], and Nokia's Notifications API (NNA) for Symbian and Meego devices [11]. Architectures of these push notification services have one common feature: application servers first send a notification message with an intended receiver (or the target mobile device) to one of the cloud-based messaging servers, which then *pushes* the message to the target mobile device. The push notification service eliminates the needs of application servers to keep track of the state of a mobile device (i.e., active or offline). Furthermore, mobile devices do not need to periodically probe the application servers for messages, thereby reducing the workloads of the application servers. While the push notification service simplifies the mobile application development, it can also be exploited by attackers in building a *highly potent* and *stealthy* mobile botnet compared to traditional HTTP, IRC, or SMS botnets.

This paper aims to expose such kind of potential attack scenario. As a proof of concept, we consider Google's C2DM service for the Android platform, and realize a cloud-based push-styled mobile botnet using the push notification service as a C&C channel. We named it as *C2DM botnet*[2], which involves no direct communication between the botmaster and various bots, but instead ex-

[1] A *botnet* is a network of compromised computers called *bots* that are remotely controlled by a *botmaster*.

[2] We note that Google's C2DM service is deprecated on June 26, 2012, after the paper is submitted for review. Nevertheless, the fundamental design of the C2DM botnet remains applicable to the next-generation C2DM service. We discuss this issue in §7.

ACSAC '12 Dec. 3-7, 2012, Orlando, Florida USA

ploits Google's C2DM service as a relay. The botmaster can disseminate probes and commands to the bots via the C2DM service, and the botnet traffic can be "hidden" within the C2DM traffic of other legitimate mobile applications. This makes the C2DM botnet stealthy. In summary, we make several contributions in motivating the practicality of this new form of mobile botnets:

- **Design and implementation of a cloud-based push-styled mobile botnet.** We build a C2DM botnet for the Android platform using Google's C2DM push notification service as the C&C channel. We propose a multiple-username strategy to enhance its *scalability* to a large botnet size, and propose how to make the botnet *resilient* against service disruption due to account unregistration.
- **Performance evaluation of the C2DM botnet.** We extensively evaluate the C2DM botnet design, and show its (i) *stealthiness* in generating heartbeat and command traffic by hiding its existence under other legitimate C2DM traffic, (ii) *resource efficiency* in bandwidth and power consumptions compared to traditional IRC and HTTP bots, and (iii) *controllability* in disseminating commands to all bots within a short period of time.
- **Demonstration of C2DM bot propagation and attack.** We show how hackers can infect and propagate a C2DM bot using legitimate mobile applications, and show how this botnet can launch an effective SMS-Spam-and-Click attack.
- **Extension to other mobile platforms.** We discuss how the C2DM botnet can be extended to other platforms, such as iOS and Microsoft Windows.
- **Recommendation of potential defense strategies.** We suggest detection and defense strategies against this underlying threat.

The rest of the paper proceeds as follows. §2 reviews related work. §3 presents an overview for the C2DM service. §4 presents the design and implementation of a C2DM botnet along with strategies that enhance its scalability and fault tolerance. §5 presents evaluation results for the C2DM botnet. §6 discusses how an attacker may propagate a C2DM bot and feasibly launch an attack. §7 shows how to generalize the C2DM botnet design to other platforms. §8 discusses potential defense strategies, and §9 concludes.

2. RELATED WORK

There are a number of studies in the literature on mobile botnets. Traynor et al. [12] study how a mobile botnet launches a DDoS attack against a cellular network core. They show that a small-size mobile botnet is sufficient to cause nationwide outages. Singh et al. [13] propose a mobile botnet using Bluetooth as a C&C channel. The commands are disseminated from one bot to another within the radio range of bluetooth. Their experiments show that commands from the botmaster can reach about 2/3 of bots in 24 hours. Xiang et al. [14] discuss the single point of failure problem in traditional centralized HTTP botnets. They propose a mobile botnet named URL Flux to make a traditional HTTP-based botnet more resilient against existing defense solutions.

Short Message Service (SMS) can also be used as a C&C medium for mobile botnets. Zeng et al. [15] use SMS messages for C&C to build mobile botnets. They leverage on the P2P topological structure in the botnet to reduce the number of SMS messages being sent and shorten the time delay for delivering SMS commands. Each bot needs to send SMS messages to its neighbors in order to join the P2P network and forward commands. Geng et al. [16] propose an SMS-based mobile botnet, in which they divide bots into normal bots and regional bot servers to build a P2P network in order to reduce the forward and search consumptions, and enhance the robustness of the botnet. Hua et al. [17] design a SMS-based mobile botnet and evaluate its construction under different topologies. Their simulations show that in a botnet containing 20,000 bots, a command from the botmaster could be covertly disseminated to over 90% of all bots, and each bot only needs to send no more than four SMS messages during the dissemination. Weidman [18] design a transparent C&C channel for a SMS-based botnet, in which the bot can intercept and read commands from incoming SMS messages before the messages are presented to users. Mulliner and Seifert [19] combine the Kademlia P2P network with an SMS-HTTP hybrid approach as the C&C channel for a mobile botnet, in which the communication is split into HTTP and SMS parts. We want to emphasize that one disadvantage of SMS-based botnets is in the cost incurred when sending commands via SMS, especially if the botnet is of large scale. Also, the use of SMS may eventually alert users and mobile operators since sending/receiving SMS messages may incur payment to mobile phone operators. In our proposed botnet, bots communicate with the C&C servers via the Internet, and this is much cheaper and stealthier.

Several botnet detection methods (e.g., [20, 21, 22, 23]) used in traditional wireline networks are also applicable to detect centralized IRC- and HTTP-based mobile botnets. Specifically for mobile botnets, Vural et al. [24] use a forensics-based approach: they first model normal activities of mobile users, and use the model results to identify the abnormal activities of malware or botnets.

3. OVERVIEW OF C2DM

We now describe Google's cloud-based push-styled messaging service, or the *Cloud to Device Messaging* (C2DM) service (as illustrated in Figure 1). It enables third-party developers to send push notifications to their mobile applications on Android devices.

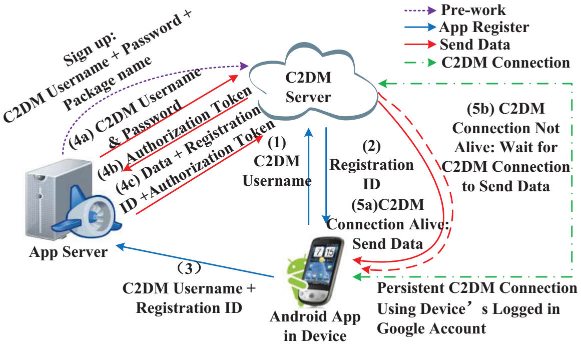


Figure 1: C2DM architecture and its workflow.

To bootstrap the C2DM service, the application developer has to first sign up for an account. This is done by providing the C2DM server with the following: *C2DM username*, *password*, and the *package name* of the mobile application. Once the account is established, the developer can embed the C2DM username in the mobile application, and distribute the application, say, via the official Android Market or other third-party marketplaces.

We now elaborate the workflow of the C2DM service as depicted in Figure 1. When the mobile application is first launched in a mobile device, it will perform the following steps.

