Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Computer Science and Engineering

BACHELOR PROJECT ASSIGNMENT

Student: **Marek Timr**

Study programme: Open Informatics
Specialisation: Software Systems

Title of Bachelor Project: **Framework for proof of concept implementations of C&C channels**

Guidelines:

The student should study characteristics of command and control (C&C) communication in botnets and design a simple C&C meta-protocol containing basic commands to control a botnet. Furthermore, the student will implement a framework in the Java programming language that will allow for testing of implementations of the proposed C&C protocol via selected web services such as Twitter, Dropbox, etc. Finally, the student will evaluate the complexity of misusing of the selected services for C&C communication.

Bibliography/Sources:

[1] Silva, Sergio S. C. et al.: Botnets: A survey, The International Journal of Computer and Telecommunications Networking, New York, 2013
[2] Bloch Joshua, Effective Java, Prentice Hall, New Jersey, 2008
Aktuální časopisecká literatura z oblasti botnetů

Bachelor Project Supervisor: Mgr. Jan Kohout

Valid until the end of the summer semester of academic year 2015/2016

doc. Ing. Filip Železný, Ph.D.
Head of Department

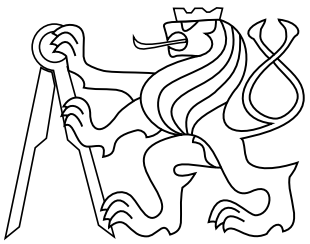
prof. Ing. Pavel Ripka, CSc.
Dean

Prague, March 26, 2015

Bachelor's thesis

Framework for Proof of Concept Implementations of C&C Channels

Marek Timr



May 2015

Supervisor: Mgr. Jan Kohout

Czech Technical University in Prague
Faculty of Electrical Engineering, Department of Computer Science

Acknowledgement

I would like to thank my supervisor Jan Kohout for supporting me and my work and for providing advice and insight into the subject matter.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne _____

_____
Podpis autora práce

Abstrakt

Botnet je síť kompromitovaných počítačů, které jsou vzdáleně ovládané skupinou hackerů. V poslední době se stávají velmi vážnou hrozbou pro internet. Tyto sítě jsou vytvořené k provádění rozsáhlých nelegálních aktivit zahrnujících krádež citlivých dat a útoků na soukromé i veřejné služby. Klíčovou součástí botnetu je Command and Control (C&C) kanál, který útočníci používají k zadávání příkazů botům and sbírání dat. V naší práci prezentujeme framework navržený pro testování implementací komunikačních protokolů. Zaměřili jsme se na myšlenku zneužití běžně používané internetové služby ke skrytí C&C centra botnetu. Komunikace takového botnetu se tak ukryje v běžném síťovém provozu. Ukážeme naší implementaci meta-protokolu komunikace s několika službami jako je například Twitter či Dropbox a rozebereme možnost jejich zneužití.

Klíčová slova

botnet, Command and Control, Command and Control simulace

Abstract

A botnet is a network of compromised computers which are remotely operated by a group of hackers. They have become a serious threat to the Internet recently. These network are created to conduct large-scale illegal activities including stealing of sensitive information and attacks on private and public services. Crucial part of botnet is Command and Control (C&C) channel that is used by attackers to issue commands to bots and collect data. In our work we present a framework designed to test implementations of communication protocols. We focused on the idea of abusing commonly used Internet service to hide C&C center of botnet. Communication of such botnet is hidden in regular network traffic. We will show our implementations of communication meta-protocols with some services such as Twitter or Dropbox and analyse possibilities of their abuse.

Keywords

botnet, Command and Control, Command and Control simulation

Contents

1 Introduction **1**
1.1 Motivation . . . 1

2 Botnet structure **3**
2.1 C&C protocol . . . 3
2.2 Usage . . . 3
Spam . . . 3
DDoS attacks . . . 3
Identity theft and sensitive data stealing . . . 3
Click fraud . . . 4
2.3 Topology . . . 4
2.3.1 Centralized . . . 4
IRC . . . 4
HTTP . . . 5
2.3.2 Decentralized . . . 5
Peer-to-peer . . . 6

3 Examples of Botnets **7**
3.1 Koobface . . . 7
C&C channel . . . 8
3.2 Gameover ZeuS . . . 8
P2P architecture . . . 9
Message structure . . . 9
3.3 Stegobot . . . 10
C&C channel . . . 10
Effectiveness . . . 11

4 Framework implementation **13**
4.1 C&C channel . . . 13
4.2 Design of framework . . . 13
4.2.1 Flow of the framework . . . 14
4.2.2 List of orders . . . 15
4.2.3 Configuration file . . . 15
4.3 Behaviors . . . 16
4.4 Drivers . . . 16
4.4.1 OAuth . . . 16
4.5 Dropbox . . . 18
4.5.1 Proposed channel . . . 18
4.5.2 Limitations . . . 18
4.5.3 Setup . . . 20
4.5.4 Summary . . . 20
4.6 Pastebin . . . 20
4.6.1 Proposed channel . . . 21

4.6.2 Limitations . . . 21
4.6.3 Setup . . . 21
4.6.4 Summary . . . 22
4.7 Twitter . . . 22
4.7.1 Proposed channel . . . 23
4.7.2 Limitations . . . 23
4.7.3 Setup . . . 23
4.7.4 Summary . . . 24
4.8 Google Spreadsheet . . . 24
4.8.1 Channel design . . . 24
4.8.2 Limitations . . . 25
4.8.3 Setup . . . 25
4.8.4 Summary . . . 25
4.9 Hybrid channel . . . 26
4.9.1 Channel design . . . 26
Commanding orders . . . 26
Responding to C&C . . . 27
4.9.2 Limitations . . . 27
4.9.3 Setup . . . 27
4.9.4 Summary . . . 28
4.10 Comparison of channels . . . 28
4.11 Drop off server . . . 29
4.12 Extension of framework . . . 29

5 Conclusion **31**

Appendices

A Content of CD **32**

Bibliography **33**

1 Introduction

Botnet is a network of private computers that were infected by malicious software in order to control them. Bot, which is short for robot, is a software application or script performing malicious activity. The infected computer is called zombie. The network is controlled by one or more attackers who are called botmasters. Bots allow botmaster to control zombie and perform a whole range of criminal activities.[1]

Botnet can spread over thousands of machines. The largest recorded controlled millions of victims. Unlike regular malicious software like viruses or Trojans, the main power of bot is its ability to cooperate with others. The crucial part of a botnet is Command and Control (C&C) channel. Botmaster uses this channel to issue commands to bots and harvest stolen information gathered by bots. Botnets employ different strategies how to establish robust C&C channel since it's the weakest part of it. Disruption of C&C takes down the whole botnet. Therefore the communication is often covert and encrypted to prevent interference with botnet. Also topologies of networks made by bots are various as we will see in the following chapter.

1.1 Motivation

Bots may use its own protocol for communication but also make use of common protocols. Recently, botnets tend to use HTTP for their communication. HTTP is massively used on the Internet and has a few significant advantages. Unlike for example formerly popular Internet Relay Chat (IRC) protocol, HTTP communication cannot be easily blocked by firewalls. HTTP overcomes port filtering. The next advantage is the ability to hide in regular traffic.

Botnet contains one or more C&C servers which are operated by botmasters. In our work we focus on the idea of hiding C&C communication into a usage of regular Internet service. Bot will connect to a commonly used social network or similar popular service to communicate with the botmaster. This approach has inherently the same advantages as HTTP botnets but adds a few more. Bots connect to well known servers which cannot be shut down, blacklisted or blocked. Communication is transferred encrypted by HTTP over TLS, HTTPS for short. Meaning that we cannot perform deep packet inspection of captured traffic. We only know the destination and size of the packet but we are missing request part and payload. This communication is blended in with regular traffic made by user of the infected computer. This principle is called hiding in plain sight.

