Abstract

The most important issue for a Bot is to conceal its commander source so that the analyzer could not find the commander if the Bot is under analysis. In this paper we propose a mechanism for managing the Botnet C&C communication based on Cloud Computing infrastructures. The proposed mechanism provides a hierarchical organization of managerial nodes while all of these nodes are deployed on current Cloud providers. The main goal of this paper is to introduce the new threats that can be utilized by malware developers as a persistence mechanism. To evaluate our work, we have, semi-formally, analyzed four features (Stealthy, Effectiveness, Efficiency, and Robustness) of the mechanism and show that in most of the times the C&C can remain anonymous. Since, knowing the future threats is the best way to prepare for, and then the paper represents the possible countermeasures against the proposed mechanism.

Keywords: BotCloud, Botnets, Botnet Persistence Mechanism, Cloud-based Botnet, Cloud-based C&C

1. Introduction

Although cloud computing helps enterprises to lower their infrastructure and managerial costs or to be applied as E-learning infrastructure\(^1\), yet it can be a suitable platform to deploy Botnets. BotCloud is a new term that has been coined by putting cloud computing and Botnets together: an attacker can create some cloud applications and use them for the sake of a Distributed Denial of Service (DDoS)\(^1\). This case is the simplest malicious usage of cloud platforms in the world of Botnets.

One of the main issues in Botnet design is to hide the main C&C (Command and Control) and to recover the network as simple as possible. There are many types of Botnet architectures, such as P2P and centralized\(^4\). To be prepared for recent and future attacks through Botnets, we must be familiar with these attacks (worth mentioning that Botnet attacks are currently one of the most important security issues of these days\(^5\)).

As an instance of future possible threats originated from Botnets, in this paper we propose a design for the Botnet C&C, called CBC2 (Cloud-based Botnet Command and Control) that combines cloud computing and onion routing to build a robust command and control center. In this paper we eliminate some technical issues on zombie sides, such as, Bot propagation, zombie infection and we only concentrate on Botnet C&C architecture and the communications.

There is much work done in the area of designing Botnets; various types of architectures have been proposed, such as Centralized, P2P, and HTTP based Centralized, and there are many examples for each, as follows:

- **Centralized**: EggDrop\(^6\), GTBot\(^7,8\), SDBot\(^9,10\), AgoBot\(^11,12\), SpyBot\(^13,14\), Storm\(^15,17\), and Lethic\(^18\).
- **P2P**: Sinit\(^19\), Nugache\(^16,20\), Conficker\(^21,22\), and TDL-4\(^23\).
- **HTTP**: Rustock\(^24,26\) and Zeus\(^27,29\).

Botnet designers have utilized many techniques and mechanisms, such as cloud computing and social networks, for recent Botnets; cloud computing is one of the newest platforms providing an appropriate place for Botnets. Clark\(^3\) has used cloud applications as his Bots and started a DDoS attack on behalf of those Bots to a particular web site.

As another example, SocialBots\(^30\) should be mentioned which are deployed on social network platforms. Social accounts in social networks can be seen as cloud accounts somehow; attackers use them to harvest different...
information from other users and to run different malicious operations.

One of major issues in Botnet design is to have a robust and easily-recoverable C&C. All of Botnet designers try to regard this issue; hence we must be aware of and be prepared for more flexible and recoverable Botnet architectures in order to plan more effective, defensive mechanisms against them.

In this paper we propose a Botnet design that alerts us about how cloud computing can be abused so as to commit malicious intentions. The proposed Botnet uses the HTTP based Central architecture model to design a new Botnet C&C, called CBC2. We have deployed CBC2 on a cloud computing platform and empowered its evasion ability by utilizing onion routing mechanism for message passing; it provides a complex and hardly-readable communication graph to keep our main C&C center from defenders.

2. CBC2

In this section, first we start with an overview of CBC2 and its threat model, and then we follow the section by discussing about CBC2 design details.

2.1 Overview

Figure 1. shows an overall view of CBC2 design. The entire network is divided in two parts: cloud channel (in the rest we refer to it as cloud, for abbreviation) and user-side parts (the gray circles). The cloud is an interface between the Botnet owner and Bots. All Bots pick commands from the cloud, and the Botnet owner sends out all messages and commands to the cloud. The main duty of the cloud is to pass messages from the Botnet owner to Bots and to hide the owner by providing a situation that makes hard finding it for Botnet detectors.

All messages transferring inside the cloud are encapsulated as cells which are encrypted to provide confidentiality assurance. Cells will be passed through an onion routing network to reach user-side Bots. This combination of onion routing (including cryptography) and cloud computing as infrastructure helps CBC2 to reach its targeted responsibilities.

2.2 Threat Model

We assume a global passive attack that relies on a large scale cloud based C&C with numerous nodes on the server-side and hundreds of thousands Bots. Also, we assume that different types of Botnet attacks (including Distributed Denial of Service, spams, and spying) can be covered by CBC2. Although it is not far from reality that network providers and cloud providers cooperate to prevent Botnet attacks, yet CBC2 is designed in such a way that can heal itself simply.

2.3 Construction

In this section we describe the architecture model of CBC2. Figure 2. shows the components and their communications and relationships. CBC2 consists of Bots, Masterbots, Superbots, C&C Proxies, StartPoints, and the main C&C. Bots are user-side applications that receive commands and carry them out. Masterbots are the connection points of Bots and the cloud in such a way that each Bot is associated to one specific Masterbot.

Superbots are the points that receive messages from upper levels and deliver them to Masterbots on their demands; each Superbot is in charge of some specific Masterbots. C&C proxies are main C&C surrogates and form the structure of nodes in the cloud. StartPoints are the points to where Bots connect for their first execution.
The main C&C is the interface between the Botnet owner and the cloud (and thus the Bots).

Nodes in CBC2 have a hybrid structure. There is a hierarchical relationship among C&C proxies and their subordinate Superbots, and the message flow is top-down among them. All Masterbots subordinate to a particular Superbot form a cluster. The Masterbots in the same cluster send their messages to the associated Superbot along an onion routing based network which they construct.