- **Step (1):** The mobile application registers itself to one of the C2DM servers using the C2DM username, which was provided by the application developer, as well as the *device ID*, which uniquely identifies the Android device that hosts the application.
- **Step (2):** The C2DM server provides a unique *registration ID* to the mobile application. The registration ID is a byte string that enables the C2DM server to identify the application running on a specific Android device.

• **Step (3):** The mobile application sends this registration ID, together with its C2DM username, to the application server, which will then record this registration ID in its database.

• **Step (4):** When the application server needs to send data to a mobile device, it sends the C2DM username and password to the C2DM server (Step (4a)), and gets an authorization token if the username and password are valid (Step (4b)). The authorization token will be used to notify a set of mobile devices in the database. The application server then sends a C2DM request, which contains the notification message, the registration ID of a mobile application, and the authorization token to the C2DM server (Step (4c)). Note that the message in Step (4c) is sent on a *per-device* basis. Thus, if there are k devices that need the notification, then the application server will send k messages to the C2DM server.

• **Step (5):** Upon receiving the message, the C2DM server looks for the specific Android device based on the registration ID. If the C2DM connection of that device is alive, then the C2DM server will send the notification message to the mobile application on that device (Step (5a)); if the mobile device is disconnected, then the C2DM server will store the notification message, and send the message to the application on the mobile device when the device re-establishes its connection with the C2DM server (Step (5b)). Note that Step (5) implies that there is a *persistent C2DM connection* between the C2DM server and the Android device that subscribes to the C2DM service.

Google's C2DM service provides a flexible solution for developers to send lightweight messages to mobile applications without requiring mobile devices to connect to their servers periodically to pull for messages. Google manages the storage and forwarding of messages. Thus, it simplifies the mobile application design, and reduces the network traffic and battery usage of mobile devices. C2DM maintains a persistent TCP connection for each mobile device with a C2DM server, and the time interval for each heartbeat message of this persistent connection is around 15-30 minutes depending on the device state (i.e., ACTIVIT, IDLE, SYNC, NOSYNC). When a new C2DM message arrives, the mobile device will wake up the mobile application to receive the message.

The C2DM service is available for a device running on Android 2.2 or higher versions. The device must have the Market (now Google changes its name to Play Market) application installed and at least one logged-in Google account [7]. Market is one of the factory-installed applications in the Android platform, along with other *popular* mobile applications like Google Maps, Gmail, etc. Users are required to log in with a Google account and enable the C2DM service when using these applications. There are also many other popular applications which rely on the C2DM service, such as Instagram, Facebook for Android, and LINE. Thus, we expect that the C2DM service is enabled in majority of Android devices.

4. DESIGN OF A C2DM BOTNET

This section presents the design and implementation of a C2DM botnet. Our design is based on the official and open C2DM architecture. To ease our presentation, we first discuss a baseline architecture for a C2DM botnet. Then we present an enhanced architecture that has a stronger stealthiness property. Finally, we address the scalability and fault-tolerance issues.

4.1 Baseline C2DM Botnet Architecture

We first describe the baseline architecture of a C2DM botnet, as shown in Figure 2. Before building a C2DM botnet, the botmaster first bootstraps a C2DM service as for other normal C2DM applications (see §3). Then the steps of how a new bot joins the C2DM botnet and how the botmaster disseminates commands can be done as follows (referring to Figure 2).

• **Bot registration** (Steps (1) to (3)): When a new bot joins a C2DM botnet, it first registers itself to one of the Google C2DM servers with the following information: (i) the package name and the C2DM username, both of which are embedded by the botmaster in the distributed malware package, and (ii) the Google's account ID of the mobile device. If the registration is successful, then the C2DM server will return a unique registration ID to the bot. Finally, the bot sends the registration ID and the C2DM username to the botmaster's C&C server, which will then record this registration ID in its database for future command dissemination.

• **Command dissemination** (Steps (a) to (c)): The botmaster disseminates commands to all registered mobile bots via a C&C server. The C&C server first authenticates itself to the C2DM server by using its Google's account ID and password. After authentication, the C2DM server will return an authorization token to the C&C server, which then constructs a C2DM request for *each* mobile device (or bot). The request contains the command by the botmaster, the registration ID of the bot, and the authorization token. The C&C server then sends the request to the C2DM server. Based on the registration ID, the C2DM server will push the message to the bot. Note that the *C2DM service only allows one registration ID in each C2DM request*. This implies that sending commands to multiple bots requires the C&C server to send multiple requests. We will discuss how to scale up a botnet and control the message dissemination in §4.3, and how to control message delivery in §5.3.

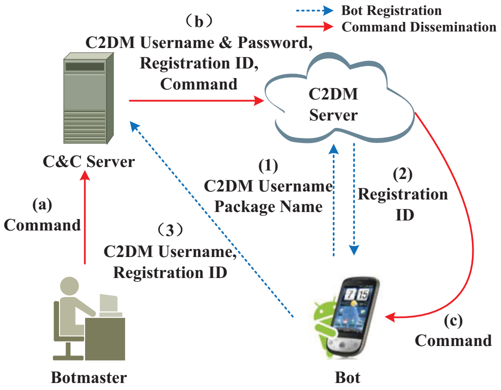


Figure 2: Baseline architecture of a C2DM botnet.

4.2 Enhanced C2DM Botnet Architecture

In the baseline architecture, each bot needs to directly send its registration ID to the C&C server. This increases the possibility of being detected and reveal the identity of the C&C server. To improve the stealthiness, the enhanced architecture eliminates the direct communication between a bot and its botmaster.

Before building a C2DM botnet, the botmaster needs to sign up *two* C2DM accounts: (i) C2DM username_M, which is used by the botmaster's C&C server to send messages to its bots, and (ii) C2DM username_B, which is used by the bots to send messages to the C&C server. The botmaster needs to register itself to a C2DM server using C2DM username_B to obtain a registration ID, which will later be used by the bots to send C2DM messages to the C&C server. We assume that C2DM username_M, C2DM username_B, and the botmaster's registration IDs are all embedded in the malware package.

Figure 3 depicts the enhanced architecture. The steps of bot registration and command dissemination are revised as follows.

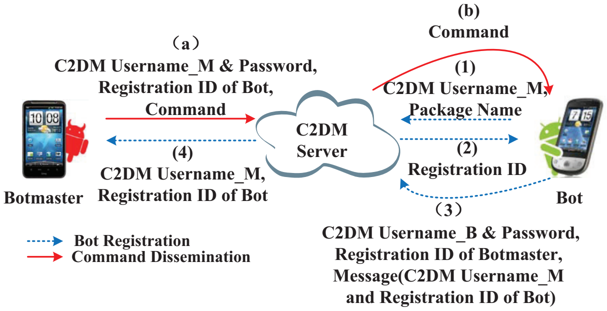


Figure 3: Enhanced architecture of a C2DM botnet.

• **Bot registration** (Steps (1) to (4)). When a new bot joins a C2DM botnet, it registers itself to one of the C2DM servers using its package name and C2DM username_M. If the registration is successful, then the C2DM server will return a unique registration ID to the bot. Then the bot uses the account of C2DM username_B and the botmaster's registration ID (both of which are embedded in the malware package) to send its own registration ID and C2DM username_M to the botmaster, which records the information for later command dissemination.

