# Botnet Triple-Channel Model: Towards Resilient and Efficient Bidirectional Communication Botnets

Cui Xiang[1], Fang Binxing[1,2], Shi Jinqiao[3], and Liu Chaoge[1]

[1] Institute of Computing Technology, Chinese Academy of Sciences, P.R. China
[2] Beijing University of Posts and Telecommunications, P.R. China
[3] Institute of Information Engineering, Chinese Academy of Sciences, P.R. China
cuixiang@ict.ac.cn

**Abstract.** Current research on future botnets mainly focuses on how to design a resilient *downlink* command and control (C&C) channel. However, the *uplink* data channel, which is generally vulnerable, inefficient even absent, has attracted little attention. In fact, most of current botnets (even large-scale and well-known) contain either a resilient (maybe also efficient) unidirectional downlink C&C channel or a vulnerable bidirectional communication channel, making the botnets either hard to monitor or easy to be taken down. To address the above problem and equip a botnet with resilient and efficient bidirectional communication capability, in this paper, we propose a communication channel division scheme and then establish a Botnet Triple-Channel Model (BTM). In a nutshell, BTM divides a traditional communication channel into three independent sub-channels, denoting as *Command Download Channel (CDC)*, *Registration Channel (RC) and Data Upload Channel (DUC)*, respectively. To illuminate the feasibility, we implement a BTM based botnet prototype named *RoemBot*, which exploits URL Flux for CDC, Domain Flux for RC and Cloud Flux for DUC. We also evaluate the resilience and efficiency of RoemBot. In the end, we attempt to make a conclusion that resilient and efficient bidirectional communication design represents a main direction of future botnets.

**Keywords:** Botnet, C&C, BTM, URL Flux, Domain Flux, Cloud Flux.

## 1 Introduction

A botnet refers to a group of compromised computers that are remotely controlled by botmasters via C&C channels. Botnets are the main cause of many Internet attacks such as DDoS, Email spam, seeding malware, and the recent BitCoin Mining [1, 2, 28] etc. As botnet-based attacks become popular and dangerous, researchers have studied how to detect, track, measure and mitigate them. Besides, some researchers focus on possible design of future botnets in order to fight against them [3-9]. However, current research on future botnets **only** focuses on how to design a resilient and efficient *downlink* (from botmasters to bots, generally used to deliver commands and new executables) C&C channel. However, the *uplink* (from bots to botmasters,

T. Zia et al. (Eds.): SecureComm 2013, LNICST 127, pp. 53–68, 2013.

generally used to monitor botnets and collect data) data channel, which is generally vulnerable, inefficient even absent in most of current botnets, has attracted little attention. In this paper, we mainly focus on the problem and discuss the model, feasibility and methodology of designing a resilient and efficient bidirectional communication botnet which supports both a resilient downlink C&C channel and an efficient uplink data channel. This kind of advanced botnet will no doubt be very attractive for botmasters, thus we should promote the development of more efficient countermeasures in advance.

### 1.1 Weaknesses of Current Botnets

The first generation botnets have a static centralized topology. The earliest well-known botnets, such as SDbot, Rbot and Agobot, mainly use the IRC protocol. In order to be stealthier, botmasters begin to adopt HTTP protocol, such as Bobax, Rustock, Clickbot and Coreflood. Due to the static centralized topology and the hardcoded C&C servers, both IRC and HTTP based botnets surfer from the single-point-of-failure problem. That is, once the domain name and IP address are located by defenders, the whole botnet could be shut down easily. For example, the well-known Rustock and Coreflood botnets have been taken down on Mar. 2011 and Apr. 2011, respectively [19].

The second generation botnets turn to adopt a decentralized topology, such as Slapper, Nugache, Storm [22], Waledac [20], Kelihos [16], Zeus [15] and ZeroAccess [21]. It's generally admitted that the essential driving force of the botnet evolution from centralized to decentralized structure is to eliminate the single-point-of-failure problem. At first glance, P2P botnets seem to be more resilient to takedown attempts than centralized botnets, because they have no single-point-of-failure. However, previous work has shown that P2P-based botnets are not really secure [10, 11, 22]. For structured P2P botnets which employ distributed hash table (DHT), such as Storm, are vulnerable to Index Poisoning and Sybil attack [11] inevitably; for unstructured P2P botnets which use custom P2P protocols, such as Waledac, Miner [1, 2], Zeus and ZeroAccess, are vulnerable to crawling and sensor injecting inescapably [10]. For example, the well-known Waledac and Kelihos botnets have been taken down on Feb. 2010 and Sep. 2011, respectively [19]. Another significant problem is that P2P botnets have no uplink data channel, so it is difficult for a P2P botnet to monitor the botnet and collect information from bots. To build a temporal uplink data channel, temporal central servers are indispensible.