All of commands from the main C&C will be packed into a cell, signed by its K (private key), and finally sent to C&C proxies. Every C&C proxy decrypts the received cell and after making sure about the source of the cell, signs the command by its own K and sends it as a cell to its subordinate Superbots. All commands has an incremental unique ID (identifier) that plays the role of timestamp. All of Superbots stack the incoming commands (for a limited number of commands); this stack is ordered by commands ID; this keeps the commands in their right order even if they do not reach their destinations on time.

When a Masterbot asks for the last commands from the associated Superbot, it sends the last timestamp that it knows (according to the last received command), and the Superbot gives all the existing commands with timestamps greater than this timestamp. This process happens for Bots in the same way; that is, Bots send their last known timestamp to the associated Masterbot, and this Masterbot sends back all the commands with timestamps greater than that timestamp. Having this mechanism in place, no command is loosen when a Bot is “OFF” for some time, due to diurnal activities, etc.

In the following subsections, we elaborate CBC2 concentrating on each component type.

### 2.3.1 Bots

Bots are client side applications that receive commands, perform them, and prepare required reports as needed. Each Bot is connected to the cloud through one of the Masterbots. The communication between Bots and Masterbots takes place through HTTP because most of nodes in the cloud are web applications, and they can be accessed and communicated through the HTTP protocol. One of the main issues in designing these Bots is to have limited exposure, and not to compromise the main C&C. Limiting message passing only to Masterbots help to limit the exposure of the entire cloud. Also, having limited messages lowers the possibility of suspicious behavior from Bots side.

Every Bot starts up for the first time (bootstrap process) by introducing itself to the cloud. Each Bot is hard-coded with a unique ID and a K along with a list of StartPoint addresses. A Bot introduces its ID to a StartPoint node through a signed cell by its K. The StartPoint informs a C&C proxy about the new Bot, and this C&C proxy finds/creates a vacant online Masterbot and introduces it to the new bot. Figure 3 shows the flowchart of Bot operations.

After the first execution, the Bot only receives new commands from its associated Masterbot; once the Masterbot does not reply, the Bot sends a REINTRODUCE message to the StartPoint, and a new Masterbot is introduced to the Bot (in the case that the associated Masterbot is Off; otherwise, the Bot will be distinguished as harmful and will be dropped from the cloud). Bots know the StartPoint from the last information received from their associated Masterbot; it means Masterbots are aware of StartPoints as they change their addresses, and this change frequency must not be as high as it puts Masterbots and their Bots into starvation situation. It is worth mentioning that every Bot encrypts its data including received messages from its Masterbot.

### 2.3.2 Masterbot

Masterbots are in charge of managing a limited number of Bots. As stated earlier, when a new Bot introduces itself to the cloud, a C&C proxy selects/creates a Masterbot to assign to the new Bot; after that, the Bot will get connected to the cloud through its associated Masterbot. Every Masterbot has a parent Superbot. Masterbots with
a common parent form a cluster and know their siblings; they also construct an onion routing network to port the messages from Masterbots to the associated Superbot.

There is no fixed number of Masterbots in the cloud, and the number varies according to the number of Bots. As the number of Bots increases, the number of Masterbots increases as well to distribute the management load and also to decrease the potentiality of compromising the cloud. Masterbots can be dispersed on different cloud platforms. This platform diversity provides cloud providers with policy confliction, and this helps CBC2 to be more robust. Figure 4 shows the flowchart of Masterbot operations.

The messages from Masterbots to Superbots pass through an onion routing network of other Masterbots in CBC2. Every implementation of onion routing, such as Tor⁴¹, has two main components: OP (Onion Proxy) and OR (Onion Router). OP is a user-side application that receives data from users, puts it into cells, selects a path randomly to direct the cell, and sends the cell to the first OR in the selected path. Paths are called circuits in onion routing networks.

A circuit is constructed from different ORs, and cells move along them router by router to reach the destination. Each cell is called an onion because it has multi-layers of encryption over the main data; each OR picks off one of the layers from a cell, until it reaches to the last OR, which is also the destination that picks the last layer of encryption. For more information about Tor and onion routing⁴¹.

Masterbots play as OP and OR at the same time. When they receive a cell, they act as an OR, attempts to process the cell and do the right operation regarding the type of the message in that cell. Every Masterbot acts as an OP when it needs to send a request periodically to its associated Superbot to get the latest commands; Masterbots send requests along a randomly selected circuit, which is formed by ORs (other Masterbots in another word) and encapsulate the request message in an encrypted, multi-layered cell.

Since in CBC2 all of nodes are deployed as cloud applications, and cloud applications are web based, they rely on HTTP requests to be triggered; in these types of applications we cannot run a timer to specify the time period. To cope with this issue, a Masterbot saves the last time it has asked for the latest commands; then, as a LAST_COMMAND_REQUEST message comes from a Bot, the receiving Masterbot checks to see if the difference between the message arrival time and the saved time is more than the predetermined time period? If yes, it asks for the latest commands from the Superbot (through a LAST_COMMAND_REQUEST_FROM_MASTER message) and the new time is saved.

### 2.3.3 Superbot

Superbots stand between Masterbots and C&C proxies. New messages are sent (pushed) to Superbots from C&C proxies. Each Superbot has a C&C proxy as its parent and, on the other side, each C&C proxy has some Superbots as its children. The number of Superbots is not fixed, and they will get more as there are more Masterbots in the cloud. This dynamicity of Masterbots and Superbots along with being multi-clouded helps to produce more complex network of communications, which is hard to be discovered and interpreted. Figure 5 shows the flowchart of Superbot operations.
As demonstrated in Figure 5, Superbots always act as ORs. When a Superbot receives a cell, identifies the type of the message and chooses what to do, such as storing a value or putting a value on the output.

2.3.4 C&C Proxy

C&C proxies are delegates to the main C&C; the main C&C changes addresses of C&C proxies periodically to make them undetectable for Botnet detectors. To do so, the main C&C drops C&C proxy applications from the cloud account and creates new ones. There are more than one C&C proxy to distribute the work. One of responsibilities of C&C proxies is to receive messages from the main C&C and to push them to Superbots. They are also in charge of creating Masterbots and Superbots and holding the structure of CBC2 together. C&C proxies also assign Bots to Masterbots and assign Masterbots to Superbots (Figure 6. demonstrates these responsibilities as a flowchart). Another responsibly of C&C proxies is to fetch the uploaded data by Bots from the data cloud (this data had been sent to the data cloud by Bots). Once a C&C proxy receives a message as a cell, it checks the message type as the first step. If the message is a command (shown as LAST_COMMAND_PUSH in Figure 6.), then it saves and sends the message to Superbots. If the message type is FIRST_CONNECTION, then it first looks for a Masterbot with at least one free position (for Bots) to assign the first-time-executing Bot to the found Masterbot.