• **Command dissemination** (Steps (a) to (b)). To disseminate commands to all registered mobile bots, the C&C server constructs a C2DM request for *each* bot. Unlike the baseline architecture, this request now contains the account C2DM username_M, the registration ID of the bot, and the command. The C&C server sends the request to the C2DM server, which will then be pushed to the corresponding bot based on the registration ID.

Remark on stealthiness and reliability. One major advantage of the enhanced architecture is that during the bot registration period, the bot sends its registration ID and C2DM username to the C&C server via the C2DM server. Thus, it is stealthier than the baseline architecture. However, one shortcoming of the enhanced architecture is that it relies on the botmaster's registration ID for communication, and Google may revoke any registration ID that is maliciously used. Although our experience is that Google seldom explicitly un-registers registration IDs, there is no guarantee that a registration ID will remain valid permanently. If the registration ID is revoked, then the bots cannot send messages to the botmaster. We will discuss how to overcome this issue in §4.4.

4.3 Scaling up the C2DM Botnet

As mentioned in §4.1, each C2DM request can only be sent to a single device. Also, some quota limits may be posed on the rate of push messages that can be delivered. For example, Google's C2DM service limits each account to send no more than 200,000 push messages per day [7]. Furthermore, when signing up for a new C2DM service, one needs to specify the estimated peak queries per second (QPS), which is a short-term push rate permitted for an application. In C2DM, there are four choices for QPS: "0 - 5", "6 - 10", "11-100" and "> 100". To ensure that the C2DM botnet is stealthy, we assume that the QPS of a C2DM botnet is less than 100 to avoid drawing Google's attention.

For a small-scale C2DM botnet, 200,000 push messages per account per day should suffice to disseminate commands to all the bots, and it takes only a short duration to disseminate commands to the whole botnet under our QPS requirement. We will discuss the controllability and estimate the time needed to disseminate commands to most of the bots in §5.3. However, using one C2DM account to maintain a botnet becomes problematic if (i) the botnet is of large size, or (ii) the botnet needs to send more than 200,000 push messages per day.

To build a large-scale C2DM botnet, we propose to use *multiple C2DM usernames* to decompose a large botnet into several smaller subnets. Each of these subnets uses a unique C2DM username to communicate with its own bots. We elaborate how to incorporate multiple C2DM usernames into a C2DM botnet, using the enhanced architecture as an example, as shown in Figure 4. Here, a bot joins the botnet using a *two-phase* C2DM registration process. The bot first registers to a C2DM server using an initial username, and then sends its own registration ID to the botmaster via the C2DM service (Step (1)). The botmaster, upon receiving the C2DM message, chooses one of the pre-defined C2DM usernames and sends it to the bot via C2DM (Steps (2) to (4)). Then the bot re-registers to a C2DM server with the new C2DM username and sends the new registration ID to the botmaster via C2DM (Steps (5) to (6)). The botmaster will then use the new registration ID to send commands to the bot.

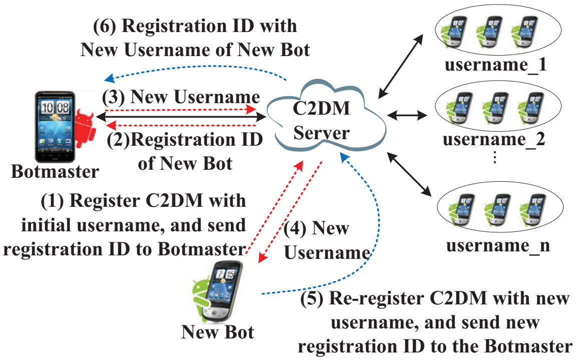


Figure 4: Registration in a large-scale enhanced C2DM botnet.

4.4 Handling Account Un-registration

Each botnet must deal with the single-point-of-failure problem. We now discuss how a C2DM botnet can self-configure in case a core component fails. In a C2DM botnet, we argue that the C&C server, though important, is not the core factor to consider due to two reasons. First, bots communicate with the C&C server only once during their registration. Once a botnet is built, bots will not directly communicate with the C&C server, so it is quite stealthy. Second, one may use well-known social networking websites such as Twitter, Facebook to set up a C&C server to enhance the stealthiness [14, 25].

On the other hand, we need to ensure the robustness of the C2DM service in a C2DM botnet. If the C2DM service is unavailable, then the communication between the botmaster and bots will be cut off. One way this may happen is that the C2DM service used by the bot is *banned* and *un-registered* by Google. When a bot registers to a C2DM server, it is required to provide both its package name and C2DM username. In our experiments, we find that when we sign up for a C2DM service, the package name need not be unique. In other words, one can sign up for the C2DM service using the same package name as other existing applications. This implies that Google is not likely to ban specific package names from the C2DM service, as it may unexpectedly ban other legitimate services. On the other hand, the C2DM username may be banned from any C2DM service, and this can shut down the communication of the entire C2DM botnet.

To overcome service disruption due to the un-registration of the C2DM username, one can again leverage the multiple-username strategy proposed in §4.3. The main idea of such a strategy is to set up several backup C&C servers, and if a bot has not received any messages from the botmaster for a pre-defined duration (e.g., one week), then it will communicate with one of the backup C&C servers and query whether it needs to change its C2DM username. If it gets a new username, it will use the new username to register to the C2DM service again and sends the new registration ID to the C&C server in order to re-join the botnet. We point out that the backup C&C server also possesses the stealthiness property because bots seldom communicate with it.

5. EVALUATION

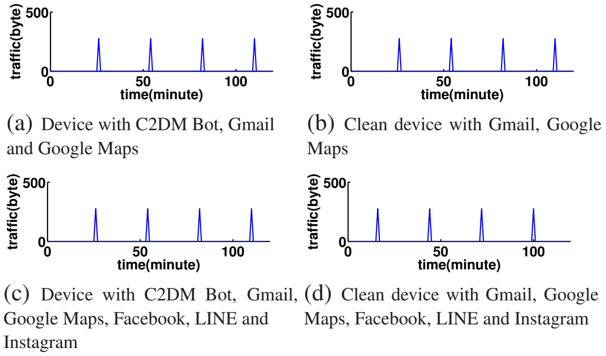
In the previous section, we showed that the C2DM botnet communication protocol possesses the stealthiness property: there is no *direct* communication between the botmaster (or C&C server) and bots. In fact, all communication and command dissemination are carried out via the C2DM service. In this section, we take a closer examination on the stealthiness property by measuring (a) the heartbeat overhead of maintaining a C2DM persistent connection; (b) the capability of hiding command dissemination among legitimate C2DM traffic, and (c) resource (i.e., bandwidth and power) consumption on mobile devices. We also explore the *controllability* of a C2DM botnet: if a C&C server wants to disseminate a command, then what is the *minimum time* needed for the botmaster to claim, with a high probability, that all bots receive the command?