We were inspired by the idea of bot abusing certain Google Services [2] to test vulnerabilities of some commonly used services. For that purpose we created a

framework called CCCBot which simulates C&C communication between a bot and C&C center located on a service. Besides studying possible abuse of selected Internet services, the framework will be a source of valuable data. It is hard to obtain real captured traffic of a botnet. Data created by our framework can be used for learning of a botnet detection software.

2 Botnet structure

In this chapter we will describe what possible malicious activities botnets are capable of carrying. Also we will study different botnet topologies.

2.1 C&C protocol

C&C communication protocol is a set of rules for exchanging information between bot and botmaster. Protocol determines the medium over which is information transferred and establishes data representation of messages and their meaning. Creator of botnet equips bots with sets of actions which they are capable of. Protocol must allow botmaster to issue orders to execute these actions. Protocol also handles distribution of messages across the network of bots in case of complex hierarchy. [3]

2.2 Usage

Botnets are used in a wide variety of malicious activities, usually generating financial profit for attackers. Botmasters can also conduct serious damaging attacks. Following list classifies common usage of botnets as stated in this paper[4].

Spam

Spamming is a very common use of botnets. About 70% to 90% of the world's spam is caused by botnets nowadays [4]. The statistics[5] show that a single botnet can produce 40 billion emails per day. In this case done by botnet called Grum. Similarly these botnets can be used to spread various malware.

DDoS attacks

Distributed denial-of-service attacks are attempts to make a service or network resource unavailable for intended users. There are several methods of attack exploiting various characteristics of communication protocols. Common method of attack is based on flooding service with enormous amount of requests, which the target cannot handle. It is quite difficult to come up with a countermeasure. The traffic cannot be simply filtered because it comes from various locations.

Identity theft and sensitive data stealing

Botnets are used in phishing attempts to obtain confidential information. For instance, botnet can create spam directing users to fake websites imitating real

ones. The victim is prompted to log in or has to input some personal information. This is commonly used to steal bank accounts. Botnets may sniff traffic passing by victim's machine or act like keyloggers collecting passwords for accounts of the user.

Other uses include instant messaging attacks. Bot steals victim's IM accounts and spreads messages that look legitimate. Their purpose is to lure others to phishing websites in order to download some malware. The bot is profiting from the fact that users trust the account that they know.

Click fraud

Botnets make use of unique IP of host for creating clicks on Internet ads. Whenever someone clicks on an ad, the advertiser pays the website owner some amount of money. Perpetrator may set up a botnet to periodically click on certain ads. Since the IP addresses of infected hosts are scattered around the world, the clicks seem valid. This damages payers of the advertisement. The attacker can also use automated clicks to affect polls or games.

2.3 Topology

C&C channels may be classified according to the arrangement of network. The main categories are centralized and decentralized. There are also hybrid combinations of those[1].

2.3.1 Centralized

Centralized topology is similar to client-server network model. Each bot connects to one, or a few, C&C servers. This design is easy to implement and comes with some benefits. Centralized botnets have quick reaction times and controllers are able to easily monitor the status of botnet. This provides them with information about number of active individuals and their distribution. Centralized topology suffers from a significant drawback. Failure of C&C center is failure of the whole botnet. Therefore attackers may insert an additional proxy layer of computers between workers and C&C server. Centralized botnets commonly use protocols as IRC[5]and HTTP.

IRC

Internet Relay Chat is protocol for sending text based messages. Users need a client that connects to a chat server. The communication is grouped into discussion forums, called channels. Botmaster creates a channel and broadcast messages there. Bots read them and report in this channel directly back to C&C server. This C&C channel is push-based. That means that bots are notified about a new order and do not have to poll for new commands. Managing bots into groups allows the botmaster to issue different commands to parts of botnet.

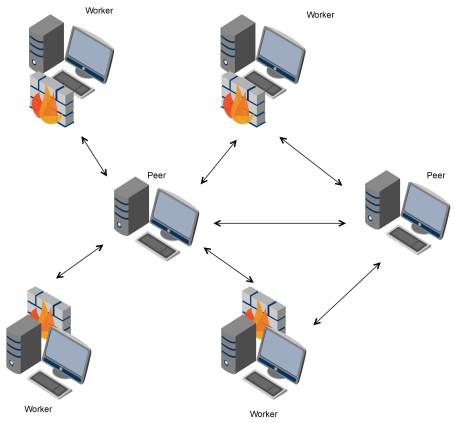


Figure 1 Peer-to-peer architecture with workers hidden behind firewalls

IRC was very popular among its users and also creators of botnets. A great amount of botnets utilizes IRC protocol but they are gradually replaced by HTTP-based botnets. Popularity of IRC continuously decreases in recent years. Usage of IRC is not very common in corporate networks. Communicating through IRC may not be permitted in these networks. However HTTP is usually allowed and gives botnet opportunity to hide in regular communication.

IRC protocol is mainly plain text. Therefore traffic can be collected and simply analyzed. IRC based botnets came with various strategies how to remain undetected. They may send obfuscated messages or use their own dialect. There are some options how to encrypt communication. Some IRC servers support SSL/TLS.

HTTP

HTTP-based botnets[6] act oppositely to IRC botnets. HTTP C&C channels are pull-based. Bot has to poll C&C server periodically to acquire new orders. As was said, it is not easy to block HTTP in networks. Normal application installed on computers may generate periodical activity which may trigger false positive detection of botnet. These applications may be email clients updating mailboxes, auto-updaters or other software synchronizing with server. This gives HTTP based botnets the advantage to hide.

2.3.2 Decentralized

Decentralized C&C is an attribute of modern botnets providing them with the ability to handle large numbers of bots. Decentralized topology is more flexible and robust[7]. Bots do not establish communication with one point, thus making uncovering botnet harder. Even if a significant part is revealed, taking down botnet is hard. Attacker can spread orders to bots from almost any part of botnet.

Peer-to-peer

Decentralized architecture is based on some kind of Peer-to-peer (P2P) overlay. Peer-to-peer based botnets are more resilient and harder to detect. Each bot in network can be peer or a worker. Peers are nodes that are connected to other peers and spread commands to workers. It depends on botnet topology and strategy how peer nodes are chosen and distributed in network and also on how the orders are distributed in the network.

Not every infected machine can be peer. If the computer is behind a firewall or its address is translated by NAT, that machine cannot be reachable by IP address. The computer may also access internet through a proxy. The figure Fig. 1 shows an example of peer-to-peer network. Workers are shared by different peers, allowing them to be assigned to a different peer in case the used one is taken down. Peers also serve the purpose of proxy. Worker communicating with a peer does not reveal true C&C server issuing commands. That also works both ways. Botmaster issues a command only to one peer, which distributes it to others.

3 Examples of Botnets

In this section we will present different examples of botnets and their C&C channels. We chose three botnets, each with contrasting architecture or an interesting C&C channel.

3.1 Koobface

Koobface [8] is very successful botnet first detected at the end of 2008. It targeted messaging networks like GMail, Skype, Yahoo Messenger and social networks like Facebook, Twitter or MySpace. Koobface was operable for long time despite numerous attempts to take it down till 2012 when the gang of five hackers was identified and arrested. Koobface mainly generated profit through pay-per-click and traffic referral schemes earning millions of dollars for attackers.

Koobface used social engineering to spread. Infected computer sends bogus messages using hijacked accounts of the victim to his friends. Such message could contain an invitation to see a video and a link. Hijacking social trust, the victim was unaware that he is redirected to a phishing page with the name YuoTube, instead of YouTube. User is prompted to download Adobe Flash Player in order to see the video. Running the downloaded file infects the victim's computer.