Based on the above analysis, we can see that a resilient and efficient bidirectional communication botnet is more desirable than a P2P botnet. Therefore, the third generation botnets, such as Conficker [23] and Torpig, begin to adopt a dynamic centralized topology named Domain Flux. However, Domain Flux is significantly limited by the performance of C&C servers, making uploading massive files by large-scale botnets very hard. Furthermore, if the authentication mechanism is not strong enough, the botnet will suffer from sinkhole attack. For example, the well-known Torpig [24] and Kraken [27] botnets have been sinkholed by defenders on Jan. 2009 and Apr. 2008, respectively.

To the best of our knowledge, most of current botnets (even large-scale and well-known) contains either a resilient (perhaps also efficient) unidirectional downlink C&C channel or a vulnerable bidirectional communication channel, making the botnets either hard to monitor or easy to be taken down. How to construct a resilient and efficient bidirectional communication botnet poses a great challenge to this day.

### 1.2 Intrinsic Cause Analysis

The internal cause of the above problems can partly explained by the fact that current botnets always rely on only one C&C protocol to accomplish all tasks, however, it is impossible for any existing C&C protocol to satisfy all requirements solely. For example, the relatively resilient P2P and URL Flux [5] protocols are limited by monitorability; the recoverable Domain Flux protocol is limited by robustness and efficiency. In a word, each C&C protocol has its particular advantages as well as corresponding limitations. Although Conficker employs both Domain Flux and P2P protocols, it only use its P2P components as backup channels in case the Domain Flux being ineffective. The proposed Botnet Triple-Channel Model aims at solving the problem to some degree.

### 1.3 Proposed Bidirectional Communication Botnet

Considering the above problems encountered by current botnets, the design of an advanced botnet, from our understanding, should satisfy four basic security properties denoting as *Resilience, Openness, Efficiency* and *Monitorability*, respectively. We believe that the four basic security properties are indispensible for constructing a practical advanced botnet.

**Definition 1**. *Resilience* denotes the **robustness** of a botnet when the crucial nodes of its infrastructure are attacked; and the **recoverability** of a botnet in case of being "shut down" temporally.

**Definition 2**. *Openness* is the **risk level** of a botnet faced in case the DNS/IP of C&C servers, the hard-coded symmetric/public keys and the hard-coded algorithms are exposed. If the risk level is low, we say the botnet has openness.

**Definition 3**. *Efficiency* is the **performance** of a botnet when managing large-scale botnets (Downlink Performance), accepting massive files in parallel and continuously (Uplink Performance), and storing massive files uploaded by large-scale botnets (Storage Performance).

**Definition 4**. *Monitorability* is the **capability** of a botnet to accept the initial *One-time Registration* (see definition 7) and the subsequent *Persistent Status Report* (see definition 8).

From the perspective of security properties requirement, the proposed bidirectional communication botnet should satisfy all the security properties shown in Table.1 (in Section 2).

In summary, our contributions are:

- We analyze the weaknesses of current botnets and the possible intrinsic cause, and then propose a Botnet Triple-Channel Model.
- We implement a BTM based botnet prototype, which is proved to satisfy the four basic security properties – *Resilience, Openness, Efficiency* and *Monitorability*.
- We propose an **open and efficient** Data Upload Channel design named *Cloud Flux*, which is generally absent in most of current botnets.
- We find that BTM based botnets make takedown efforts more challenging, which should be given more consideration in advance.

### 1.4 Paper Organization

The remainder of this paper is structured as follows. Section 2 gives an overview of BTM. In Section 3, we introduce the implementation of RoemBot based on BTM. Section 4 provides an analysis of the resilience and efficiency of RoemBot. In Section 5, we discuss how to defend against RoemBot. Finally, we outline related work in Section 6 and summarize our work in Section 7.

## 2 Botnet Triple-Channel Model

To construct an "Ideal Botnet" which could satisfy all the four basic security properties, we proposed a *Botnet Triple-Channel Model (BTM)*.

**Architecture.** BTM (shown in Fig.1) divides a traditional C&C channel into three independent sub-channels, denoting as *Command Download Channel (CDC), Registration Channel (RC)* and *Data Upload Channel (DUC)*, respectively. That is, BTM includes three independent but cooperative C&C sub-channels.

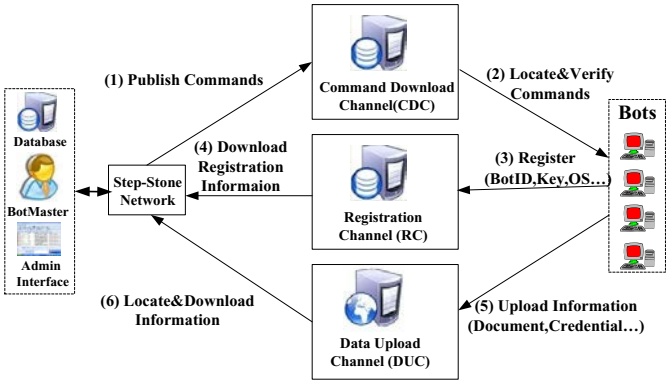


**Fig. 1.** Botnet Triple-Channel Model

**Security Properties Requirement.** Each sub-channel, determined by its functionality and characteristic, requires particular properties (summarized in Table.1) and is only responsible for particular tasks.