5.1 Stealthiness in Control/Data Plane

We first examine the stealthiness of a C2DM botnet in terms of network traffic. System operators can perform online/offline network traffic analysis to detect a botnet [22, 23], and examine the following *suspicious behaviors*: (i) connecting to some unauthorized servers (i.e., C&C servers) using domain names, URLs, or IP addresses, and (ii) communicating with one or more servers with certain periodic patterns or with an abnormally high frequency. For example, a traditional HTTP bot will connect to a C&C server with an unauthorized domain name or IP address, and periodically pull commands from the C&C server. These periodic connections can expose the presence of a bot. An advanced HTTP bot may use URL flux [14], which uses authorized domain names such as "twitter.com" to pull commands, but frequent connections can also be classified as abnormal behavior.

We expect that a C2DM botnet has high stealthiness with the following intuition. In the baseline architecture (see §4.1), a bot connects to a C&C server only *once*, i.e., when sending its registration ID. Then during the keep-alive period, the bot will never communicate with the C&C server. In the enhanced architecture (see §4.2), a bot connects to a C&C server only when the registration ID is unregistered (see §4.4). Thus, the probability for a C2DM botnet to expose a bot or the C&C server is kept at minimum. In the following, we examine the stealthiness of a C2DM botnet by measuring the network traffic in both *control and data planes*. We implement a C2DM bot in the Android emulator [26], while the measurement results are also applicable for real Android phone devices.

Stealthiness of heartbeat traffic. We first look into the control plane, and focus on the periodic heartbeat messages. Note that if an Android phone enables any legitimate C2DM service, then it will connect to a C2DM server with a persistent TCP connection and sends periodic heartbeat messages to a C2DM server to check for any new push message. We compare the C2DM heartbeat traffic of two Android phones, one being installed with a C2DM bot and another being a clean Android phone. Both Android phones are installed with a number of legitimate applications that require the C2DM service. Figure 5 shows the traffic patterns of heartbeat messages under different settings, where the x-axis is the time (with unit in minutes) and the y-axis is the traffic volume (with unit in bytes). Figures 5(a) and 5(c) show the heartbeat traffic patterns for a phone with a C2DM bot, while Figures 5(b) and 5(d) show the heartbeat traffic patterns for clean a mobile phone, with three or five legitimate C2DM applications, respectively. We observe that the traffic patterns of these settings are *almost identical*, since all legitimate mobile applications which use the C2DM service share the same persistent TCP connection per device. This allows a C2DM bot to hide itself under legitimate C2DM heartbeat traffic. Since a C2DM bot relies on the existing C2DM service of the mobile phone, if the C2DM service is not enabled, then the bot will be dormant and will not generate any C2DM heartbeat traffic.



(a) Device with C2DM Bot, Gmail and Google Maps

(b) Clean device with Gmail, Google Maps

(c) Device with C2DM Bot, Gmail, Google Maps, Facebook, LINE and Instagram

(d) Clean device with Gmail, Google Maps, Facebook, LINE and Instagram

Figure 5: C2DM traffic: (a) and (c) are traffic for a C2DM bot; (b) and (d) are traffic for clean devices, with 3 or 5 apps.

Stealthiness of command dissemination. We examine the stealthiness of a C2DM botnet in the data plane. Note that many Android phones typically come with a number of *pre-installed* applications such as Gmail, Google Maps, Google Play, or some *popular* applications like Facebook, LINE, Whatsapp, etc. All these applications use C2DM services. Upon installation, the C2DM bot knows which of these legitimate C2DM applications are installed in the mobile phone, and this provides valuable information to enhance its stealthiness. For example, in [27], authors indicate the interarrival time of Gmail messages is about 0.53 hours. Thus, if a botmaster disseminates commands with an average interarrival time longer than those of the legitimate C2DM applications, then it is easy to hide the existence of a C2DM bot.

We perform experiments by installing a C2DM bot into an Android phone that is pre-installed with Gmail and Google Maps. We then measure the data traffic generated by the bot and the legitimate applications Gmail and Google Maps as follows. During the experiment, the C&C server sends commands to the bot. In the same measurement period, the phone also receives several email notifications from Gmail and check-in notifications from Google Maps. Figure 6 shows the data traffic generated by the bot and the legitimate applications in one particular measurement. We observe that the C2DM botnet traffic only occupies less than 20% of the overall data traffic. The size of each C2DM botnet traffic burst is also very small, in the range of 200-400 bytes. This shows the stealthiness property of a C2DM botnet in the data plane.

5.2 Efficiency in Resource Consumption

One major difference between a conventional PC-based botnet and a mobile botnet is that the latter needs to consider resource consumption, especially for bandwidth and power, because most mobile phones have limited network and battery capacities. If a mobile bot significantly consumes bandwidth or battery resources, then it may draw unnecessary attention of users and reveal the pres-

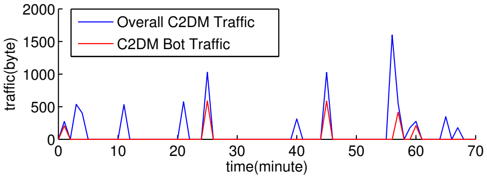


Figure 6: C2DM traffic generated by a C2DM bot and legitimate applications.

ence of a bot. We now evaluate and compare the bandwidth and power consumptions of a C2DM botnet with other mobile botnets that use traditional IRC and HTTP as C&C channels.

5.2.1 *Bandwidth Consumption*

We install a C2DM bot, an IRC bot and three HTTP bots (with time intervals for pulling commands from the C&C server as 5, 10, and 30 minutes) into five different Android phone emulators. We then measure the traffic consumption of each Android phone for one hour. To do a fair comparison, we compare the overhead in the control plane (in other words, we assume these bots have the same bandwidth consumption in disseminating commands). Thus, the traffic consumption of each bot depends on two factors: (a) the time interval of heartbeat connections, and (b) the volume of the traffic of each heartbeat connection. Figure 7 depicts the heartbeat traffic of these three types of bots.

• **C2DM bot:** The time interval of the heartbeat connections of the C2DM service is in the range of 15 to 30 minutes depending on the state of the mobile phone. In our experiment, the time interval is around 28 minutes. The traffic of each heartbeat message is between 250 and 300 bytes. Note that if the phone has are other legitimate applications which also use the C2DM service, then the bot will use the existing heartbeat connection and will not generate any extra heartbeat traffic.

• **IRC bot:** The IRC bot uses the *ping-pong mechanism* to keep alive, such that it sends a ping request to the C&C server and waits for the response from the C&C server. This can be treated as the heartbeat connection [28]. The time interval of the ping-pong conversations can be customized and is usually between 30 and 600 seconds as set in most of today's IRC server software. Note that the ping-pong interval cannot be too large, or it will impose heavy burden on the IRC server to maintain many persistent connections for a long time. In our experiment, the IRC bot is implemented based on PircBot [29], and the C&C server is built using the open-source IRC server software named beware-ircd [30] with default settings, where the ping-pong interval is 90 seconds and the traffic generated by each ping-pong conversation is around 200 bytes.