Downloaded file didn't contain the actual Koobface malware. The program scanned the computer to identify which social networks the victim uses and contacted C&C to download appropriate components. Koobface consisted of these parts:

- Koobface downloader
- Social network propagation components
- Web server component
- Ads pusher and rogue antivirus
- Captcha breaker
- Data stealer
- Web search hijacker
- Rogue Domain Name System changer

Components serve various purposes. Social network propagation component is responsible for spreading the infection. It steals victim's account by inspecting saved cookies of a web browser and starts sending messages to victim's friends with links to malware loader. This component can also harvest sensitive information about the victim. These data are scraped from social network profiles. Loader even upload photos of the victim, so that it could be used for blackmailing.

Other components for instance change how the victim browses the Internet. Koobface could plant bogus results of searching by serving sites with adds or

resolving addresses to point to forged websites. Data stealer component contained keylogger sending captured credentials to C&C.

Interesting component is the Captcha breaker. When Koobface encounters a captcha, it does not solve it by using some advanced recognition algorithms, but instead the captcha is presented to the real user with a panic notification and a timer mimicking a message made by operating system. Input is then send to C&C.

C&C channel

Architecture of the C&C is quite basic. Koobface started with a centralized network consisting only infected hosts and C&C domains[9]. After that, the architecture was updated by adding a layer of infected zombies used as proxies between victims and C&C. There was only one working C&C server at a time.

Koobface uses HTTP as a protocol for communication. Bot issues GET and POST requests to servers, which are running a PHP server. C&C responds with a plain text containing commands and other information. A Bot possess hard-coded list of domains where C&C should reside. It first check availability by requesting a file `/achcheck.php` to which server responses with string `ACH_OK`. If the domain is reachable, bot sends a GET request to file `gen.php` and includes information about social networks used by the victim as a part of the request. Then it downloads components selected by C&C.

C&C issues commands also in plain text. Sample orders included text `PERMANENTLIST`, `STARTONCE` or for instance `SHARELINK`. Bot, sending stolen data by POST, encrypted information using bitwise-ADD operation. Encryption key was part of the malware.

Koobface was taken down due to the effort of Facebook researchers. Attackers evaded shut down by reading articles about Koobface research and their ability to update botnet.

3.2 Gameover ZeuS

ZeuS is the most popular DIY botnet kit distributed by cybercriminals on underground markets. Botnet was first sighted in 2006. ZeuS is a banking trojan stealing account numbers and credit card credentials. We will focus on latest version of ZeuS called Gameover ZeuS (GOZ)[10]. Original ZeuS botnet has centralized C&C. Its successor GOZ introduces private peer-to-peer architecture.

Controllers of GOZ spread malware by renting a large spamming botnet called Cutwail. Send emails impersonated known companies, financial institutions or networking sites and lured victims to click on bogus links. For instance by warning about an unpaid bill. Link directs the user to a fake loading screen where a Blackhole exploit kit is downloaded by JavaScript in the background. Blackhole exploits known security bugs in PDF files, Java applets and Flash plugins. Vulnerability allows inserting a shell code, which downloads a malware loader. The loader called Pony then tries to retrieve a proper binary from hard-coded

compromised web servers. Pony also scans victims system to steal credentials for transforming victim's computer to a FTP server, which could further seed malware binaries. Result of the scan is send using encrypted POST through a proxy to C&C.

P2P architecture

After launch of the downloaded malware, the bot uses hard-coded list of domains to contact the C&C. If successful, C&C sends additional binaries, updates and an actual list of peers. If no domain is reachable, bot uses a domain generation algorithm, which is seeded by current date. The algorithm produces 1,000 pseudo random domains per day. The purpose of these is to provide a list of peers in the network.

Communication is carried over TCP or UDP on random ports between 10,000 to 30,000[11]. Configuration files, binaries with malware and some control messages are signed by botmasters to avoid poisoning and interfering with the botnet by disruptors.

Topology of GOZ is divided into three layers. These are C&C Layer, C&C Proxy Layer and P2P Layer. Workers from P2P Layer does not contact directly C&C Proxy Layer. Instead, botmasters select some bots as a proxy to communicate with upper layer.

Each bot stores a list of neighbor peers and proxy bots. Proxy bots are announced by botmaster. Bots manage their lists cleverly. Every time another bot contacts them, they add it to the list, if it contains less than 50 peers. Bots test availability of peers every 30 minutes, giving them a few attempts to respond. When lacking peers, they contact the C&C and pull a fresh list. If the bot is completely isolated, it tries to bootstrap again from the list of hard-coded addresses, but if it also fails, bot switches to the domain generation algorithm. When finding reachable domain, it fetches a fresh list of peers, which is digitally signed to prevent poisoning. Botmasters implemented various strategies to prevent disruption. For instance bots manage blacklist of peers which contacts them too often. They also won't add a new peer, if it belongs to the same subnet. This makes it harder for disruptors to analyze the botnet and isolate bots by introducing fake bots into the network.

Message structure

Bots usually communicate over UDP, but messages including binary updates or configuration files are send over TCP. Messages have custom form. Only communication between proxy bots and C&C Proxy Layer seems to be wrapped into HTTP. Structure of the message is divided into parts as follows:

Random First byte of message is a random value. It's an error detecting code used to verify successful encryption of message. Some bots use XOR encryption algorithm leaving first byte unencrypted. This byte has to match first byte of Session ID flag described below.

Time to live Some messages carry number, which is lowered each time a bot forwards it to other peers. When this number reaches zero, message is not propagated. This prevents infinite message forwarding through the network.

Length of padding Determines how many bytes are added to the end of the payload. This is a defense mechanism avoiding signature detection. Messages therefore have random length.

Type Flag determining purpose of the message. Bots uses several types of messages including data request, peer list or version request and their corresponding responses.

Session ID Bot sending a request includes a random session number with a length of 20 bytes. The response has to reply with the same number or the message is discarded. This prevents some attacks on the botnet trying to spoof messages.

Source ID Each bot has unique 20 byte length identifier which is calculated upon infection of victim using a hash algorithm. Bots are able to update their list of peers and proxies when they are contacted from the same bot from a different address.

Payload Payload itself conforms the type of the message. Each type has its own structure. There are inserted padding bytes after payload.

Gameover ZeuS botnet was successfully taken down leaving losses worth millions of dollars [12]. FBI is offering a reward of up to $3 million for information leading to arrest one of the criminal involved in GOZ activity.

3.3 Stegobot

Stegobot is a proposed idea of a botnet with a covert C&C channel described in a research paper [13]. The goal was to create a botnet with a probabilistically unobservable C&C channel. Botnets transformed from centralized architecture to decentralized and according to the researchers, the next step of evolution will be becoming unobservable. For the sake of research, they made botnet that uses image steganography to hide presence of the C&C channel.

C&C channel

Characteristic of the C&C channel is exploitation of widely used social networks that allow sharing multimedia content. Using such service brings a benefit of high resistivity to take down. Social networks form free-scale graphs, which are highly robust to removal of random nodes.

Communication of a bot is quite unique, because a bot does not actively request orders and responds to them. Communication is carried by the victim. Every time he uploads an image, bot adds a payload to it. Similarly, every time the service shows the user a picture uploaded by his friend, the bot checks it for included payload.

Selected social network acts as a peer-to-peer overlay over which data are send. Messages travel along social links of the victim. Bot can receive either a command from botmaster or a response from another bot. In both cases, it resend the

message to others using restricted flooding algorithm. Each message carry information about how many times it was forwarded. Once it reaches the maximum number of hops, bot does not spread it anymore. The number of hops must be chosen appropriately to the size of the botnet. High number of hops increases the chance that botmaster receives the response, but also reduces channel bandwidth, because of high amount of duplicate messages transferred.

For hiding information into images, algorithm called YASS [14] was selected. Unlike common used algorithms for steganography, YASS is more resistant to detection. It embeds data randomly into a JPEG picture while preserving statistical behavior of the image. In can also withstand a second compression carried by the service to which the picture is uploaded. However, the undetectability is compensated by the low embedding rate of bits in the image. Also YASS insert redundant bits for errors made by compression. With balanced robustness and steganographic capacity an image can hold on average 10,000 bits.