However, if no Masterbot is found, the C&C proxy will create a new Masterbot account on an arbitrary cloud; after that, it will assign the new Masterbot to a Superbot (if no Superbot with free position is found, a new one will be created and assigned to the Masterbot); at last, the new Bot will be assigned to the found/created Masterbot, and the ID of this Masterbot will be sent to the output. Once the C&C proxy receives a REINTRODUCE message, the associated Masterbot with the sender Bot will be verified; if the Masterbot is Off, the sender Bot will be treated as a first-time-executing one; otherwise, the Bot will be considered as a threat (see subsection 2.3.1) and will be removed from the cloud by cascading a DROP_BOT message.

C&C proxies have a monitor component inside them. The monitor is in charge of investigating the main parameters of the cloud, such as number of Masterbots under each Superbot, number of Bots under each Masterbot, the size of selection window (see 3.1) for each Masterbot, and the time unit period at which each Masterbot sends a request to its associated Superbot. The monitor oversees the frequency of Bot introduction to the cloud; considering the traffic rate and attempting to keep the traffic linear, the monitor changes the aforementioned parameters. We will discuss the effect of these parameters on traffic rate in subsection 5.2.2.

2.3.5 StartPoint

C&C proxies introduce new Bots to the cloud, but a direct connection from Bots to C&C proxies is dangerous because it can simply compromise current C&C proxies. For this reason, StartPoint nodes whose operations are shown in Figure 7. are defined to stand between C&C proxies and Bots.

There are some predefined StartPoint nodes in the cloud whose main duty is to connect the new coming Bots to the cloud through a C&C proxy. The addresses of these StartPoints are always being changed by the main C&C to make them more flexible against Botnet detectors. Each Bot contains a list of current StartPoints and a K’ and sends an encrypted message for one of StartPoints.

All of StartPoints are cloud applications containing two lists of all K’ (public key)s corresponding to Bots K; one of these lists is an archive, and the other one, which is called NonRunningBot, holds the K’s of Bots which have not introduced themselves to the cloud. Referring to Figure 7., when a StartPoint receives a cell with REINTRODUCE message type from a Bot, it looks in the NonRunningBot list to make sure if there is one corresponding K’ to decrypt the message; if found, so the message will be
authorized and will be ported to one of C&C proxies; if the message type is FIRST_CONNECTION, then the K will be removed from the NonRunningBot list to refuse to accept the Bot introduction one more time.

2.3.6 Main C&C

The Botnet owner sends her commands to the main C&C, and these commands will be spread to the cloud through the main C&C. C&C proxies send specification of all Bots, Masterbots, and Superbots added to the network to the main C&C to make it informed about the current network structure. The main C&C uses this information to recover the network as needed.

2.4 Message Types

In the previous subsection we referred some message types through the flowcharts. Because of having an integrated list of all message types, we bring them in Table 1.

![Figure 7. StartPoint operations.](image)

**Table 1. CBC2 message types**

| Message Type                  | Required Data                        | Description                                                                 |
|------------------------------|--------------------------------------|----------------------------------------------------------------------------|
| FIRST_CONNECTION              | IP and ID of the new bot             | this message will be sent when a Bot is executed for the first time on a compromised system; the new Bot will introduce itself to the cloud by sending this message. |
| LAST_COMMAND_REQUEST         | Last Timestamp                       | when a Bot wants to be aware of the last commands, it sends this message to its relevant Masterbot. |
| LAST_COMMAND_REQUEST_FROM_MASTER | ID of Masterbot, Last Timestamp | Masterbots check for the last commands periodically from Superbots. For this, they send this type of messages to Superbots. |
| NEW_SUPER_ADDED              | ID, Address, and Public Key of the new Superbot | when a new Superbot is added to the cloud (by a C&C proxy), all Masterbots must be aware of this; so this message is used to inform Masterbots about the new Superbot. |
| NEW_MASTER_ADDED             | ID, Address, and Public Key of the new Masterbot | when a new Masterbot is added to the cloud (by a C&C proxy), all Masterbots and Superbots must be aware of this; so this message is used to inform Masterbots and Superbots about the new Masterbot. |
| NEW_BOT_ADDED                | ID and Address of the new bot        | when a new Bot is added to the cloud (by a C&C proxy), one selected Masterbot must be aware of this; so this message is used to inform this Masterbots about the new bot. |
| LAST_COMMAND_PUSH           | Command string                       | when the main C&C wants to push a command to a C&C proxy, or a C&C proxy wants to push a command to Superbots, they use this message type. |
3. Onion Routing in CBC2

There are some directions of communication in the cloud: Main C&C ↔ C&C Proxies; C&C Proxies → Superbots; Superbots ← Masterbots. The main C&C and a C&C proxy send messages directly to each other. A C&C proxy also sends messages directly to Superbots. Yet Masterbots send all the messages as cells to Superbots based on the onion routing network; this network is based on random routing paths and produces a complex structure of communications that is hard to be understood for Botnet detectors.

In this design an OP is inside the cloud instead of being in the client-side, like Tor and other ordinary implementations of onion routing networks; this keeps the cloud from Botnet detectors because they cannot exert honeypot OPs to intrude into the cloud.

### 3.1 Constructing Circuits

When a cell is going to be sent from a Masterbot to a Superbot, the cell must move through a circuit. Each Masterbot constructs a new circuit for each cell; the algorithm is presented in Table 2. The Masterbot selects a random path through Masterbots in the same cluster. The destination of the circuit is always the parent Superbot. In line 1 of the algorithm the number of hops (intermediate Masterbots) will be specified that is a random number between 1 and the number of Masterbots in the same cluster. Line 2 empties the circuit list. Lines 3 to 5 select some unrepeated Masterbots by random to construct the circuit. Then in line 6 the given Superbot will be added to form the final destination of the circuit (Figure 8).