**Command Download Channel (CDC).** CDC is **only** responsible for commands distribution. CDC must be resilient and open to defend against coordinated countermeasures, and must have excellent downlink performance to support large-scale management. However, the uplink channel could be absent. Thus, a resilient, open and efficient unidirectional C&C protocol is suitable for CDC.

**Definition 5.** *RI* denotes Registration Information. RI=<BotID, SymmetricKey, HostInfo>, where *BotID* is used to identify a bot uniquely and is generated randomly based on host information when a bot compromises a new victim; *SymmetricKey* is used to encrypt all kinds of uploading data such as SI (see definition 6) and stolen files. Since the hardcoded key can in all cases be found through reverse engineering, each bot should generate an individualized and different symmetric key. In this way, investigating one or more bots will not impact the confidentiality of the whole botnet; *HostInfo* includes basic information describing a victim such as internal IP address, operation system and version, CPU/Memory, installed Antivirus software, and system language etc. Note that BotID and SymmetricKey must keep unchanged in the whole lifespan of a bot, and RI must be encrypted by the **hardcoded public key** of bots.

**Definition 6.** *SI* denotes Status Information. General SI includes *command received, command execution finished, download finished, upload finished, victim environment changed* etc. Note that SI must be encrypted by the individualized *SymmetricKey* (see Definition 5) to ensure confidentiality; thus detecting and then investigating one or more bots will not impact other bots.

**Definition 7.** *One-time Registration (a.k.a. Call-Home)* means a bot must report its individualized RI after initial execution. One-time Registration makes a botmaster could monitor the membership, population size, and geographical distribution of a botnet.

**Definition 8.** *Persistent Status Report* means a bot should report its SI persistently or on-demand according to the received commands. Persistent Status Report makes a botmaster could monitor the active size and activities of botnets in time.

**Registration Channel (RC).** RC is **only** responsible for RI and SI collection. RC must be recoverable and open to defend against the physical control and sinkhole attack, and must have uplink channel to accept the incoming RI and SI during one-time registration and persistent status report, respectively. Since the registration servers must be lightweight and easy to deploy, the robustness and efficiency is not necessary. Since the RI and SI could be downloaded and removed by botmasters in time, the excellent storage performance is not indispensable. Thus, a recoverable, open and monitoring bidirectional communication protocol is suitable for RC.

**Table 1.** The Security Properties Requirement of the Divided Sub-Channels

RO=Robustness, RE=Recoverability, D/I=DNS/IP, K=Key, AL= Algorithm, DP=Downlink Performance, UP=Uplink Performance, SP=Storage Performance, OR= One-time Registration, PSR= Persistent Status Report

| Sub-channel\ Property | Resilience | | Openness | | | Efficiency | | | Monitorability | |
|---|---|---|---|---|---|---|---|---|---|---|
| | RO | RE | D/I | K | AL | DP | UP | SP | OR | PSR |
| CDC | √ | √ | √ | √ | √ | √ | | | | |
| RC | | √ | √ | √ | √ | | | | √ | √ |
| DUC | | | √ | √ | √ | √ | √ | √ | | |

**Data Upload Channel (DUC).** DUC is **only** responsible for transferring stolen data to botmasters. DUC must have excellent uplink performance to enable massive data uploading in parallel by large-scale botnets, have excellent downlink performance for botmasters to download the massive files, have huge storage performance to store the uploaded data for some time. However, DUC itself is not necessary to be very resilient because the DUC related resources (i.e., the address of the given cloud services) could be dynamically delivered to bots via CDC, hence providing a recoverable capability indirectly; DUC need not Monitorability, because the uploading status could be sent to botmasters using SI via RC. Another important thing we have to considerate is that DUC must ensure the uploaded data can and can only be located and decrypted by botmasters who own the RI of each bot. Thus, an open and efficient bidirectional communication protocol is suitable for DUC.

## 3 RoemBot: A BTM-Based Botnet

To explain the proposed BTM in more detail, we implement a prototype named ***RoemBot*** (a ***R***esilient, ***O***pen, ***E***fficient and ***M***onitoring ***b***ot). We analyze and evaluate the resilience and efficiency of RoemBot emphatically in section 4.

### 3.1 Overview of RoemBot

The architecture of RoemBot is shown in Fig.2. RoemBot exploits URL Flux [5] for CDC, Domain Flux for RC and a new protocol (named *Cloud Flux* for convenience) for DUC. The C&C procedures of RoemBot are explained as below.

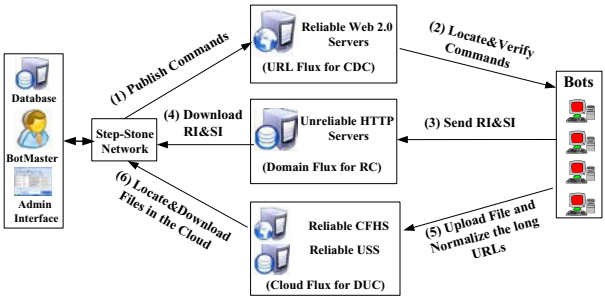


**Fig. 2.** Architecture and Implementation of RoemBot

**Phase 1:** A botmaster encrypts and signs the commands, and then publishes them to reliable Web 2.0 servers (i.e., Twitter).