Effectiveness

Testing showed that network consisting of 7200 nodes was capable of reaching up to 86,13MB of bandwidth per month. 18000 different messages were delivered to a botmaster. That was achieved with low redundancy of bits in images. Higher redundancy creates more robust channel, but bandwidth drops significantly. Testing also showed that botmaster received about 10% of stolen information send from the bots. In conclusion, it is possible to create botnet with unobservable C&C channel with purpose to steal sensitive data.

4 Framework implementation

This chapter will introduce our implementation of a framework for testing C&C protocols. Our goal was not to create a functional botnet which capabilities that would be comparable with a real botnets. We wanted to test, whether some services on the Internet are vulnerable to botnet infection and suitable for botnet operations.

4.1 C&C channel

The target services we wanted to test were massively used social networks or commonly used Internet services. We set a requirement, that the service itself should play a role of C&C server. There are botnets, which abuse social networks like Facebook[1] or Twitter[2], but they use them as a medium to spread the infection. We wanted to delegate the most parts of C&C to the service. This approach has a certain advantage for the controller of the botnet. Because of the massive usage of these services, there applies the principle of hiding in plain sight. That means that the botnet does not have to care so much about hiding its communication. The communication might get lost in regular traffic produced by a human user. Since such botnet would use HTTP protocol and regular ports, it is not possible to disallow these port or even block access to the services. In a network of computers we might forbid ports used by IRC channels or blacklist servers known to be used as C&C nodes. Blocking port 443 used for HTTPS or blocking access to certain services is not viable option because it strongly limits the user.

Establishing a C&C channel between a bot and a service has another characteristic. Unlike IRC traffic, which isn't encrypted by default, services usually use HTTPS. The communication is encrypted and one cannot simply eavesdrop it. Capturing the traffic gives us distinctively less information than capturing a regular HTTP. We will know destination server and size of the request, but we won't know the whole request. Our proposed protocol is considering centralized topology.

4.2 Design of framework

The CCCBot framework is implemented using Java language. It's meant to be extendable and during development there was emphasis on an ability to add a new functionality by a user of that framework. There are two main components.

[1] `https://www.facebook.com/`
[2] `https://www.twitter.com/`

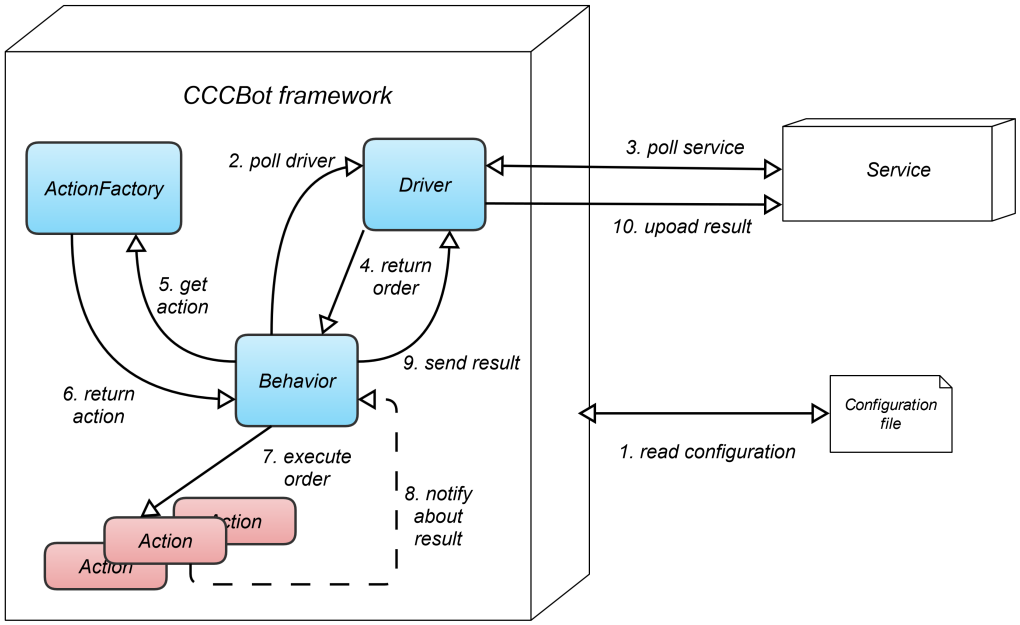


Figure 2 General flow of the framework

Former is driver enabling communication with a service. It is represented by java interface `Driver`. Driver is capable of retrieving orders from a C&C center and send responses. The latter is behavior of the bot. Behavior controls frequency of polling the service, schedules execution of orders and sends reports of their completion through driver back to the service. In framework is behavior represented by `Behavior` interface.

4.2.1 Flow of the framework

Diagram Fig. 2 explains the basic flow of the framework.

1. Initial configuration of bot is read from a configuration file `config.txt`. This file contains information about desired behavior of the bot and the service to which it will be connecting. There might be additional settings specific to the service, for instance accessing credentials.
2. When behavior decides, it polls the driver for an order.
3. `Driver` connects to the service and attempt to withdraw an order, if it's present.
4. `Driver` parses order with its parameters using a set protocol and passes it to the `Behavior`.
5. Behavior prompts an `ActionFactory` to get instance of an `Action`
6. Factory creates an action. This represents a work necessary to complete the order. The action is passed back to the behavior.
7. Behavior creates a new thread for the action and starts execution of it in that thread. This execution runs concurrently to the run of communication with service.

8. Upon completion of execution of the order, the action notifies about it the behavior.
9. Behavior receives the result of execution and issues a request to the driver to send a report back to C&C.
10. Driver translates the result according to the protocol and uploads it to the service. Flow then repeats from the point 2.

4.2.2 List of orders

CCCBot has a fixed set of orders. These orders are instructions which can be issued by the botmaster. They differ from their input and result. Selection of orders is not random. We chose following set of orders after studying which actions botnets usually conduct. An order can be commanded with or without parameters. The result of an order can be a simple string informing about outcome of execution or a file. The framework lists all kinds of orders in enum class `OrderType`.

BOTNAME Every bot has an identification name assigned to it. The name is saved in the configuration file. It servers to distinguish recipients of commands by the botmaster. Order of type BOTNAME simply retrieves the name of the bot.

SYSINFO This command issues a bot to return some information about the computer, where the bot resides.

NETINFO Bot collects data about network setup of the victim.

DOWNLOAD This order commands the bot to download a file. The path of the file is a parameter of the command. With this command, botmaster is able to push an additional malware to the infected computer.

UPLOAD Bot searches for a file located on the computer and sends it to the botmaster. The path to the file is passed as a parameter of the order.

SCREENSHOT Bot captures a screenshot of the current screen on victim's computer and uploads it.

SHUTDOWN Deactivates the bot. The process of the bot will exit.

EXECUTE The most potent command of all. The parameter of the order is a command line command. The botmaster should know the operating system on host prior issuing this order. With this command, the botmaster is able to fully control the victim's computer. Result of this order is string printed to the standard output. Since each activity in the framework is run in separate thread, it is possible to launch a process, which does not terminate immediately. Let's say a DoS attack. The bot will continue to listen to new orders.

4.2.3 Configuration file

The initial setup of the bot is specified in the configuration file. The file called `config.cfg` contains pairs of keys and values following scheme `key:value`. These settings tune the behavior of the bot. Configuration file is loaded on the start. Some settings could look like this example:

```
driver:SpreadsheetDriver
behavior:RandomBehaviour
random_reaction:20000
```

The configuration file is useful, when a driver or a behavior needs an additional information for its run. This setting is then prefixed with a word to match it to the concrete module. In the example above, the key `random_reaction` modifies a waiting time of a chosen behavior. There is a fixed set of keys which are mandatory to run the framework. These are:

driver name of the driver in form of the case sensitive name of Java class.

behavior name of the behavior according to which the framework will act. It's also case sensitive name of behavior class.

botname this setting assigns a name to the bot. Botmaster can then command orders to a single bot or a group which shares the same name.