One of the major goals in CBC2 is to make the C&C center invisible or hardly visible to Botnet detectors; hence, it is not a good idea that each Masterbot freely selects any of other Masterbots in a cluster by random because after a while, it is possible that a Masterbot selects all other Masterbots in the same cluster. This phenomenon may reveal all the Masterbots in the same cluster and put the cloud into danger. To eliminate this possibility, we have reformed the algorithm such that every Masterbot is only allowed to select the Masterbots from its own window. Window is a set of Masterbots that can be selected for a circuit, and it is different for each Masterbot. Since each Masterbot has the list of all other Masterbots from the cluster where it belongs to, the Masterbot can specify its window by assigning a Min and Max to determine the start and end of its selection window, respectively (Figure 9). Since the Masterbot selects the Min and Max numbers by random, its window can be any portion of the main List. However, there is the limit that the length of the selection window cannot be larger than a particular fraction of the number of all Masterbots in the same cluster; the importance of this limitation will be known in subsection 5.2.2.

As a new Masterbot is added, each of the current Masterbots updates its selection window. Even by this improvement, we do not need to change the circuit construction algorithm: we should only modify the input

---

Table 1. Continued

| Message Type | Required Data | Description |
|--------------|---------------|-------------|
| NEW_DATA_CLOUD | Account information of a data cloud | when the main C&C changes the data cloud, Masterbots and all Bots will be informed by this message type (see subsection 5.2). |
| REPORT / REPORT_DATA | None | when a C&C proxy wants to collect data from a Bot, it sends this message type (see subsection 5.2). |
| REINTRODUCE | ID of bot | A Bot sends this message to a StartPoint when it cannot receive any response from its Masterbot. |
| NEW_STARTPOINTS | Address of StartPoints | Bots receive this message from their Masterbots to be aware of new StartPoint addresses to reintroduce themselves as needed (see subsection 5.2). |
| DROP_BOT | ID of Bot | C&C proxies send this message to Masterbots and StartPoints to delete a particular Bot from their associated Bot list. |
parameter of Masters in this way: "Masters: A list of Masterbots in the same cluster [Min-Max]".

## 3.2 Cells in CBC2

Cell is the unit of messaging in CBC2. Masterbots (in their Onion Proxy role) encrypt cells layer by layer according to the number of hops in the constructed circuit. The message data constitutes the core of a cell. Also, according to the number of circuit nodes, the Masterbot puts layers of ID and encryption upon the message. IDs are put to specify the steps of message transition until the message reaches its destination (Figure 10).

Every message has one layer of ID and encryption for each hop and destination in the circuit; this multi-layered structure makes the cell an onion. The multi-layered structure of the cell is demonstrated in Figure 11. This figure shows that there is a layer of encryption along with the ID of the next hop. The encryption is made by a symmetric cryptography method, such as Triple DES algorithm. All of Masterbots and Superbots are hard-coded with a key (triple keys $K_1$, $K_2$, and $K_3$ for 3DES) to encrypt and decrypt data.

### Table 2. Circuit construction algorithm

| Input: Masters: A list of Masterbots in the same cluster, Destination: A Superbot |
| Output: Circuit: A list of Masterbots + Destination |
| HopLength = Select a Random number between 1 and LengthOf (Masters) |
| Circuit = $\emptyset$ |
| For $I = 1$ to HopLength Do |
| $M = \text{An element from } \{ M \in \text{Masters} \mid M \notin \text{Circuit} \}$ |
| Append $M$ to Circuit |
| Append Destination to Circuit |
| Return Circuit |

[Figure 8. Circuit path.]

[Figure 9. Masterbot selection window.]

[Figure 10. Circuit and message encryption.]

Figure 11. shows that a message is comprised of two parts: header and body. Header specifies the type of the message and body carries required data for the message type. CBC2 defines some types of message presented before in Table 1. There is also a limitation on header and body: the sum of these two must not exceed 512 bytes for each cell; hence, if a message is larger than 512 bytes, it must be divided into more than one cell, and each cell could be sent through different circuits. Because many of messages carry public keys and other significant items, this mechanism helps not to lose them easily. It also makes racking the exchanged information in the cloud difficult for Botnet detectors.

The header consists of suitable information to put the received messages in the right order. As mentioned, each command issued from the main C&C has an ID; using this ID and a counter, the circuit destination can put the received messages in the right order. Then the structure of the header section will be as follows:
the main C&C signs its commands by its $K$; since the receiver C&C proxy has the corresponding $K'$, it can investigate whether the sender is the main C&C? On the other hand, Superbots have K’s of C&C proxies; hence, when they receive messages from C&C proxies, they can be sure of the origin. Also, since Superbots own a pair of $<K', K>$, they sign their messages by their K, and when a Masterbot asks for new commands, it decrypt the new command by the corresponding $K'$ to make sure the message is really from the selected Superbot. Applying this mechanism refuses any hijacking.

4.2 Monitoring by C&C

Sometimes the Botnet owner needs to know how many of its Bots are “ON” or to collect data (mostly spying data) from Bots. For these reasons, the main C&C sends REPORT and REPORT_DATA messages, respectively. Every time that the main C&C uses these commands, it creates a new data cloud account to collect the data. Thus, along with a report command, a NEW_DATA_CLOUD message is sent mostly. To make sure that all needed reports are sent from identified Bots, each Bot signs its related data by its private key ($K_-$), and the data is decrypted by its corresponding public key so as to make sure about the identity of the source.

After a limited and specific amount of time, the data cloud account is dropped (of course, after collecting the data requested by C&C). Then, another account is created, and the main C&C sends a new NEW_DATA_CLOUD message; this process continues for one day (24 hours) so that most of Bots from any place with various time differentiations can reply; the ones that have no response during one day are not counted as active Bots. If one Bot replies to the command once, it will not reply again. The entire process helps CBC2 to keep itself from compromising its data.

Monitoring process is not only limited to Bots; yet Masterbots, Superbot and C&C proxies must answer REPORT messages by their ID. The main C&C takes these report feedbacks into account to make some managerial decisions in different situations; for example see 5.4.