• **HTTP bot:** Unlike C2DM and IRC bots, a HTTP bot does not use any persistent TCP connection to keep alive with the C&C server, so it cannot receive commands instantly and has to connect to the C&C server from time to time to pull for commands. We regard this periodic pull-command connection as its heartbeat connection. The traffic of each pull-command connection is about 1000 bytes even when there is no command. This size is much larger than those of C2DM and IRC bots because the HTTP bot has to re-establish a TCP connection each time and transmit packets to complete the TCP 3-way handshake. HTTP packets are also larger than TCP packets in general due to the additional application-layer payload. In addition, the time interval of the pull-command connections affects the average delay of command dissemination. For example, if the time interval for pulling commands is set to X minutes, then the average time delay for each bot to receive a new command will be $X/2$ minutes. In our experiment, we set the bot to periodically retrieve commands from the C&C server using HTTP requests to imitate the pull-command behaviors of a HTTP bot.

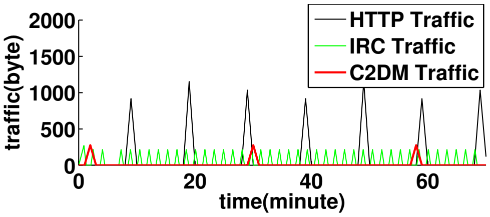


Figure 7: Heartbeat bandwidth consumption for C2DM, IRC and HTTP bots.

Figure 7 shows the result with the pull-command interval set to 10 minutes. We summarize the results here. The average traffic rate of the C2DM heartbeat connections is 900 bytes/hour, while those of the IRC and HTTP heartbeat connections are 5,760 bytes/hour (i.e., 6.4 times) and 7,200 bytes/hour (i.e., 8 times), respectively. This shows that a C2DM bot generates significantly less heartbeat traffic and hence bandwidth resources.

5.2.2 *Power Consumption*

We now evaluate and compare the power consumptions of C2DM, IRC, and HTTP bots. Our evaluation is built on [31], which propose power estimation models for different components on the basis of an Android HTC Dream phones. Here, we focus on the power consumption in the *communication* component, including WiFi and 3G, during the heartbeat period of each type of bots.

WiFi power consumption. To measure the WiFi power consumption, we install each bot in an Android phone emulator running on a PC, which is connected to a C&C server over a university WiFi network. We install Wireshark on the same PC, and capture all packets of the emulator in order to calculate the transmission time.

Figure 8 shows the power states of WiFi transmissions in an Android HTC phone. The WiFi power consumption depends on the number of packets sent/received per second, while the packet size has limited influence [31]. When a phone connects to a WiFi network and stays idle, it is in the "low" power state. When it sends/receives data, it switches to either the "ltransmit" or "htransmit" state, according to the transmission speed of the packets; when it finishes sending/receiving data, it will switch back to the previous state. In our experiment, the maximum packet speeds of all types of bots are below 15 packets per second, so the only involved power states during our experiments are "ltransmit" and "low" power states, as shown in Figure 8.

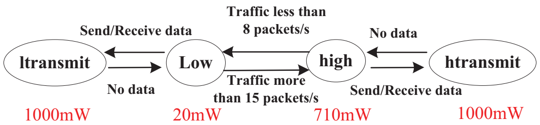


Figure 8: Wi-Fi power states of Android HTC Dream [31].

We calculate the power consumption (in Joules) during heartbeat transmission in one hour of each bot based on Equation (1), assuming that the total time span of transmitting packets[3] in one hour is T_l and the power of the "ltransmit" state is P_l:

$$W = P_l \cdot T_l. \qquad (1)$$

[3] We do not consider the power consumption of the "low" power state since every phone which connects to the Internet with WiFi will stay in the "low" power state while idle.

Suppose that it takes time t (in seconds) for a bot to transmit all packets in each heartbeat connection. Let the heartbeat interval be λ (in seconds). We have:

$$T_l = t \cdot 3600 \cdot \lambda^{-1}. \tag{2}$$

Thus, we get:

$$W = P_l \cdot t \cdot 3600 \cdot \lambda^{-1}. \tag{3}$$

Table 1 shows the WiFi power consumption for heartbeat transmission of each bot in one hour, such that the inputs are based on our measurements (some results are obtained in §5.2.1). Note that for the HTTP bot, its packet transmission time t for each heartbeat connection is longer than those of C2DM and IRC bots, as it needs to set up a TCP connection with 3-way handshake. From Table 1, we observe that under the WiFi power model, the C2DM bot consumes the *least* amount of power resources compared to HTTP and IRC bots.

Type of Bot	t (in sec)	λ (in sec)	W (in mJ)
C2DM	0.1	1680	214.3
IRC	0.1	90	4000.0
HTTP (5min)	0.3	300	3600.0
HTTP (10min)	0.3	600	1800.0
HTTP (30min)	0.3	1800	600.0

Table 1: WiFi power consumption of each bot in one hour.

3G power consumption. To measure the 3G power consumption, we install each bot on a real Android phone that is connected to a C&C server over an operational 3G data network. We use tcpdump to capture all data packets within the Android phone.

Figure 9 shows the power states of 3G transmissions of an Android HTC phone. The 3G power consumption depends on the traffic volume. When the state is IDLE, the phone cannot send/receive any data through 3G, and it consumes almost no power. In the CELL_FACH state, it can send/receive a few hundred bytes of data per second, and waits for 6 seconds before switching back to the IDLE state. If the traffic rate is much larger than the transmission speed of the CELL_FACH state, then the power state will enter the CELL_DCH state, and the phone will wait for 4 seconds before switching back to the CELL_FACH state. The CELL_DCH has a higher transmission speed and higher power consumption than the CELL_DCH state.

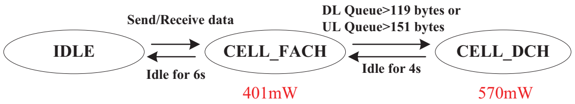


Figure 9: 3G power states of an Android HTC Dream [31].

We calculate the 3G power consumption during heartbeat transmission in one hour of each bot based on Equation (4), assuming that the power consumption in the IDLE state is zero, the data transmission time in CELL_FACH is T_f, the idle time in CELL_FACH is T_{fi}, the data transmission time in CELL_DCH is T_d, the idle time in CELL_DCH is T_{di}, the power in CELL_FACH is P_f, and the power in CELL_DCH is P_d:

$$W = P_f \cdot (T_f + T_{fi}) + P_d \cdot (T_d + T_{di}). \tag{4}$$

As in the WiFi analysis, we let t (in seconds) be the packet transmission time in each heartbeat connection, and λ (in seconds) be the interval of keep-alive connections. For C2DM and IRC bots, their heartbeat traffic volumes are generally small (see §5.2.1), so they remain in the CELL_FACH state (i.e., low power) during the heartbeat connections. Thus, the 3G power consumptions for C2DM and IRC bots are:

$$W = P_f \cdot (t + 6) \cdot 3600 \cdot \lambda^{-1}. \tag{5}$$

On the other hand, for the HTTP bot, its heartbeat traffic volume is larger (see §5.2.1), and it will stay in the CELL_FACH state (i.e., high power). Thus, the 3G power consumption of the HTTP bot is:

$$W = P_f \cdot 6 \cdot 3600 \cdot \lambda^{-1} + P_d \cdot (t + 4) \cdot 3600 \cdot \lambda^{-1}. \tag{6}$$

Table 2 shows the 3G power consumption for heartbeat transmission in one hour for each type of bots. Note that the packet transmission time t for the HTTP bot is about 1 second in 3G, which is larger than 0.3 seconds in WiFi, since the TCP round-trip delay is more significant in 3G than in WiFi. Similar to WiFi, the C2DM bot consumes the *least* amount of 3G power among all types of bot.