**Phase 2:** The bots try to locate the authentic commands using URL Flux protocol.

**Phase 3:** The bots begin to locate the authentic registration servers using Domain Generation Algorithm (DGA) [14], depending on the *Seed* value such as current date/time and Twitter trends obtained from commands. Note that the *Seed* value must be distributed via commands to defend against sinkhole attack (see Section 4.3).

**Phase 4:** The botmaster downloads the encrypted RI and SI, and then decrypt them using the corresponding private key and *SymmetricKey*, respectively.

**Phase 5:** Based on the URL of *Cloud-based File Hosting Services* (*CFHS*) obtained from commands, the bots begin to upload the collected data to CFHS. And then normalizes the long URL to shorten URL which could be predicted by the botmaster who owns the RI of each bot.

**Phase 6:** The botmaster locates each file uploaded by each bot and then downloads the files one by one. Note that the files can and can only be identified and decrypted by the botmaster who owns the RI of each bot.

### 3.2 URL Flux Protocol for CDC

**Protocol Selection.** According to the requirement of CDC (Table.1) and the security properties of each C&C protocol (Table.2), we can see that URL Flux is suitable for CDC. The architecture of URL Flux is described in Fig.3. More detail about URL Flux is introduced in [5].

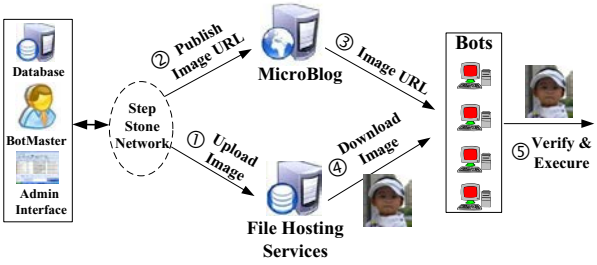


**Fig. 3.** URL Flux based CDC of RoemBot

### 3.3 Domain Flux Protocol for RC

This registration procedure is very crucial for botmasters to monitor the botnet and locate the uploaded stolen files.

**Protocol Selection.** According to the requirement of RC (Table.1) and the security properties of each C&C protocol (Table.2), we can see that Domain Flux is suitable for RC. The registration procedure is described in Fig.4.

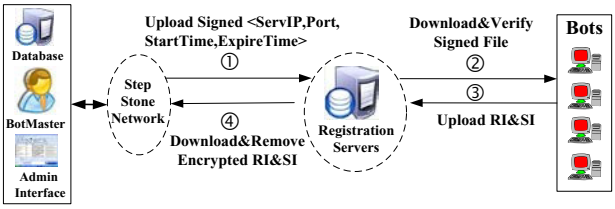


**Fig. 4.** Registration Procedure of RoemBot

**Phase 1:** Botmasters upload a certification to the registration server. The certification must include but not limited to Server IP Address, Server Port, Start Time and Expire Time. In this way, it is impossible for defenders to forge registration servers. After this, botmasters publish the randomly generated DGA *seed,* making bots could locate the registration servers using Domain Flux protocol.

**Phase 2:** Bots retrieve commands via CDC, subtract the seed and then calculate the domain names of registration servers using the hard-coded DGA which is shared with botmaster.

**Phase 3:** Bots upload RI and SI to the authentic registration servers, encrypted by the hard-coded public key and the generated *SymmetricKey*, respectively.

**Phase 4:** Botmasters download and decrypt the RI and SI, and then remove them, eliminating the risk of computer forensics or other kinds of data leakage.

### 3.4 Cloud Flux for DUC

**Motivation.** Although it seems a simple task to construct a DUC which could satisfy all of the requirements of DUC listed in Table.1, it is not the case. In fact, even well-known botnets such as Conficker, Mariposa, Torpig, Coreflood, Waledac, and Kelihos botnets are all ineffective in the aspect of retrieving the collected data from bots. Let us take Torpig as an example, which is mainly designed to harvest sensitive information from its victims. Stone-Gross took control of the Torpig botnet and observed more than 180 thousand infections and recorded almost 70 GB of data that the bots collected [24]. How to construct an **open** DUC with good downlink performance, uplink performance, and storage performance poses a great challenge to this day.

**Cloud Flux Designing.** To address the above difficulties, we propose a new protocol named *Cloud Flux* for convenience. We attempt to employ *Cloud-based File Hosting Services (CFHS)* and *URL Shortening Services (USS)* to build a qualified DUC. More specifically, CFHS provide an efficient way to upload and store files anonymously, which could also be exploited by bots. However, the cloud servers usually return a random URL pointing to the uploaded file. It happens that USS could solve the problem by mapping a given URL to a customized shorten URL. In a word, we could combine the two services together to establish an open and effective DUC. To describe the idea in detail, we outline the complete working procedure in Fig.5.