5. Evaluation

In order for evaluating CBC2 we consider some evaluation criteria then discuss about them.
5.1 Evaluation Criteria
Jian, et al.\(^{32}\) proposes an evaluation model along with some measurements for peer-to-peer Botnets. We have adapted the proposed criteria and metrics for CBC2 in this section. Four main criteria have been proposed in\(^{32}\): Stealthy, Effectiveness, Efficiency, and Robustness. Stealthy refers to the power of resistance against antiviruses and detector systems, and also to prevent the traffic and communications from being suspicious; Effectiveness refers to the destructive power of the Botnet; efficiency indicates the speed of transferring commands from Botmaster to Bots; robustness shows the stability degree of the Botnet architecture and considers how it can recover itself from crashes. Since effectiveness has a direct association with the number of Bots (in fact, the destructive power of a Botnet depends on the number of Bots under its control to perform the orders), we discard this criterion for CBC2 evaluation; on the other hand, there are so many different issues inside a Cloud node (cloud-internally)\(^{33}\) which can have side-effects on the whole CBC2 system that we ignore them in this paper.

5.2 Stealthy
Silva, et al.\(^{4}\) presents a taxonomy for Botnet detection techniques. They show a broad range of techniques can be used to detect Botnets, such as honey pots, signature detection, DNS monitoring, and host-based anomaly detection. All of these techniques need collaboration among users, network assistances and service providers. Using cloud platforms, such as CBC2 infrastructure, makes it much harder to construct such collaborations.

Monitoring cloud applications is impossible for a person/system outside the cloud; in addition, it is not reasonable for cloud service providers to monitor all of their applications because there can be thousands of applications running on a cloud, and providers cannot check all of their activities unless a particular abuse is reported. Yet another problem is that who can and will report an abuse, and how it can be reported? On the other hand, how the providers will response to these reports? Unfortunately, there are many unsolved regulation issues in cloud environments, which at the moment make cloud platforms suitable places for Botnet C&C.

In addition to the above mentioned issue, as stated in subsection 4.3, CBC2 can spread its nodes on more than one cloud platform and thus be more protected against Botnet detectors. Finally, since detection techniques are mainly based on network behaviors and features, to evade being detected, one can take into account some mechanisms and features\(^{34}\) as follows.

5.2.1 Communication Encryption
To resist against hijacking and intrusion detection, the Botnet architecture must use an encryption mechanism for its communications. CBC2 uses public key encryption mechanism when a particular message is to be sent from one node to another one to specify the true origin of messages as well as to protect messages from unauthorized detection. CBC2 also utilizes an onion routing mechanism to amplify the evasion power. This mechanism helps CBC2 to complicate the communication graph for detectors.

5.2.2 Communication Traffic
In the presence of Bots, communication traffic rate should not be changed dramatically to prevent suspicious behavior and thus to evade the Cloud from being detected. Generally, communication traffic implies the entire traffic needed for Bots to receive the commands; simply looking at CBC2 architecture (Figure 2.), first, we can say that the major traffic rate can be observed on Masterbots: on one hand, Bots send/receive messages to/from Masterbots, and on the other hand, Masterbots request for new information from Superbots. Second, the traffic should be considered for only one Masterbot because each node in the cloud is treated as an isolated application (the cloud provider is not supposed to know the relationships among these cloud applications), and thus suspicious behavior (or in fact, unusual traffic rate) is tracked per each application or Masterbot here.

The first part of the traffic rate for one particular Masterbot follows equation (1):

\[
T = N \times S
\]

Where \(T\) is the traffic rate between the Masterbot and its associated Bots, \(N\) refers to the number of these Bots, and \(S\) is the average size of command messages (including encryption, header, and body).

The second part of the traffic rate occurs when the Masterbot reaches its time period (see subsection 2.3.2). At this time, it must send a request to its associated Superbot to get the latest commands. In this situation we need to estimate the traffic rate from Masterbots to their Superbots approximately. In the worst case, an onion routing path which is responsible for transmitting messages from Masterbots to Superbots consists of the
entire Masterbot selection window; since the selection window can have a fraction of Masterbots under a particular Superbot, the traffic rate from a Masterbot to its Superbot in the worst case will be:

\[ T_m = \frac{M}{f} \times S \]  

(2)

Where \( T_m \) is the traffic rate from a Masterbot to its associated Superbot, \( M \) is the number of Masterbots under a particular Superbot, \( S \) is the average message size, and \( f \) specifies the fraction of Masterbots which forms the selection window for each Masterbot. For instance, \( f=5 \) means each Masterbot can construct a circuit from at most one fifth of Masterbots at the same cluster. Smaller values of \( f \) will result in more complex circuit paths which will be harder to be detected by Botnet detectors. On the other hand, according to equation (2), smaller values of \( f \) bring into more traffic rate and thus more suspicious behavior. Regarding such a tradeoff, the suitable value of \( f \) should be tuned by the Botnet owner in the operational environment.

Since equation (2) should not be considered every time a Bot asks for commands, it must be amortized according to the number of messages from a Bot to its Masterbot. As a result, we present an equation to show the traffic rate per one hour:

\[ T_{total} = \frac{60}{H_b} \times T + \frac{60}{H_m} \times T_m \]  

(3)

Where \( T_{total} \) refers to the total traffic rate for a particular Masterbot. Each Bot asks for new commands from its associated Masterbot every \( H_b \) time unit, and each Masterbot asks for new commands from its associated Superbot every \( H_m \) time unit; \( H_b \) and \( H_m \) are defined in seconds.

Configuring parameters in equation (3), we can always keep the traffic at a low level. For instance, we can always choose particular values for parameters to keep the traffic approximately lower than 2MB; having this example in place, if we try to solve equation (3), we can have it as the following:

\[ 2MB \leq \frac{(H_m (f)(N)+(H_b (M))}{3600 (S) \times \frac{H_b (f)}{H_m (f)}} \]  

(4)

Regarding inequality (4), the monitor component of a C&C proxy should keep the values of parameters in such a way that this inequality always holds (It should be noted that \( H_b \) is the dynamicity of the systems and is not under-control).