Type of Bot	t (in sec)	λ (in sec)	W (in mJ)
C2DM	0.1	1680	4892.2
IRC	0.1	90	97844.0
HTTP (5min)	1	300	63072.0
HTTP (10min)	1	600	31536.0
HTTP (30min)	1	1800	10512.0

Table 2: 3G power consumption of each bot in one hour.

5.3 Controllability

We now explore the following question: if the botmaster wants to send a command to N mobile bots, what is the minimum time T^* needed such that with a high probability p, all these N bots have received the command? We say that a botnet has good *controllability* if T^* is small for given N and p. Enabling good controllability is important for a botnet, for instance, when the botmaster wants to launch a synchronized jamming or DDoS attack. Here, we seek to show that a C2DM botnet can have good controllability.

Let $\hat{T}_k$ be the random variable of the time duration that a botmaster needs to send a notification to the k^{th} bot via the C2DM service. $\hat{T}_k$ can be expressed as:

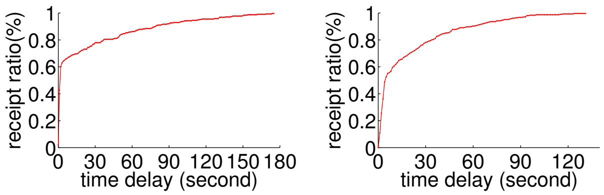
$$\hat{T}_k = (k-1)\delta + \hat{\tau}_k, \quad k = 1, \ldots, N, \tag{7}$$

where δ is the time between two consecutive C2DM requests generated by the botmaster, and $\hat{\tau}_k$ is the random variable of the time duration of sending a request from the botmaster via the C2DM service to the k^{th} bot. In our measurement, we set $\delta = 1/50$ seconds, implying that the botmaster generates 50 requests to the C2DM service per second. Note that this is significantly below our 100 QPS requirement to avoid alerting the C2DM service (see §4.3).

Note that $\hat{\tau}_k$ is related to the network conditions, phone status, C2DM server load, etc. We perform two experiments to measure the probability distribution of $\hat{\tau}_i$ in both WiFi and 3G networks, using the similar setups in §5.2.1. Figure 10(a) shows the measurement results for the WiFi experiment. We find that in 60% of time, a command is delivered within 2 seconds; and in 95% of time, a command is delivered within 2 minutes. Figure 10(b) shows the measurement results for the 3G experiment. In 53% of time, a command is delivered within 5 seconds; in 99% of time, it can be delivered in 2 minutes.

Let $\hat{T}$ be the random variable for *all* bots to receive the same command from the botmaster. We can express $\hat{T}$ as:

$$\hat{T} = \max_{i \in \{1,..,N\}} \{\hat{T}_k\} = (N-1)\delta + \max_{i \in \{1,..,N\}} \{\hat{\tau}_k\}. \tag{8}$$



(a) t in WiFi (b) t in 3G

Figure 10: Probability distribution function of time delay $\hat{\tau}$ for bots to receive a C2DM message: (a) WiFi (b) 3G.

To ease our analysis, we approximate $\hat{\tau}_k$ as an independent and identically distributed exponential random variable with parameter μ, which denotes the average command arrival rate to a bot. Let f_{τ^*} be the probability density function of $\max_{i\in\{1,..,N\}}\{\hat{\tau}_k\}$. Using order statistics [32], we have:

$$f_{\tau^*}(t) = N\left(1-e^{-\mu t}\right)^{N-1}\mu e^{-\mu t} \quad \text{for } t \geq 0. \tag{9}$$

To find T^* such that with a high probability p all N bots have received the command, we can numerically evaluate the following expression to obtain T_d:

$$\int_{t=0}^{T_d} f_{\tau^*}(t) = p. \tag{10}$$

Finally, $T^* = (N-1)\delta + T_d$. To illustrate, we consider a C2DM botnet with $N = 10000$ bots, $\delta = 1/50$ seconds, $1/\mu$ is 21.7 seconds for WiFi, and 18.9 seconds for 3G (the latter two are based on our measurements in Figures 10(a) and 10(b), respectively). Table 3 shows different values of T^* under different probability requirements p. In summary, it takes less than 8 minutes to reach all bots in a large-scale C2DM botnet. This shows the good controllability of a C2DM botnet.

p	T^* (WiFi)	T^* (3G)
0.80	432.4 sec	402.4 sec
0.90	448.7 sec	416.6 sec
0.95	464.3 sec	430.2 sec

Table 3: Controllability: value of T^* under different probability guarantees.

6. BOTNET DEPLOYMENT AND ATTACK

In this section, we present the details of deploying a C2DM botnet from an attacker's perspective. As a proof of concept, we also implement a small-scale C2DM botnet. We demonstrate how it can be injected into legitimate mobile applications and used to launch a real-life spamming attack.

6.1 Deployment

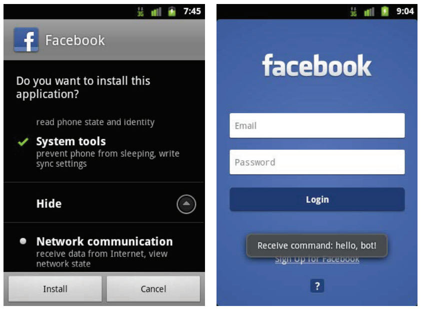
Infection. We first explain how we infect a target Android application with a C2DM bot. Note that the infection approach we describe here is also applicable for adding any types of malicious code.

A typical Android application is composed of multiple object files and packaged into a single file with .apk extension. Each object file contains executable bytecode designed for the runtime environment called the Dalvik Virtual Machine. To inject malicious code into a target Android application, one cannot directly operate on the application's source code that is generally unavailable. Instead, one can *disassemble* the bytecode of the target application into assembly-like code called *Smali* code, using the official Android Apktool software [33].

After disassembling an Android application, one can access a manifest file called AndroidManifest.xml, which describes the metadata of an Android application, including the name, permissions, activities, and services. It also specifies the *main activity*, which is the first activity that will start when the application is launched, with a tag called "android.intent.action.MAIN". An attacker can modify the tag to change the main activity of the target application to start the malicious activity (i.e., the C2DM bot program). The malicious activity should run in the background so that the infected application appears to behave normally. In addition, the attacker needs to modify AndroidManifest.xml to add extra permissions for the malicious activity. For the C2DM bot, we add the admission "com.google.android.c2dm.permission.RECEIVE".

Propagation. One way to propagate the infected target application is to upload it to the official Android Market, but the infected application may be blocked due to the checking procedure. An attacker can upload the infected application to some third-party markets such as HiAPK and AppChina. In China and some Middle East countries, the Android Market may have low download bandwidth or may be blocked [34], so users may have to rely on third-party markets to download applications. This makes the propagation of an infected application feasible.