4.3 Behaviors

There are two behaviors implemented in the framework. Former is called `FixedTimeBehavior`. It simulates a "dumb" bot. The behavior polls service in precise set periods of time. After fetching a command, the behavior immediately executes it and responses as fast as possible. This makes actions of the bot very obvious.

The latter is called `RandomBehavior`. As the name suggests, the behavior polls the service randomly to fetch new orders. The response does not immediately follow the request. Instead, the response after execution of the command is random and as well and the time span, in which response could occur, is greater than polling frequency. By this approach, the response could be send out of a polling window, in which the command was received. By doing so, we are trying to break otherwise synchronous communication. As stated in paper about botnet detection[15], the researches found it difficult to notice a botnet communication which was asynchronous. In their setup, the botnet used plain HTTP protocol. It was obvious, what is a request and what is an activity report. In our case, where we use HTTPS, it may be not so easily distinguishable but it could be guessed.

4.4 Drivers

This section will describe how we implemented each communication protocol for tested services. We will also study characteristics of services and possibilities of their misuse.

4.4.1 OAuth

Before further reading, we want the reader to get familiar with a concept of OAuth. OAuth is a delegation protocol [16] which is often a part of authorization protocols on large web services. OAuth solves delegation of access to some resources when the user is not present. There are currently two versions of OAuth.

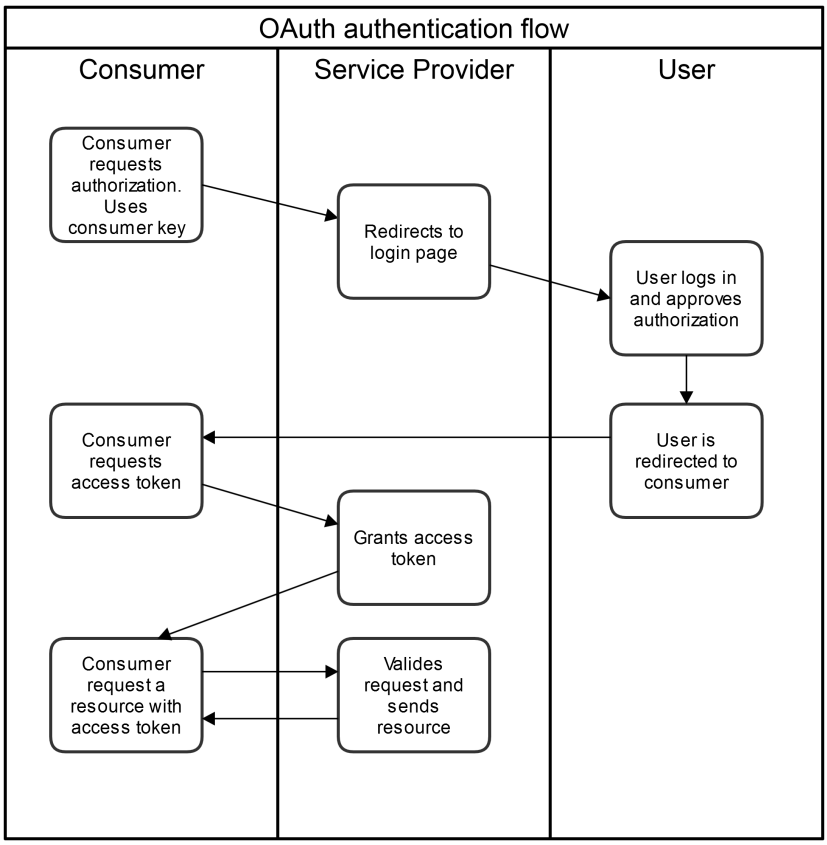


Figure 3 OAuth authentication flow

Each specification describes how to proceed in different situations. The point of this explanation is that in each protocol a real user plays a role and we want to overcome his presence during authorization. We will describe how we coped with this problem in following sections about implementations of protocols.

OAuth advocates usage of access tokens and client secrets. The main idea is that there are pairs of values shared between client and service. Only one value is send over network at a time to identify a user and the second one called secret is used to sign that request. By eavesdropping the communication, we get the token but without knowing the secret, we can't create legitimate request to a service.

Let's look at a slightly simplified flow of OAuth authorization[17] in Fig. 3. The flow differs from version 1.0 and 2.0 but the main idea stays the same. The real example might look like as follows. An application wants to issue some calls on an API of a service. It presents itself with a consumer key, which identifies the application and its permissions. The service sends back an URL to authorize an user. The application redirects to this URL in a web browser. The real user of the application logs in and accept that the application will perform actions on his behalf. After logging in, a numeric code is showed to the user. The user inputs it into the application. With presenting this code to the service, the application is authorized and receives the access token. The token is then used in every request to the service.

4.5 Dropbox

Dropbox[3] is a free service allowing to store large amount of files on a cloud. More than fifty million people use Dropbox to share files with each other. Dropbox targets all kinds of platforms ranging from computers to mobile phones [18]. They encourage programmers to create applications using Dropbox as a data store. Therefore they offer an API for working with the files saved in Dropbox.

The fact, that there is an opportunity to create applications based on Dropbox is good, because then we are not so concerned about mimicking behavior of the real user of service.

4.5.1 Proposed channel

The idea of communication channel is quite obvious. Botmaster creates a file containing orders. This file is shared among every bot. Botmaster continuously updates the file with new orders. Bots keep synchronizing this file. For every command, which requires an answer, they create a file containing response. This file is uploaded by Dropbox and shared with the botmaster.

Order file should stick to the following structure. Every command is a string on a separate line. The string matches scheme `number:name:order:parameters`

- number helps the bot to keep track of the last command. The file could change unpredictably. Every time the bot fetches a new order from the file, it will look for the first order with a number greater than the last one executed.
- name part determines, if the order is commanded to the bot. Bot will ignore a command, if the name wouldn't match bot's name, string `all` or empty string.
- `order` part is the name of issued order.
- `parameters` is a string containing additional information necessary for execution of the command. The ;ast part including the semicolon can be omitted, if the order doesn't need parameters.

4.5.2 Limitations

The API of Dropbox offers a whole range of actions including synchronizing shared files or creating and updating new ones. According to developer guides, the API calls are not strictly limited and regular applications should not exceed these limits[19]. But code documentation suggests to synchronize files no often than every five minutes. This could prolong communication between the bot and the botmaster. On the other hand, the communication should not be so frequent in order not to draw so much attention.

Prior making some API calls, each application should authenticate itself using Dropbox implementation of OAuth 2.0. This requires a user interaction. The user should be redirected to a login page, sign in and allow access to his files as

[3] `https://www.dropbox.com/`

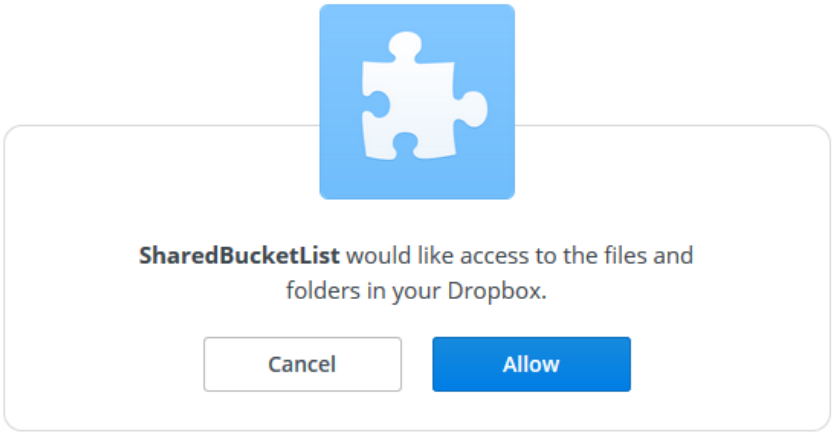


Figure 4 Authorization of access by the user

showed on Fig. 4. User receives a code, which should input into the application. One way to overcome this issue is to set up an auxiliary server collecting URLs for the authentication. Botmaster would open these pages, signs in and send back the received code to the bot to complete the authentication. This approach has two major drawbacks. First, the botmaster would log in from a different and possibly distant network. Bot making API calls from different place would definitely be suspicious in the eyes of Dropbox. The latter problem is not huge, but may matter. By creation of auxiliary server we are breaking the idea that C&C is part of the service.