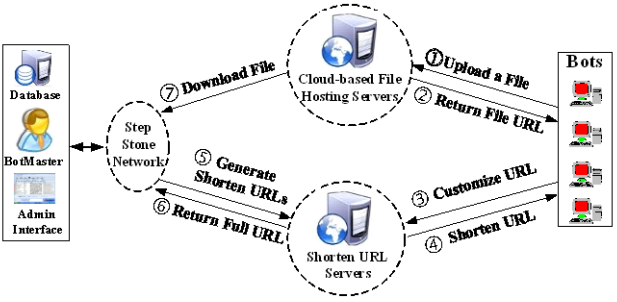


**Fig. 5.** Cloud Flux based DUC of RoemBot

**Phase 1:** A bot collects interesting contents such as credentials and sensitive files on the victim, encrypts (RC4) them using its *SymmetricKey* and stores the ciphertext into a file. After that, the bot uploads the file to a randomly selected CFHS which is obtained from the received commands. This phase can be descripted more formally as below:
*Bot_Upload = Bot.Encrypt (File, Key)* → *CFHS*

**Phase 2:** The cloud server returns a random URL representing the downloading URL of the uploaded file (i.e., http://www.sendspace.com/file/rz3ivc) to the bot.
*CFHS_Response = CFHS.Response (Full URL)* → *bots*

**Phase 3:** The bot visits a randomly selected shorten URL server which is obtained from the received commands, submits the above full URL and a desired customized shorten URL based on its *BotID* (already generated in the procedure of registration and reported to botmasters via RC) and current date. For example, if the BotID is abcd1234, the current date is 20130508, then the desired shorten URL is "http://tinyurl.com/abcd123420130508".
*Bot_Request = Bot.Request (Full URL, Desired Shorten URL)→ USS*

**Phase 4:** If successful, the desired shorten URL will be returned; otherwise, if the desired shorten URL is occupied (a low probability event), the bot has to queue the file to the next day.
*USS_Response = USS.Response (Desired Customized Shorten URL, RetCode)→ bots*

**Phase 5:** The botmaster owns all BotID thanks to the registration procedure, so he can enumerate each BotID one by one (we prefer to wait for an "*upload finished"* SI report for efficiency consideration, otherwise, enumerating the whole botnet population is very inefficient) and then generates the possible destination URL by combing the BotID with current date.
*Botmaster_Request = Botmaster. Request ( http:// USS Domain / BotID#CurrentDate)→ USS, where '#' denotes conjunction of two strings.*

**Phase 6:** If the shorten URL does exist, the corresponding full URL will be returned.
*USS_Response = USS.Response ( Full URL) → Botmaster*

**Phase 7:** The botmaster downloads the destination files automatically (using an automated crawler program) based on the returned full URL.
*Botmaster_Download = Botmaster.Download (Full URL) → LocalStorage*

Here, Cloud Flux, which starts from *Bot_Upload* and ends with *Botmaster_Download*, finishes its complete work.

**Cloud Flux Experiment.** We have evaluated the novel methodology using SendSpace [12] and TinyURL [13], the results show that it can work completely automatically in a quite efficient way.

## 4 RoemBot Resilience and Efficiency Study

### 4.1 Security Properties of Current C&C Protocols

Although the C&C protocols of botnets have evolved from centralized to decentralized topology and from static to dynamic addressing, to the best of our knowledge, there is no publicly reported botnets that could satisfy all of the four basic security properties. The summary of current C&C protocols as well as the proposed Cloud Flux is shown in Tab.2. In comparison, we also exhibit the security properties of BTM.

For an IRC botnet, it has a group of IRC servers which could link together in a P2P topology, so its CDC is robust. However, in case the DNS/IP addresses of IRC servers, the hard-coded login password in bots are exposed, the botnet will suffer

from a single-point-of-failure or hijacking. The bot can only push some limited text messages to botmasters, so the uplink performance is low.

For a HTTP protocol, it has only limited HTTP servers, more badly, all kinds of resources, such as Domain Name, publicly accessible IP address, and the physical computers, must be considered by botmasters. In case the DNS/IP addresses of HTTP servers are exposed, the botnet will also suffer from a single-point-of-failure. Although the efficiency can be enhanced by increasing the number and performance of HTTP servers, it is generally very limited and cost sensitive.

For structured P2P protocol, it is vulnerable to Index Poisoning and Sybil attack inevitably; for unstructured P2P protocol, it is vulnerable to crawling and sensor injecting inescapably. In case the hardcoded keys are exposed, the commands broadcasted among the P2P botnet could be monitored in time by defenders who employ Sybil nodes.

IP Flux (a.k.a. Fast Flux) protocol evolves from HTTP protocol. When the Domain Name of its mothership [17] is exposed, it still has a single-point-of-failure risk. Although there are multi step-stones, the efficiency of mothership is not enhanced at all. The main objective of IP Flux is to conceal the real IP address of motherships.