Implementation. We implement a proof-of-concept C2DM bot prototype and inject it into a popular application called Facebook for Android, which uses many permissions including the C2DM access, full Internet access, read contact data, and GPS. This feature enables us to inject the C2DM bot into the application without requiring extra permissions. First, we note that the permission "com.google.android.c2dm.permission.RECEIVE" for the C2DM service is not as sensitive as other permissions, such as sending SMS and receiving SMS, in drawing attention of users. During the installation, this permission is represented as "receive data from Internet" (see Figure 11(a)), which is hidden in a collapsed list. Users may not notice this permission is included. Figure 11(b) shows that once the application is launched, the C2DM bot service will start in background and it can receive commands from the botmaster via the C2DM service.



Do you want to install this application?

read phone state and identity

✓ System tools
prevent phone from sleeping, write sync settings

Hide

● Network communication
receive data from Internet, view network state

Receive command: hello, bot!

(a) Installation (b) Startup

Figure 11: Injecting a C2DM bot into the Facebook app.

6.2 Attack Demonstration

We conduct a trial study on deploying a C2DM botnet in reality and launching a spamming attack known as the *SMS-spam-and-click* attack. Our goal is to demonstrate the threat of a mobile botnet in practice, using our C2DM botnet as a motivating example.

Specifically, we recruited 10 volunteers and installed C2DM bots into their Android phones. Thus, we construct a real-life, small-scale C2DM botnet with 10 bots. These bots will generate an SMS message containing a URL that refers to a website under our control for data collection. We then experiment three different forms of attacks:

- **Random spamming:** Each of the 10 bots sends 10 SMS messages to 10 different registered phone numbers selected at random. We are interested in finding how many people will click our URL link. Note that this attack is similar to the conventional spamming attack. In our experiment, none of the 100 people clicked our link, which means that the conventional spamming attack may not receive a high number of clicks.
- **Contacts spamming:** Each of the 10 bots now sends an SMS message to 10 known people who are in the phone's contact list. The result is that 23 out of 100 people clicked our URL, and this shows that exploiting friendship in an attack can enhance the click success rate.
- **Man-in-the-middle spamming:** Each of the 10 bots monitors the SMS conversation of the compromised mobile phone, and the bot sends an SMS message to the contacted person on the other end. We instruct each bot to send a maximum of 10 SMS messages. Our experiments show that 45% of contacted persons clicked the URL. This shows that the man-in-the-middle spamming can also have a high success rate.

7. EXTENSION TO OTHER PLATFORMS

We mention in §1 that many platforms other than Android provide push notification services and can be exploited as well. We now elaborate how to extend a C2DM botnet to other platforms.

The push notification services of other platforms are similar to C2DM in the architectural design, as shown in Figure 12. When an application launches in a mobile device, it needs to register to the push service to get a unique ID (it may have different names in different platforms, e.g., *device token* in iOS and *push URI* in Windows), and then sends it to the application server. When the application server wants to send a push notification to an application, it sends the ID together with the payload to a push server, which then forwards the payload to the application.

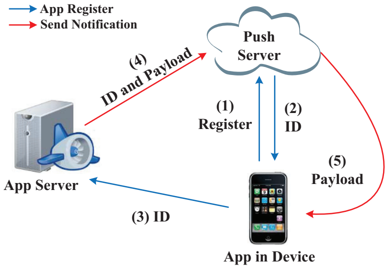


Figure 12: Architecture of push notification service.

Table 4 compares different push notification services, including Google's C2DM, Apple's Push Notification Service (APNS), Microsoft's Push Notification Service (MPNS), Blackberry's Push Service (BPS), and Nokia's Notification API (NNA). The maximum payload size of APNS is 256 bytes, which is the smallest among all, but it still suffices for a typical botnet command. In terms of request quota, for APNS and NNA, they do not set any limit on the number of push notifications that the application server can send per day. For MPNS, the quota is unlimited for an authenticated web service, and 500 per day per device for unauthenticated ones. To authenticate a web service, a developer needs to apply for a certificate from a certificate authority and upload it to Windows Phone Marketplace. For BPS, it has two versions: Essential and Plus. Their difference is that the Plus version supports delivery status report. The quotas of the Essential and Plus versions are unlimited and 100,000 for free per day. Thus, compared to the C2DM platform, botnets on other platforms in general have better scalability due to less strict quota limits.

Service	Max. Payload	Quota (per Day)
C2DM	1 KB	200,000
APNS	256 Bytes	Unlimited
MPNS	3 KB (+ 1KB Header)	Unlimited (authenticated) 500 (unauthenticated)
BPS	8 KB	Unlimited (Essential version) 100,000 free (Plus version)
NNA	1.5 KB	Unlimited

Table 4: Comparison of push notification services.

Similar to C2DM, each push request of other push notification services can be sent to one device only (see §4.1). Thus, the controllability of a botnet in other platforms is also determined by the QPS limitation, as discussed in §5.3.

Note that the C2DM service is deprecated on June 26, 2012, and is replaced by the *Google Cloud Messaging for Android (GCM)* [35]. GCM is a push notification service that is based on C2DM, with several enhancements: supporting multicast messages, no quotas, supporting JSON-based notification messages, better battery efficiency, etc. Nevertheless, the fundamental design of the C2DM botnet is to exploit the push notification service as a C&C channel, and we expect that our proposed mobile botnet design remains applicable in GCM. In fact, the GCM botnet might potentially be more effective in command dissemination as a command can be sent to multiple bots via multicast, and no quota is imposed. We plan to further investigate it in future work.

In summary, we believe that it is feasible to extend the C2DM botnet design to other push notification platforms. Furthermore, the botnets on such platforms share the same properties as the C2DM botnet, such as stealthiness, resource efficiency, and controllability.

8. RECOMMENDATIONS ON DEFENSE

In this section, we discuss possible defense strategies against a C2DM botnet, or more generally, push-styled mobile botnets deployable in today's push notification platforms. While we have yet identified this kind of botnets in the wild, our work suggests that their eventual existence is anticipated.

We point out that most existing botnet detection methods cannot effectively detect push-styled mobile botnets. Since both push-styled bots and legitimate applications connect to official push notification servers to receive messages, it is non-trivial for anomaly-based detection methods (e.g., BotSniffer [22]) and mining-based detection methods (e.g., BotMiner [23]) to separate botnet traffic from other legitimate application traffic. In the following, we suggest possible defense strategies against push-styled mobile botnets.

Based on our C2DM botnet design, a bot will send its unique ID to the botmaster directly, or via push notification servers, during its registration. Thus, we may deploy a sandbox system in a mobile device to monitor the network behavior of an application. The sandbox system can check whether an application sends data to suspicious addresses or sends suspicious requests to push notification servers. Since many legitimate applications also send data to legitimate addresses or push notification servers, this defense mechanism requires further traffic analysis to confirm the existence of a push-styled bot.