There is a Java library called HtmlUnit[4] that simulates web browser without a GUI. We wanted to simulate a behavior of a real user logging into Dropbox and authenticating the application using that library. But when we inspected the logging form, we found that there are several Javascript functions attached on various input events. Their names and bodies were obfuscated but one of them was named `monkey_check()` . Monkey tests in computer science are automated test used for testing software without the user. We can guess, that the function `monkey_check()` wants to detect a software impersonating a human. Therefore HtmlUnit was not an option in this case.

Implementation of the channel utilizes the ability to create an access token prior authentication. This token should serve only for testing purposes to speed up the development process of an application. This brings a disadvantage that only one account can use this token. Therefore, we need a distinct account for every bot.

The file with orders could be labeled as public, making it accessible without using the Dropbox API. Unfortunately, public files have a limit of accesses in a time period. Larger botnet could reach this limit quickly.

Botmaster may not have an option to use hijacked accounts in his botnet. Changes made in files are showed to the real user by Dropbox application installed

[4] `http://htmlunit.sourceforge.net/`

on victim's system. The user could very quickly spot malicious activity on his computer.

4.5.3 Setup

This section will give reader instructions how to use the Dropbox channel in the framework.

1. User should register an app on developer site of Dropbox[5] and give it full access to Dropbox account.
2. The app should allow setting called *Allow implicit grant*.
3. User has to generate an access token and create a directory in the root folder of Dropbox having the same name as the bot
4. User has to include all needed information in configuration file

The list of settings necessary in configuration file is following:

driver The pair should be `driver:DropboxDriver`

botname Unique name of bot. The bot will save responses to directory with the same name

dropbox_appkey The value is string labeled as App key on developer site

dropbox_appsecret The value App secret completes the pair with App key

dropbox_apptoken This marks the generated access token

dropbox_sharedfile This is path to the file with orders. The path has to start with slash and should not use backslashes for delimiting directories.

4.5.4 Summary

Identifying one bot in the network could be dangerous for the whole network, since each bot shares a file with others. This bot could uncover the whole network of bots leading to possible shutdown of all of them.

In case of low number of bots in the botnet, we believe Dropbox could be misused. The channel could last for days. Unfortunately, the channel suffers from the single point of failure. If Dropbox revokes the access token, forcing client to authenticate again, the bot has no way to recover.

On the other hand, in case the C&C account is banned and the rest of accounts are left untouched, the botnet can be rebuild by botmaster simply by creating a new shared order file with the same name.

4.6 Pastebin

Pastebin is one of the biggest paste tools on the Internet. This service lets a user to store a text for sharing it with others. It was created mainly for programmers who needed to share snippets of code with their collaborators.

Pastebin exposes a quite simple API for working with uploaded texts called pastes. The only requirement for using it is creating an account and obtaining unique developer API key.

[5] `https://www.dropbox.com/developers/apps`

4.6.1 Proposed channel

Each paste created on Pastebin has its own unique identifier. Pastes can be made public or private. The latter is only visible to its creator. Botmaster creates a private file, which holds orders for the botnet. Each bot will know identifier of this file and will regularly check it for new orders. After completing the command, the bot will create a paste containing the response. In case of a non-text file, the content will be encoded by Base64 encoding scheme. If the file is bigger than 512 kilobytes, it will be split. Because the order file is private, each bot has to be logged using the same username and password as the botmaster.

The form of the order is the same as proposed in the section about Dropbox channel. A command contains a number, a name, an order and parameters separated by semicolon. Each command is on a separate line in the order file.

4.6.2 Limitations

It turns out that Pastebin is quite strict and the implemented channel is very fragile. Pastebin has various strategies how to fight with spammers. Therefore a registered user is only allowed to create up to 20 new pastes per 24 hours. Using a web interface, botmaster can edit the paste containing orders. Unfortunately, this action is not possible using the Pastebin API and responses has to be placed in new pastes.

Pastebin prohibits certain content in files. One should not paste sensitive data like emails, passwords or links. Newly created pastes are inspected for particular keywords that Pastebin consider suspicious. Therefore we implemented a dictionary for name of orders. For instance, instead of the word *execute*, the botmaster should use *do* or instead *shutdown* there is equivalent word *sleep*. When the service assumes that it's abused, it responses with captcha prompt. If the client does not respond in 10 minutes, the paste is not created.

4.6.3 Setup

To use Pastebin as a communication channel in CCCBot framework, user should follow these steps

- Create an account on Pastebin and get Unique Developer API key[6]
- Create a private paste for orders
- Set properly configuration file

Driver requires these properties be found in configuration file

driver Set as `driver:PastebinDriver`
botname The identification of the bot
pastebin_username Username of botmaster
pastebin_password Botmaster's password
pastebin_apikey Unique Developer API key

[6] `http://pastebin.com/api`

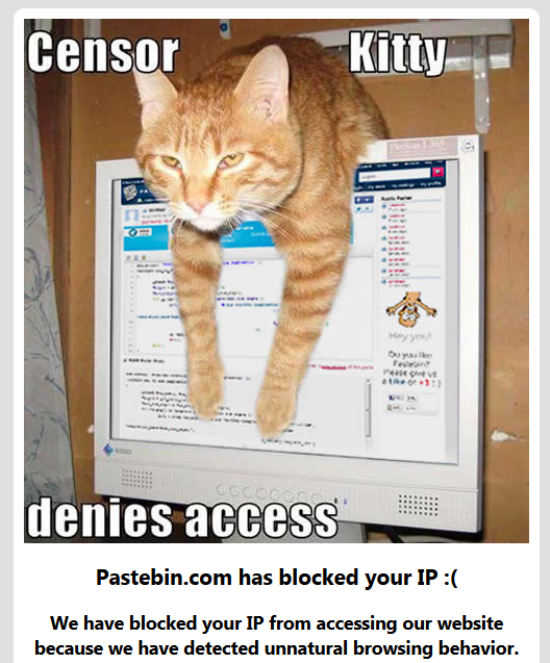


Figure 5 Pastebin informing about blocking an account

pastebin_orderfile Unique string identifying paste with orders. The string can be extracted from link to that paste. For example link `http://pastebin.com/4JCuDUxv` directs to paste with id `|4JCuDUxv`

4.6.4 Summary

Pastebin does not offer a good way how to establish a robust C&C channel. The network of bots could be very small because every bot shares the same account and there is a strict limit on number of created pastes. Also the size of pastes is limited.

We observed that Pastebin starts to test client with a captcha very quickly under certain conditions. We polled the order file in fixed time intervals. The orders were edited by the botmaster several times in a short period of time. Pastebin requested the bot with captcha several times. Since our framework cannot deal with captchas, it ignored them. The account was immediately blocked. However Pastebin showed some sense of humor in the way how he informed us about it. The screen on Fig. 5 depicts how.

4.7 Twitter

Twitter is huge, well known social network[7] with hundreds of millions of users around the world. The idea of using Twitter as a part of a botnet in some way is not new [20]. Let's look at how we managed to establish C&C channel using Twitter.

[7] `https://twitter.com/`

4.7.1 Proposed channel

The botmaster controls a Twitter account. Every order is composed into a Tweet. Bots follow the account of the botmaster and read his Tweets. They extract command from the Tweet and sends their response using direct message mechanism to a second botmaster account.