Domain Flux protocol evolves from HTTP protocol, it introduces a DGA to make the Domain Name of C&C servers predictable so as to equip with recoverability [23]. Although the DGA could be easily reverse analyzed, the botmasters will not lose control due to certification mechanism.

URL Flux protocol also involves from HTTP protocol, it introduces a UGA (Username Generation Algorithm) to make the URL of C&C servers predictable so as to equip with recoverability [5]. Same to DGA, UGA is also resilient to reverse engineering. The downside of URL Flux lies in its absence of uplink capability.

**Table 2.** The Security Properties of Common C&C Protocol

RO=Robustness, RE=Recoverability, D/I=DNS/IP, K=Key, AL= Algorithm, DP=Downlink Performance, UP=Uplink Performance, SP=Storage Performance, OR= One-time Registration, PSR= Persistent Status Report, H=High, L=Low, S=Support, O=On-demand, Y=Yes, N=No

| **Protocol\ Property** | **Resilience** | | **Openness** | | | **Efficiency** | | | **Monitorability** | |
|---|---|---|---|---|---|---|---|---|---|---|
| | RO | RE | D/I | K | AL | DP | UP | SP | OR | PSR |
| IRC | H | | N | N | | H | L | L | S | O |
| HTTP | L | | N | Y | | L | L | L | S | S |
| IP Flux | L | | N | | | L | L | L | S | S |
| Domain Flux | L | H | Y | Y | Y | L | L | L | S | O |
| URL Flux | H | H | Y | Y | Y | H | | | | |
| P2P | L-H | L-H | | N | | L | | | | |
| Cloud Flux | | | Y | Y | Y | H | H | H | | |
| BTM | H | H | Y | Y | Y | H | H | H | S | O |

## 4.2 URL Flux Resilience and Efficiency Study

**URL Flux Attack Model.** Security defenders such as CERT, ISP and the most important Web 2.0 providers, could reverse analyze the UGA and monitor the

behavior of particular usernames which could be generated by UGA. In addition, defenders may try to *replay* the commands.

**Resilience and Efficiency Analysis.** For username monitoring attack, URL Flux exploits a large number of public Web 2.0 services as downlink C&C servers; thus, its robustness depends on the Web 2.0 services. Only when all the hard-coded Web 2.0 services become unavailable, URL-Flux fails. Obviously, the extreme situation is an extremely low-probability event. In case the usernames generated by UGA on one Web 2.0 service is blocked by the service provider, botmasters could switch to other Web 2.0 services. Therefore, RoemBot is very resilient. The C&C servers are high-performance websites which could serve millions of communications concurrently. Therefore, RoemBot is very efficient. The published commands always include "*StartDate*" and "*ExpireDate*" [5], making replay attack impossible. Furthermore, because the private key is owned only by botmasters, injecting malicious commands is impossible.

### 4.3 Domain Flux Resilience and Efficiency Study

**Domain Flux Attack Model.** Security defenders could identify the active authentic registration servers in time using the same DGA with bots. After that, they could either setup a **sinkhole** to measure the botnet or **physically control** the active registration servers.

**Resilience and Efficiency Analysis.** Since the *Seed* used by DGA is distributed via commands dynamically, so the defenders could not predict the future domain names used by bots until they monitor the issued commands, so they can't register the domain names in advance, making sinkhole attack difficult. For botmasters, they should always setup the servers in advance, and then publish the *Seed*. In this way, bots will always firstly locate the authentic servers rather than the fake sinkhole servers. Since bots encrypt the RI using the hard-coded public key, even if the registration servers are completely controlled by defenders, the RI is also secure. Anyhow, the RI and SI will never be accessed by unauthorized people.

### 4.4 Cloud Flux Resilience and Efficiency Study

**Cloud Flux Attack Model.** The desired shortened URL (i.e., BotID+Date) makes it easy for the USS providers to enumerate potential bots by searching (and disabling) short URLs that have such a date suffix. Also, once a BotID is discovered, those URLs can be banned going forward.

**Resilience and Efficiency Analysis.** CFHS and USS could ensure sufficient performance even for large-scale botnets (i.e., more than ten millions) to store numerous files and request shortening services in parallel. So DUC is very efficient. Since the files uploaded by bots are encrypted and have random filenames, CFHS providers are hard to find them out. Since the combination of BotID and current date is not unusual, there is a relatively high collision probability with normal URLs, it is

impossible for USS providers to block all malicious requests. Furthermore, the Cloud Flux technique could also use some simple enhancements. For example, it is more useful to use keyed hashes. Hence, a better shortened URL could be HMAC (BotID+Date, SymmetricKey), where SymmetricKey is reported in RI via RC. This would defeat such enumeration efforts.

## 5 Defense against RoemBot

We introduce possible defense strategies in three ways. First, a coordinated cooperation channel should be set up to identify and defend against this technology; second, we should infiltrate botnets to monitor their activities in time; third, we should pay more attention to the relatively vulnerable step-stones used by botmasters.