Our experience is that not every mobile device platform has strict authority management over push notifications. Users may unnecessarily enable certain applications to support push notification services. Specifically, in Android, the permission of receiving C2DM messages "com.google.android.c2dm.permission.RECEIVE" is not a *native permission*, which starts with "android.permission.". Thus, it has not been considered as sensitive as other native permissions which may incur monetary costs to users or require access to private information, such as "android.permission.SEND_SMS" and "android.permission.RECEIVE_SMS". We suggest that the permission of receiving C2DM messages should also be treated as one of the sensitive permissions. If an application requires this permission, then it should be examined to ensure that the permission is actually required for its functions. In addition, the manifest file AndroidManifest.xml of the application should be checked to see whether there is more than one C2DM receiver. In this case, the application may have been injected with some unauthorized C2DM receivers.

9. CONCLUSION

Push notification services have provided great convenience and flexibility for applications to receive light messages from application servers. In this paper, we propose the design of a novel cloud-based push-styled mobile botnet, which exploits the push notification service for lightweight command dissemination. We take Google's C2DM service for the Android platform as a motivating example to construct this type of mobile botnets. We demonstrate how a C2DM botnet can feasibly utilize Google's C2DM service as a C&C channel, and justify how its design can be made scalable and resilient against service disruption. Through in-depth evaluation, we show that the C2DM botnet is stealthy, resource-efficient, and controllable. We prototype the C2DM botnet and demonstrate how it can be deployed in reality. Finally, we provide recommendations on potential defense strategies against such push-styled mobile botnets in general. Our C2DM botnet prototype is available for download at **http://ansrlab.cse.cuhk.edu.hk/software/c2dmbotnet**.

10. ACKNOWLEDGMENTS

The work is supported in part by the National Natural Science Foundation of China under grants 60921003 and 61103241.

11. REFERENCES

[1] A. Apvrille. Symbian worm yxes: Towards mobile botnets? In *19th Annual EICAR Conference, France*, 2010.

[2] P. Porras, H. Saïdi, and V. Yegneswaran. An analysis of the ikee. b iphone botnet. *Security and Privacy in Mobile Information and Communication Systems*, pages 141–152, 2010.

[3] Lookout Inc. Security alert: Geinimi, sophisticated new android trojan found in wild, 2010. http://blog.mylookout.com/blog/2010/12/29/geinimi_trojan.

[4] Trend Micro Inc. Zeus targets mobile users. http://blog.trendmicro.com/zeus-targets-mobile-users, 2011.

[5] X. Jiang. Security Alert: AnserverBot, New Sophisticated Android Bot Found in Alternative Android Markets. http://www.csc.ncsu.edu/faculty/jiang/AnserverBot/, Sep 2011.

[6] Kaspersky Inc. Irc bot for android. http://www.securelist.com/en/blog/208193332/IRC_bot_for_Android, 2012.

[7] Google Inc. Android Cloud to Device Messaging Framework. http://code.google.com/android/c2dm.

[8] Apple Inc. Local and Push Notification Programming Guide. http://developer.apple.com/library/mac/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/RemoteNotificationsPG.pdf, 2011.

[9] Microsoft Inc. Push Notifications Overview for Windows Phone. http://msdn.microsoft.com/en-us/library/ff402558(v=vs.92).aspx.

[10] Reserach In Motion Inc. Blackberry push service. http://http://us.blackberry.com/developers/platform/pushapi.jsp.

[11] Nokia Inc. Notifications api. https://projects.developer.nokia.com/notificationsapi/wiki.

[12] P. Traynor, M. Lin, M. Ongtang, V. Rao, T. Jaeger, P. McDaniel, and T. La Porta. On Cellular Botnets: Measuring the Impact of Malicious Devices on a Cellular Network Core. In *Proc. of ACM CCS*, 2009.

[13] K. Singh, S. Sangal, N. Jain, P. Traynor, and W. Lee. Evaluating Bluetooth as a Medium for Botnet Command and Control. In *Proc. of DIMVA*, 2010.

[14] C. Xiang, F. Binxing, Y. Lihua, L. Xiaoyi, and Z. Tianning. Andbot: Towards Advanced Mobile Botnets. In *Proc. of USENIX LEET*, pages 11–11. USENIX Association, 2011.

[15] K.G. Zeng, Y.and Shin and X. Hu. Design of SMS Commanded-and-Controlled and P2P-structured Mobile Botnet. In *Proc. of ACM WiSec*, 2012.

[16] G. Geng, G. Xu, M. Zhang, Y. Yang, and G. Yang. An improved sms based heterogeneous mobile botnet model. In *Information and Automation (ICIA), 2011 IEEE International Conference on*, pages 198–202. IEEE, 2011.

[17] J. Hua and K. Sakurai. A SMS-based Mobile Botnet Using Flooding Algorithm. In *Proc. of IFIP WISTP*, 2011.

[18] G. Weidman. Transparent Botnet Command and Control for Smartphones over SMS. In *Shmoocon*, 2011.

[19] C. Mulliner and J.P. Seifert. Rise of the iBots: 0wning a Telco Network. In *Proc. of IEEE MALWARE*, pages 19–20, 2010.

[20] M. Akiyama, T. Kawamoto, M. Shimamura, T. Yokoyama, Y. Kadobayashi, and S. Yamaguchi. A Proposal of Metrics for Botnet Detection Based on its Cooperative Behavior. In *Proc. of SAINT Workshops*, pages 82–82. Ieee, 2007.

[21] H. Choi, H. Lee, H. Lee, and H. Kim. Botnet Detection by Monitoring Group Activities in DNS Traffic. In *Proc. of IEEE CIT*, 2007.

[22] G. Gu, J. Zhang, and W. Lee. BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic. In *Proc. of NDSS*, 2008.

[23] G. Gu, R. Perdisci, J. Zhang, and W. Lee. BotMiner: Clustering Analysis of Network Traffic for Protocol-and Structure-Independent Botnet Detection. In *Proc. of USENIX Security*, 2008.

[24] I. Vural and H. Venter. Mobile Botnet Detection Using Network Forensics. In *Future Internet Symposium*, 2010.

[25] E. Kartaltepe, J. Morales, S. Xu, and R. Sandhu. Social Network-Based Botnet Command-and-Control: Emerging Threats and Countermeasures. In *Proc. of ACNS*, 2010.

[26] Android Developers. http://developer.android.com.

[27] Tim Hopper. My email analytics. http://www.stiglerdiet.com/2012/04/05/my-email-analytics/.

[28] J. Oikarinen and D. Reed. Internet relay chat protocol. *RFC 1459*, 1993.

[29] P. Mutton. Pircbot 1.2. 5 java irc api: Have fun with java. *Java Developer's Journal*, 8(12):26–32, 2003.

[30] Beware ircd. http://ircd.bircd.org.

[31] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R.P. Dick, Z.M. Mao, and L. Yang. Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones. In *Proc. of ACM CODES+ISSS*, pages 105–114. ACM, 2010.

[32] Herbert A. David. *Order Statistics, 2nd Ed.* Wiley-Interscience, 1981.

[33] Apktool. http://code.google.com/p/android-apktool/.

[34] S. Ye. Android Market is Currently Blocked in China. Here are your Alternatives, Sep 2011. http://techrice.com/2011/10/09/android-market-is-currently-blocked-in-china-here-are-your-alternatives/.

[35] Google Inc. Migration. http://developer.android.com/guide/google/gcm/c2dm.html, 2012.