5.2.3 Communication Maintenance

To keep the Botnet stable, the Botnet owner needs to know which Bots are either offline or blocked (how many are accessible). The main C&C sends a REPORT message periodically, and Bots reply to this message. In this way, the main C&C can check the number of currently online Bots at any time slot. Details were discussed in subsection 4.2.

5.3 Efficiency

Efficiency is the required time for a particular message to be received by a particular Bot and thus depends on the diameter of the Botnet (from server-side to client-side). If we suppose set \( B = \{d_1, d_2, \ldots, d_n\} \) as the set of distances from the main C&C to each of \( n \) Bots, it seems that we can find the diameter easily; but the problem is that CBC2 has a random structure, and there is no specific path from the main C&C to any Bot.

Nevertheless, since mainly every Bot receives commands from its direct Masterbot, efficiency will be high because we only need a simple HTTP request; yet sometimes the request will be held because the Masterbot reaches its time period to ask its Superbot. Therefore, to evaluate efficiency, we need to find the longest possible path from the main C&C to a Masterbot and amortize the taken time to the Bots' requests number because once the Masterbot becomes up to date, it is updated for all of Bots' requests. Suppose \( L \) is the size of the biggest set of Masterbots under a Superbot. Thus, we have:

\[ P = \frac{L}{f} + 2 \]  

(6)

Where \( P \) is the length of the longest path, \( f \) specifies the fraction of Masterbots which forms the selection window for each Masterbot and from that C&C proxy to a Superbot; although, in practice we can eliminate value 2.

In summary, the highest number of required messages (on which the required time depends) for Bots to be aware of commands is normally constant (or better saying one; it means that most of the time the commands reach to Bots through only one hop path) unless the Masterbot reaches
5.4 Robustness

Robustness must include sustainability\(^{35}\); a good definition of sustainability is the one that captures the effect of elimination of a specific node from the whole Bot network\(^{36}\). Since there are different types of nodes in CBC2, their elimination will have different effects on the network. To study these effects, we must know what portions of the network each node will disclose. Table 3 shows the knowledge of each type of node about others in CBC2.

A typical robustness study is to eliminate different nodes and watch the reaction of Botnet architecture in each situation\(^{36,37}\). Some nodes may be inaccessible because of diurnal activities\(^{38}\), and some may be blocked by defenders, but for simplicity, we consider both types the same. Since all of Bots are the same, the effects of a blocked Bot do not make any difference with those of another one.

### Table 3. Information saved in each of cloud nodes

| Main C&C: | L1 = \{(CP_1, ID_1, K_1^0), (CP_2, ID_2, K_2^0), ..., (CP_p, ID_p, K_p^0)\} |
|-----------|---------------------------------------------------------------------|
| L2 = \{(S_1, ID_1, K_1^1), (S_2, ID_2, K_2^1), ..., (S_q, ID_q, K_q^1)\} |
| L3 = \{(M_1, ID_1, K_1^0), (M_2, ID_2, K_2^0), ..., (M_m, ID_m, K_m^0)\} |
| L4 = \{(B_1, ID_1, K_1^0), (B_2, ID_2, K_2^0), ..., (B_b, ID_b, K_b^0)\} |
| L5 = \{(SP_1, ID_1, K_1^0), ..., (SP_s, ID_s, K_s^0)\} |

CP refers to the \(i\)th C&C Proxy, ID refers to the identification number of this C&C proxy, and \(K_i^0\) is the public key of this C&C proxy. The main C&C needs the information of its C&C proxies to get connected to. Of course, the main C&C stores the list of all Bots, Masterbots and Superbots for recovery. \(S\) refers to Superbot, \(M\) refers to Masterbot, and \(B\) refers to Bot. \(cp\) refers to the number of all C&C proxies, \(s\) refers to the number of all Superbots, \(m\) refers to the number of all Masterbots, and \(b\) refers to the number of all Bots.

| C&C Proxy: | L1 = \{(CC, ID, K')\} |
|-------------|------------------------------------------------------------------|
| L2 = \{(S_1, ID_1, K_1'), (S_2, ID_2, K_2'), ..., (S_q, ID_q, K_q')\} |
| L3 = \{(M_1, ID_1, K_1'), (M_2, ID_2, K_2'), ..., (M_m, ID_m, K_m')\} |
| L4 = \{(B_1, ID_1, K_1'), (B_2, ID_2, K_2'), ..., (B_b, ID_b, K_b')\} |
| L5 = \{(SP_1, ID_1, K_1'), ..., (SP_s, ID_s, K_s')\} |

\(CC\) refers to the main C&C, \(S\) refers to Superbot, \(M\) refers to Masterbot, \(SP\) refers to StartPoint, and \(B\) refers to Bot. \(s_p\) refers to the number of Superbots under a particular C&C proxy, \(m_p\) refers to the number of Masterbots under these Superbots, \(b_p\) refers to the number of Bots under these Masterbots, and \(sp\) refers to the number of all StartPoints.

| Superbot: | L1 = \{(M_1, ID_1, K_1'), (M_2, ID_2, K_2'), ..., (M_m, ID_m, K_m')\} |
|-----------------------------|------------------------------------------------------------------|
| L2 = \{(B_1, ID_1, K_1'), (B_2, ID_2, K_2'), ..., (B_b, ID_b, K_b')\} |
| L3 = \{(M_1, ID_1, K_1'), (M_2, ID_2, K_2'), ..., (M_m, ID_m, K_m')\} |

\(M\) refers to Masterbot. \(m_p\) refers to the number of Masterbots under a particular Superbot.

| Masterbot: | L1 = \{(S_1, ID_1, K_1'), (S_2, ID_2, K_2'), ..., (S_q, ID_q, K_q')\} |
|------|------------------------------------------------------------------|
| L2 = \{(B_1, ID_1, K_1'), (B_2, ID_2, K_2'), ..., (B_b, ID_b, K_b')\} |
| L3 = \{(M_1, ID_1, K_1'), (M_2, ID_2, K_2'), ..., (M_m, ID_m, K_m')\} |

\(S\) refers to Superbot, and \(s\) is the index of the associated Superbot; \(B\) refers to Bot, and \(M\) refers to Masterbot. \(b_p\) refers to the number of Bots under a specific Masterbot, and \(m_p\) refers to the number of Masterbots under a specific Superbot.

| StartPoint: | L1 = \{(CP_1, ID_1, K_1'), (CP_2, ID_2, K_2'), ..., (CP_p, ID_p, K_p')\} |
|-------------|---------------------------------------------------------------|
| L2 = \{(B_1, ID_1, K_1'), (B_2, ID_2, K_2'), ..., (B_b, ID_b, K_b')\} |

CP refers to C&C Proxy, and \(B\) refers to Bot. \(cp\) refers to the number of all C&C proxies, and \(b\) refers to the number of all Bots.

| Bot: | L1 = \{(M_1, ID_1, K_1')\} |
|-----|----------------------------------|
| L2 = \{(M_2, ID_2, K_2'), ..., (M_m, ID_m, K_m')\} |

\(M\) refers to Masterbot, and \(m\) is the index of the associated Masterbot.
Inaccessible Bots will not destroy the communication of other Bots to the cloud; thus, merely blocking one particular Bot does not affect the Bot network.