The structure of a command is very similar to the previous cases. The command consists of a number, the bot name, the order and parameters. There is a single difference. The whole command is surrounded with double quotes. The rest of the Tweet could be some random text mimicking real Tweets.

Because each Tweet and direct message is limited by 140 characters, there has to be an auxiliary server catching uploaded files.

On start of the framework, the bot reads the latest Tweet of the followed account. Then it will read every new Tweet created afterward.

4.7.2 Limitations

The first obvious limitation is restricted length of messages. We are not able to compose the whole botnet communication into Twitter. Botnet has to communicate with a side server to upload larger files.

We introduced a random text into commands for hiding the real communication. This is necessary since Twitter does not allow repeating Tweets. The unimportant content in the Tweet let us command the same order multiple times. We chose to respond using direct messages, because then uniqueness of messages does not matter. Since private messages are unlike the Tweets private, we are not exposing the whole communication to the public. Bot has to follow the account on Twitter to which he will send responses.

Twitter, similarly as Dropbox, wants the user be authenticated using their implementation of OAuth. We can avoid direct login of the user through a web browser by generating an access token on developer sites of Twitter.

Only a limited number of bots can share the same account so botmaster should acquire a sufficient amount of them for building the network. Twitter allows regular accounts to use its API to read various data. But main problem arise when an user wants to create a data like Tweets or messages. In that case, Twitter permits these actions only to accounts that were verified by a telephone number. This may be the greatest obstruction in creation of a botnet solely communicating through Twitter.

4.7.3 Setup

Using Twitter driver in the framework is similar to other drivers but there are some additional steps.

- Create an account on Twitter, validate account using email and mobile phone.
- Create an app[8] with access level read, write and direct messages

[8] `https://apps.twitter.com/`

- Get Consumer Key, Consumer Secret, Access Token and Access Secret
- Set properly configuration file

Driver requires these properties be found in configuration file. There are quite a few of them.

driver Set as `driver:TwitterDriver`
botname The identification of the bot
twitter_consumerkey Consumer Key
twitter_consumersecret Consumer Secret
twitter_accesstoken Access Token
twitter_accesstokensecret Access Token Secret
twitter_source Identification number of an account which will post orders
twitter_recipient Identification number of an account to which the bot will send responses
twitter_uploadsite IP adress of side server collecting files
twitter_uploadport port number on which the server will listen

4.7.4 Summary

The channel cannot rely only on Twitter. There has to be a secondary channel for uploading files. But the main problem is verification of an account by the telephone number. This may limit the botmaster from creating a huge network of bots. Otherwise the channel is usable. Of course, it's not prefect because like every channel proposed so far it suffers from the single point of failure. Ban of the account used for C&C or response certainly inhibits the whole botnet.

4.8 Google Spreadsheet

There are many services associated with Google. One of them is a cloud file storage Google Drive similar to Dropbox. Within the Google Drive, user can manage various office documents. We were inspired by work that proposed proof of concept of a communication channel using spreadsheets [2]. We recreated this concept of channel only with little changes.

4.8.1 Channel design

Botmaster set up a spreadsheet into which he will insert commands. Bots read this file and for every included command they submit a preset Google Form. They enclose their response and name in the questionnaire.

Google offers various APIs for working with its services. In recent versions of API, Google tends to use OAuth for authentication of the user. Luckily, Google supports an older version of API where the content of the spreadsheet can be withdrawn in JSON[9] format. When the file is made public by enabling setting called *Publish on the web*, there is no required authentication to access the data.

[9] `http://www.json.org/`

The response form automatically inserts data into a second spreadsheet. Naturally, Google does not offer an API for programmable submission of the forms. These actions should be made by a real user. But since the questionnaire is a quite simple HTML form, we used HtmlUnit library this time. As mentioned in in section about Dropbox, HtmlUnit simulates a regular web browser. The form does not observe typing patterns of the user. There are only some scripts managing access statistics. We could just send plain POST request but because the automatic submission is not supported, we decided to attempt to imitate a real user's behavior.

4.8.2 Limitations

The cells in spreadsheets can hold longer texts but they are not suited for files. This C&C channel cannot do without an auxiliary server collecting files.

When submitting the form, Google may become suspect that client is a bot. In that case, the client is presented with a captcha. We try to avoid it by simulating real behavior.

4.8.3 Setup

CCCBot framework requires doing these steps prior using this C&C channel.

- Create an empty spreadsheet. First line of cells should contain these values: number, name, what, how. These values denote content of cells below. The protocol is very similar to previous ones. *number* is for identification of order, Value *name* targets group of bots. *what* describes name of the order and *how* is optional cell for parameters of the order.
- User should enable *publish on the web* setting. By doing so a link for sharing is created.
- User has to create a form consisting one short text input for bots name and one long input for response.
- Set properly configuration file

List of parameters in configuration file are

driver Set as `driver:SpreadsheetDriver`

botname The identification of the bot

spreadsheet_order Identification string of the spreadsheet with orders. The string can be obtained from generated link. `https://docs.google.com/spreadsheets/d/XXXXXXXXX/`. The letters X mark the place of needed id.

spreadsheet_form The whole link to the Google Form for collecting responses.

spreadsheet_uploadsite IP adress of side server collecting files.

spreadsheet_uploadport port number on which the server will listen.

4.8.4 Summary

Building a C&C channel using Google spreadsheets is quite clever and suitable because of no need to authenticate the client. The botmaster does not have to

care about creating bulk of Google accounts. If the bot had to use an account, it could be banned leading to decomposition of botnet. This makes the channel more robust. But the channel is not perfect. We have to use side server for files again and the is a weak point which potentially breaks the botnet. In case of deleting spreadsheet with orders by Google, botnet cannot recover.

4.9 Hybrid channel

So far we used only one service for our channel. This section will discuss a design of a C&C channel utilizing more than one service at a time in order to establish a scalable and robust channel.

4.9.1 Channel design

The channel will use a different service for sending orders to bots and responding with results.

Commanding orders

For commanding, we utilized Twitter Streaming API. Twitter offers real time streams of Tweet data.A client does not have to poll the service. Instead, Twitter notifies the client every time a Tweet is created. The client can set up a filter reducing flood of Tweets.

Using the API does not require an account verified by telephone number as we have seen in the section about Twitter. API is only used for reading the data. There is however some level of authentication required. Client should register an app and let Twitter to generate access tokens. Bot then listens of a stream of Tweets filtered by a set of words sets by the botmaster. Twitter will only notify user when a Tweet containing all of these words is created. Additionally, bot knows a magic word which has to be present in order Tweet. This separates randomly created Tweets by people around the world from the ones containing an order.

The main advantage of this approach is that commanding account is decoupled from the network of bots. Botnet does not asks for Tweets from a single account and shutting down the C&C account does not destroy the whole botnet. Orders can be issued from a new one as long as the Tweet contains the set keywords and the magic word. The bot itself should not be banned for abusing or spamming the service since it uses the API as it was intended. Choosing the right filter, we can achieve desired frequency of incoming Tweets in order not to create suspicious traffic in the network. We might also benefit from the fact that creation of the communication is random and may follow some patterns regarding to the global activity of Twitter users during the day.

The structure of the order is same as with channel solely using Twitter.

Responding to C&C

For responding channel, we chose Google Forms. Sending data to the botmaster through forms is a viable option because no authentication is required. Since the orders are issued through Twitter, the bot does not have to poll a spreadsheet file and possibly create a suspicion from abuse. This could make the channel more covert. However, this channel still suffers from single point of failure. In case the Google decides to deny access to the form, the botnet may still execute orders but can't communicate back to the C&C.