**Building International Coordinated Mechanism:** RoemBot relies on Web 2.0, CFHS and USS heavily. For this reason, defenders should focus their defense effort on security enhancement for publicly available services. This effort can prevent these services from being abused to some degree. In the case that abnormalities are detected, there should be a coordinated channel such as CERT and ISP to stop the corresponding services.

**Infiltration:** Since all bots must find commands in an active way, all of them are inescapably vulnerable to an infiltrator [18, 25]. After reverse engineering of RoemBot, an infiltrator can be written using the same protocol and algorithm to simulate RoemBot. In this way, defenders are able to track the botnet activities in time.

**Step-stone Penetration and Forensics:** Botmasters always use step-stones to conceal their origination; however, step-stones are generally vulnerable and relatively easy to penetrate. Once compromising one or more step-stones, defenders could monitor the incoming traffic, making tracing back to the active botmasters possible. In addition, defenders could also infer the characteristic of botmasters based on their habits such as the keyboard layout, language preference and time-zone [26].

Although the above defense mechanisms cannot shut down or decrease the C&C capability significantly, they still could increase the cost of botmasters to some degree.

## 6 Related Works

Wang et al. [3] presented the design of an advanced hybrid peer-to-peer botnet. Vogt et al. [4] presented a “super-botnet” - that works by inter-connecting many small botnets together in a peer-to-peer fashion. Ralf Hund et al. [6] introduced the design of an advanced bot called Rambot, developed from the weaknesses they found when tracking a diverse set of botnets. Starnberger et al. [9] presented Overbot, which uses an existing P2P protocol, Kademlia, to provide a stealth C&C channel. Singh et al. [8] evaluated the feasibility of exploiting email communication for botnet C&C. Cui et al. [5] proposed URL-Flux for botnets C&C which has proved to be robust and efficient. Kui et al. [29] conducted a systematic study on the feasibility of solely using DNS queries for massive-

scale stealthy communications among entities on the Internet. Their work shows that DNS can be used as an effective stealthy C&C channel for botnets.

Nevertheless, none of existing research works has studied how botmasters might design a **resilient and efficient bidirectional** communication channel. Specially, all of the above proposed P2P and URL-Flux based botnets are unidirectional although they are resilient. Although the DNS-based C&C channel is bidirectional, its authoritative domain name servers suffer from single-point-of-failure problem, making massive-scale uploading stolen data in parallel very hard; furthermore, the botmasters must create and register the new domain names continuously. Thus, our study compliments the existing research works to some degree.

## 7 Conclusion and Future Works

In this paper, we present a Botnet Triple-Channel Model and implement a corresponding prototype named RoemBot. RoemBot exploits URL Flux for CDC, Domain Flux for RC and a new proposed protocol named *Cloud Flux* for DUC. Compared with traditional botnets, RoemBot has a more resilient commands distribution channel, a recoverable information registration channel, and a more efficient data uploading channel, which could satisfy all of the four security properties of botnets, thus promising to be very attractive for botmasters. We believe our findings demonstrate that research on alternative advanced botnets mitigation methods is urgently needed.

We also believe that BTM-based botnet design represents a main direction of future botnets. Therefore, we plan to prove that any botnet must accomplish a BTM-style architecture in order to construct an “ideal” botnet. The ultimate goal of our work is to increase the understanding of advanced botnets; we will invest more research on how to fight against this kind of advanced botnet in the future.

**Acknowledgment.** The authors would like to thank the anonymous reviewers for their helpful comments for improving this paper. This work is supported by the National Natural Science Foundation of China under grant (No. 61202409) and the National High Technology Research and Development Program (863 Program) of China under grant (No. 2012AA012902 and 2011AA01A103).

## References

1. Plohmann, D., Gerhards-Padilla, E.: Case Study of the Miner Botnet. In: Proceedings of the 4th International Conference on Cyber Conflict (2012)
2. Werner, T.: The Miner Botnet: Bitcoin Mining Goes Peer-To-Peer, Blog article by Kaspersky Lab (2011), http://www.securelist.com/en/blog/208193084/
3. Wang, P., Sparks, S., Zou, C.C.: An advanced hybrid peer to peer botnet. In: Proceedings of the First Workshop on Hot Topics in Understanding Botnets, HotBots 2007 (2007)
4. Vogt, R., Aycock, J., Jacobson, M.: Army of botnets. In: Proceedings of 14th Annual Network and Distributed System Security Symposium, NDSS 2007 (2007)