What detectors mainly do is to cut the communications among Bots and C&C\(^9\). Since the communications in CBC2 is through the cloud, we must consider aggressive defenses against cloud-side nodes. Detecting a C&C proxy should not be considered as the most aggressive movement because C&C proxies are changing their addresses all of the time, and thus, they are not accessible to be detected.

Let us have a deeper analysis: CBC2 has a structural organization; if a branch in CBC2 is removed or deactivated, then some of the running client-side Bots cannot connect to their Masterbots; in this situation the only thing happens is that, those Bots reintroduce themselves to the cloud (through sending REINTRODUCE message to one of StartPoints), and new branches are created simply; regarding this case, it should be recalled that dropping and creating an account for a cloud application is very cheap in cloud environments.

Now two questions are raised: is there always at least one StartPoint to introduce or to reintroduce a bot? What will happen if none of StartPoints responds? To answer both questions, we should say that the worst case scenario is that all of StartPoints are deactivated somehow, and at the same time, all of Superbots and their associated Masterbots are deactivated; in this case, running Bots cannot receive new StartPoint addresses, and thus, they will be disconnected from the cloud because they cannot reintroduce themselves. However, we demonstrate that CBC2 can configure itself somehow to reduce the probability of being detected.

Each Bot is added to the network every \(H_j\) time unit in average and introduces itself through some StartPoint; on the other hand, a Botnet detector must find/block the StartPoint before its domain name expires; let \(H_{sp}\) be the life time of the StartPoint domain name. There must be an appropriate tradeoff for picking \(H_{sp}\): it should be short enough to disallow detectors to find/block the StartPoint, and on the other side, it should be long enough not to put Masterbots in the starvation situation (because Masterbots need to be aware of current StartPoint domains periodically).

Figure 13. shows a timeline that specifies the time points where detectors can find the StartPoint.

Say \(C\) is the required time for a detector to find the StartPoint; As Figure 13. shows, \(C\) is the difference between \(H_{sp}\) and \(H_j\), so it implies the following equation:

\[
C = H_{sp} - H_j
\]

\(C\) is the average of \(H_{j1}\) to \(H_{jn}\), the equation will be as follows:

\[
C = H_{sp} - H_j
\]  

Closer values of \(C\) to zero turn into the less chance for detectors to have StartPoints detected. According to equation (9), the monitor component can oversee \(C\) value according to \(H_j\) and decide to pick an appropriate \(H_{sp}\) so as to make \(C\) as close as possible to zero. Figure 14. shows that these parameters must be set in such a way that keeps the cloud from detectors. According to equation (9) \(C\) must tend to zero, then Figure 14. shows an example of this setting.

Suppose Masterbots ask for new commands from Superbots each \(H_m\) time unit; and Bots ask for new commands from Masterbots each \(H_s\) time unit. Figure 14. shows a time line graph for Masterbots request sending, Bots request sending, and StartPoint domain changes. As this figure shows, with an appropriate time slot, it is possible to bring about zero time unit distance between Masterbots awareness and Bots awareness of new StartPoint domains (the circled portions).

As mentioned before, the entire Botnet can be disabled if all of StartPoints along with all of Masterbots of all of Superbots are detected and disabled. In any other situations the network will continue its operations. For instance, if all of StartPoints are disabled (yet Masterbots and Superbots are not detected), then new ones will be created by Botnet owner, their address will be sent to the cloud, and \(H_{sp}\) will be diminished to lower the detectors chance. On the other side, if all of Masterbots are disabled (even though along with Superbots and some StartPoints),
$H_{sp}$ will be diminished dramatically to take any time from detectors.

6. Possible Countermeasures

In this section we propose some possible countermeasures against CBC2; we will expand these issues as our future work.

6.1 Honey Cloud

The Honeypot mechanism along with blocking Bots and even blocking cloud-side nodes are not complete approaches because CBC2 can recover itself simply (see subsection 5.5). The main mechanism to fight with BotClouds is to let them grow in under-control cloud computing platforms which are provided specially as free suitable places for attackers; because of free infrastructure proposed by HoneyClouds, the chance that attackers use them to run their attacks is high. In fact, we must attract the attackers to place their Botnet on our HoneyClouds.

On the other hand, since BotClouds may use cloud accounts as their Bot army to run a DDoS attack or any other type of attacks, and the main difference among HoneyClouds and ordinary cloud platforms is that they provide limited number of cloud accounts to control their traffic completely and to find Botnet C&C, HoneyClouds can be considered as good candidates to control and block BotClouds. Moreover, HoneyClouds can also distinguish various behavioral patterns from different BotClouds so as to provide them for cloud computing service providers to block users with pre-recognized behavior patterns.

HoneyClouds may be provided by security corporations, governments and any other societies. Of course, there must be many light HoneyClouds to elevate the chance that attackers use them as their desired infrastructure.

6.2 Cloud Alliance

At the moment, there are many cloud platforms (either free or commercial), and creating one or more accounts is really cheap and easy (there are even cloud platforms on which users can create account freely with only one email); this situation provides a safe place for attackers.

Therefore, no single approach can be effective enough against BotClouds by itself because cloud accounts are easily accessible with free or low costs, and attackers can deploy new Botnets simply upon other cloud platforms. An alliance among cloud service providers can be a turning point on the way of fighting with BotClouds. There must be a universal mechanism to automatically report attacks from different cloud platforms.

7. Conclusions and Future Work

To be prepared for future malicious systems, such as advanced Botnets, we must anticipate different mechanisms Botnet designers may use. BotCloud (and Botnet in general) is one of major problems in today’s Internet; hence, in this paper we attempted to propose a Botnet design utilizing cloud computing as its infrastructure and exerting onion routing mechanism to make its main C&C undetectable. The proposed design provides robust server-side communication and recovers easily from failing nodes. In future, we want to spend more research to propose more effective approaches for detecting BotClouds in ordinary cloud computing platforms and to propose an automatic inter-cloud anti-BotCloud solution.

8. References

1. Durairaj M, Manimaran A. A study on security issues in cloud based e-learning. Indian Journal of Science and Technology. 2015 Apr; 8(8):757–65.
2. Boopathi K Sreejith S, Bithin A. Learning cyber security through gamification. Indian Journal of Science and Technology. 2015 Apr; 8(7):642–9.
3. Clark K, Warnier M, Brazier FMT. BotClouds, The Future of Cloud-based Botnets. 1st International Conference on Cloud Computing and Services Science; 2011.
4. Silva SSC, et al. Botnets: A Survey, Computer Networks. 2013; 57.
5. Sathish V, Abdul Khader SP. Deployment of proposed botnet monitoring platform using online malware analysis for distributed environment. Indian Journal of Science and Technology. 2014 Aug; 7(8): 1087–93.
6. EggHeads, EggHeads.org-eggdrop development. 2015 Jul 22. Available from: http://eggheads.org/
7. Bacher P, Holz T, Kotter M, Wicherski G. Know your enemy: Tracking botnets (using honeynets to learn more about Bots). Technical Report, The Honeynet Project. 2008.
8. Macesanu G, Codas T, Suliman C, Tarnauca B. Development of GTBot, a high performance and modular indoor robot. IEEE International Conference on Automation Quality and Testing Robotics (AQTR); 2010. p. 1–6.
9. Dumbledore, Well Known Bot Families. 2015 Jul 22. Available from: http://dumbledore.hubpages.com
10. Micro T. Worm SDBot, 2015 Jul 22. Available from: http://aboutthreats.trendmicro.com/ArchiveMalware.aspx?language=us&name=WORMSDBOT.AZ.

11. Micro T. Worm AgoBot, 2015 Jul 22. Available from: http://about-threats.trendmicro.com/ArchiveMalware.aspx?language=us&name=WORMAGOBOT.XE.

12. Symantec, W32.Gaobot.CEZ, 2015 Jul 22. Available from: http://www.symantec.com.

13. Li C, Jiang W, Zou X. Botnet: survey and case study. Fourth International Conference on Innovative Computing, Information and Control (ICICIC); 2009. p. 1184–7.

14. Symantec, Spybot worm, 2015 Jul 22. Available from: http://www.symantec.com/securityresponse/writeup.jsp?docid=2003-053013-5943-99

15. Keizer G. Top Botnets control hijacked computers, 2015 Jul 22. Available from: http://www.computerworld.com

16. Stover S, Dittrich D, Hernandez J, Dietrich S. Analysis of the storm and nugache: P2p is here. Proceedings of the 4th USENIX Workshop on Cyber Security Experimentation and Test (CSET’11), USENIX Association; 2007.

17. 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. Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats, USENIX Association. Berkeley, CA, USA. 2008.

18. Symantec, Message labs Intelligence. Security Response, Symantec. 2010.

19. Davis C, Fernandez J, Neville S, McHugh J. Sybil attacks as a mitigation strategy against the storm Botnet. 3rd International Conference on Malicious and Unwanted Software, MALWARE 2008, 2008; p. 32–40.

20. Wilson T. Competition may be driving surge in Botnets, spam, 2015 Jul 22. Available from: http://www.darkreading.com

21. Experts bicker over conficker numbers. 2015 Jul 22. Available from: http://news.techworld.com

22. Symantec, The downaduo codex. Security Response, Symantec, 2009.

23. Tdl-4 top Bot, 2015 Jul 22. Available from: http://www.securelist.com

24. Keizer G. Top Botnets control hijacked computers. 2015 Jul 22. Available from: http://www.computerworld.com

25. Spam Botnets to watch in 2009. 2015 Jul 22. Available from: http://www.secureworks.com

26. Miller C. The Rustock Botnet Spams Again. 2008.

27. Messmer E. America’s 10 Most Wanted Botnets. 2015 Jul 22. Available from: http://www.networkworld.com

28. Symantec, Zeus: King of the Bots. Security Response, Symantec, 2008.

29. Stevens K, Jackson D. Zeus Banking Trojan Report. 2015 Jul 22. Available from: http://www.secureworks.com

30. Boshmaf Y, et al. Design and analysis of a social Botnet. Computer Networks. 2012.

31. Dingleidine R, Mathewson N, Syverson P. Tor: The Second Generation Onion Router. Available from: https://svn.torproject.org/svn/projects/design-paper/tor-design.html

32. Jian G, et al. An Evaluation model of Botnet based on peer to peer. Fourth International Conference on Computational Intelligence and Communication Networks; 2012.

33. Jeon H, Min YG, Seo KK, A performance measurement framework of cloud storage services. Indian Journal of Science and Technology. 2015 Apr; 8(8):105–11.

34. Jian G, et al. An Evaluation model of Botnet based on peer to peer. Fourth International Conference on Computational Intelligence and Communication Networks; 2012.

35. Leonard J, Xu S, Sandhu R. A Framework for Understanding Botnets. International Conference on Availability, Reliability and Security, 2009.

36. Albert R, Jeong H, Barabasi AL. Error and attack tolerance of complex networks. Nature. 2000.

37. Yu J, et al. Structural Robustness in Peer to Peer Botnets. International Conference on Networks Security, Wireless Communications and Trusted Computing; 2009.

38. Dagon D, Zou C, Lee W. Modeling Botnet propagation using time zones. Proceedings of 13th Annual Network and Distributed System Security Symposium (NDSS); 2006. p. 235–49.

39. Micro T. Taxonomy of Botnet Threats, Threats. Technical Report, Trend Micro White Paper; 2006.