5. Cui, X., Fang, B.X., Yin, L.H., Liu, X.Y.: Andbot: Towards Advanced Mobile Botnets. In: Proceedings of the 4th Usenix Workshop on Large-scale Exploits and Emergent Threats, LEET 2011 (2011)
6. Hund, R., Hamann, M., Holz, T.: Towards Next-Generation Botnets. In: Proceedings of the 2008 European Conference on Computer Network Defense (2008)
7. Yan, G., Chen, S., Eidenbenz, S.: RatBot: Anti-enumeration Peer-to-Peer Botnets. In: Lai, X., Zhou, J., Li, H. (eds.) ISC 2011. LNCS, vol. 7001, pp. 135–151. Springer, Heidelberg (2011)
8. Kapil, S., Abhinav, S., et al.: Evaluating Email's Feasibility for Botnet Command and Control. In: Proc. of the 38th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, pp. 376–385. IEEE Computer Society, Washington, DC (2008)
9. Starnberger, G., Kruegel, C., Kirda, E.: Overbot: A Botnet Protocol Based on Kademlia. In: Proceedings of the 4th International Conference on Security and Privacy in Communication Networks (2008)
10. Rossow, C., Andriesse, D., Werner, T., Stone-Gross, B., Plohmann, D., Dietrich, C.J., Bos, H.: P2PWNED: Modeling and Evaluating the Resilience of Peer-to-Peer Botnets. In: 34th IEEE Symposium on Security and Privacy, S&P 2013, San Francisco, CA (2013)
11. Wang, P., Wu, L., Aslam, B., Zou, C.C.: A Systematic Study on Peer-to-Peer Botnets. In: Proc. of International Conference on Computer Communications and Networks (ICCCN), pp. 1–8. IEEE Computer Society, Washington, DC (2009)
12. Roland, D.P.: Malware Uses Sendspace to Store Stolen Documents (2012), doi: `http://tinyurl.com/use-Cloud-but-no-ShortenURL`
13. Neumann, A., Barnickel, J., Meyer, U.: Security and privacy implications of url shortening services. In: Proceedings of the Workshop on Web 2.0 Security and Privacy (2010)
14. Antonakakis, M., Perdisci, R., Nadji, Y., Vasiloglou, N., Abu-Nimeh, S., Lee, W., Dagon, D.: From Throw-Away Traffic to Bots: Detecting the Rise of DGA-Based Malware. In: Proceedings of the 21st USENIX Security Symposium (2012)
15. Bukowski, T.: ZeuS v3 P2P Network Monitoring, Technical Report by CERT.pl (2012)
16. Bureau, P.-M.: Same Botnet, Same Guys, New Code: Win32/Kelihos. In: VirusBulletin (2011)
17. Holz, T., Gorecki, C., Rieck, C., Freiling, F.C.: Detection and mitigation of fast-flux service networks. In: Proc. of the 15th Annual Network and Distributed System Security Symposium. USENIX Association, Berkeley (2008)
18. Cho, C.Y., Caballero, J., Grier, C., Paxson, V., Song, D.: Insights from the Inside: A View of Botnet Management from Infiltration. In: Proc. of the 3th USENIX Conference on Large-Scale Exploits and Emergent Threats: Botnets, Spyware, Worms and More, p. 2. USENIX Association, Berkeley (2010)
19. Dittrich, D.: So You Want to Take Over a Botnet. In: Proceedings of the 5th USENIX Conference on Large-Scale Exploits and Emergent Threats (2012)
20. Stock, B., Engelberth, M., Freiling, F.C., Holz, T.: Walowdac Analysis of a Peer-to-Peer Botnet. In: Proc. of the 2009 European Conference on Computer Network Defense, pp. 13–20. IEEE Computer Society, Washington, DC (2009)
21. McNamee, K.: Malware Analysis Report: ZeroAccess/Sirefef, Technical Report by Kindsight Security Labs (2012)
22. Holz, T., Steiner, M., Dahl, F., Biersack, E., Freiling, F.: Measurements and Mitigation of Peer-to-Peer-based Botnets: A Case Study on Storm Worm. In: Proceedings of the 1st USENIX Workshop on Large-Scale Exploits and Emergent Threats (2008)
23. Porras, P., Saidi, H., Yegneswaran, V.: A Foray into Conficker's Logic and Rendezvous Points. In: USENIX Workshop on Large-Scale Exploits and Emergent Threats (2009)

24. Stone-Gross, B., Cova, M., Cavallaro, L., Gilbert, B., Szydlowski, M., Kemmerer, R., Kruegel, C., Vigna, G.: Your Botnet is My Botnet: Analysis of a Botnet Takeover. In: Proc. of the 16th ACM Conference on Computer and Communications Security, pp. 635–647. ACM, New York (2009)
25. Jonell, B., Joey, C., Ryan, F.: Infiltrating waledac botnet's convert operation[EB]. Trend Micro (2009), http://us.trendmicro.com/imperia/md/content/us/pdf/threats/securitylibrary/infiltrating_the_waledac_botnet_v2.pdf (June 10, 2011)
26. APT1: Exposing One of China's Cyber Espionage Units, http://intelreport.mandiant.com/Mandiant_APT1_Report.pdf
27. Amini, P., Pierce, C.: Kraken Botnet Infiltration [EB]. Blog on DVLabs (2008), http://dvlabs.tippingpoint.com (June 10, 2011)
28. Barrett, B.: http://gizmodo.com/gaming-network-employee-turns-14-000-users-into-bitcoin-487054354
29. Xu, K., Butler, P., Saha, S., Yao, D.: DNS for Massive-scale Command and Control. IEEE Transactions of Dependable and Secure Computing (TDSC) 10(3), 143–153 (2013)