In paper [21] the authors proposed an idea of using a URL flux in botnet communication. There are countless of services providing shortening of an URL. These services found their application for example on Twitter, where messages are limited by 140 characters. The shortening service takes an URL and generates a short link, which may look like this: `http://bit.ly/1Ip8Msd`. This link redirects the user to the original destination. What we want to use is the fact that a shortening service TinyURL[10] offers custom name of the link for free. Other similar services may be also free, but generated links are random.

We provided the bot with a simple algorithm generating a link where it would expect the form for submission of the results.
Obtained link may look like this: `http://tinyurl.com/keyword14311`. It consists of a keyword selected in advance by the botmaster and a UNIX timestamp divided by 100,000. This generates an unique link, that is valid for roughly one day. The botmaster is obligated to register these shortened links pointing to valid Google Forms.

The keyword could be usable to balancing traffic leading to a single file. Retrieving responses from various files distributed on separate accounts makes some overhead but we can use this principle to enlarge the botnet.

4.9.2 Limitations

Despite using two of our choice services, we still need an auxiliary server for uploading larger binary files. Choosing right filter for incoming Tweets we won't reach rate limits.

4.9.3 Setup

The setup of framework shares configuration of `SpreadsheetDriver` and `TwitterDriver`

- Create public spreadsheet with same layout as for `SpreadsheetDriver`
- Create a Twitter app to get pairs of consumer keys and access tokens
- Setup a Google Form for responses and generate shortened link following generation algorithm
- Provide configuration file with right properties

List of parameters in configuration file are:

driver Set as `driver:CombinedDriver`

[10] `http://tinyurl.com/`

botname The identification of the bot
combined_order Identification string of the spreadsheet with orders. The string can be obtained from generated link.
Example link: `https://docs.google.com/spreadsheets/d/XXXXXXXXX/`. The letters X mark the place of needed id.
combined_form The whole link to the Google Form for collecting responses.
combined_consumerkey Consumer Key
combined_consumersecret Consumer Secret
twitter_token Access Token
twitter_secret Access Token Secret
twitter_filter Set of comma separated words serving as filter for incoming Tweets.
twitter_magicword Keyword included in Tweet containing the order.
combined_shortenedurl Keyword used in generation of shortened link to the Google Form
combined_uploadsite IP address of side server collecting files
combined_uploadport port number on which the server will listen

4.9.4 Summary

By combining two services, we obtained a quite robust channel. By bringing URL flux and stream of Tweets into account, we ensured that botnet can recover a from failure. By splitting of getting orders and responding to two services, we prevented or lowered possibility of denial of access to the service because of suspicion from abuse. The only downside of the proposed channel is that we didn't manage to perform the whole botnet's C&C communication using solely a web service. We still need a secondary server collecting uploaded files.

4.10 Comparison of channels

In this section, we analyze the observed advantages and disadvantages of implemented C&C channels. The key properties will be following.
Scalable The ability of channel to withstand large size of botnet and managage the communication.
Service only The whole C&C communication resides in an Internet service.
Recoverable The botnet should be able to recover from a failure.
Difficult to setup Our evaluation of difficulty of arranging the channel.

Table 1 Comparison of channel properties

	Scalable	Service only	Recoverable	Difficult to setup
Dropbox	yes	yes	no	yes
Pastebin	no	yes	no	no
Twitter	yes	no	no	yes
Google Drive	yes	no	no	no
Hybrid	yes	no	yes	no

From all implemented C&C channels, the hybrid channel stand out from others because of his good ability to recover from a failure. The communication is split, so it is more difficult for the service or the observer watching traffic to spot malicious communication.

Only drawback is the necessity of a secondary server. Therefore the next choice of an usable channel would be utilizing Dropbox.

4.11 Drop off server

Some implementations of C&C channels could not work without an auxiliary server collecting files. For the purpose of testing, we created a simple HTTP server. We found it necessary in case of channels using Twitter, Google spreadsheets and combination of these. The server only processes multipart POST requests. The request has to contain only the file. The destination of the request denote suggested path of the file.

4.12 Extension of framework

The CCCBot framework is meant to be extended. New behaviors of bot and communication protocols can be added. This is done by implementing appropriate interfaces. Classes with implementation should be then compiled and placed into designated folder. When a class is selected in configuration file, framework will load it on demand.

For creating new behavior of the bot, interface with full name `cz.cvut.fel.timrmare.behavior.Behavior` has to be implemented. This interface consists of following methods.

```
public interface Behavior {

    public void run() throws Exception;

    public void shutdown();

    public void initialize(Driver driver,
                           ActionFactory actionFactory)
                               throws InitializationException;

    public ExecutionCompletionListener getListener();
}
```

Method `initialize` is called automatically by framework. Passed driver serves for communication with a service and `ActionFactory` is competent to serve implementations of actions depending on operating system where the bot runs.

In case of new driver, interface `cz.cvut.fel.timrmare.driver.Driver` has to be implemented.

```
public interface Driver {

    public Order poll() throws IOException;

    public void send(Report report) throws IOException;

    public void sendFile(FileReport report)throws IOException;

    public void initialize() throws InitializationException;

}
```

Functionality of methods is straightforward and as with `Behavior`, method `initialize` is called by framework.

5 Conclusion

In our work, we focused on Command and Control (C&C) channels of botnets. We summarized possible criminal activities associated with them. We studied, how bots can be organized in networks. We described the architecture and the operations of three different botnets. First two were Koobface and Gameover Zeus. We described their network topologies, strategies and capabilities which made them powerful to deal damages worth of millions of dollars. Last of the three was proposal of possible design of a botnet utilizing steganography to hide its presence in a traffic. This botnet called Stegobot piggybacks on actions of real users on social networks showing that we might encounter almost undetectable botnets in near future.

We created a framework for proof of concept implementations of C&C channels. Our goal was to create protocol of communication, which is carried solely by an Internet service. We observed that it is possible to create a botnet which almost entire C&C is a part of a widely used service. Abusing a social network or other popular web service enables hiding botnet C&C communication in plain sight, because it may be very similar to real communication made by user using the same service. By doing so, presence of the botnet is less obvious for an observer of a network but we have to deal with countermeasures done by owners of service.

We successfully designed a communication protocol of a botnet with centralized C&C using combination of Twitter and Google Spreadsheets. This design minimizes authorization needed to use the services and decouples symmetric communication consisting of commands and bot's responses. After adding domain generation algorithm to protocol, we achieved quite robust C&C channel, which will recover from some failures.

This framework provides valuable source of labeled data. They may be used for machine learning software designed to scan network traffic without deep packet inspection.

Appendix A

Content of CD

This appendix summarizes the content of the enclosed compact disk. Only top level directories are listed.

Directory	Description
`Thesis`	Contains this thesis in PDF format
`CCCBot`	Source codes of the CCCBot framework
`CatchAFile`	Source codes of auxiliary drop-off server
`JavaDoc`	Generated documentation of code in form of JavaDoc

Bibliography

[1] Sérgio SC Silva et al. “Botnets: A survey”. In: *Computer Networks* 57.2 (2013), pp. 378–403.

[2] OpenSecurity. *Xenotix xBOT*. 2014. URL: `http://opensecurity.in/xenotix_xbot/` (visited on 10/14/2014).

[3] Chia Yuan Cho et al. “Inference and Analysis of Formal Models of Botnet Command and Control Protocols”. In: *Proceedings of the 17th ACM Conference on Computer and Communications Security*. Chicago, Illinois, USA: ACM, 2010, pp. 426–439. ISBN: 978-1-4503-0245-6.

[4] Jing Liu et al. “Botnet: Classification, Attacks, Detection, Tracing, and Preventive Measures”. In: *EURASIP J. Wirel. Commun. Netw.* 2009 (2009), 9:1–9:11. ISSN: 1687-1472.

[5] Won Kim et al. “On Botnets”. In: *Proceedings of the 12th International Conference on Information Integration and Web-based Applications & Services*. ACM, 2010, pp. 5–10. ISBN: 978-1-4503-0421-4.

[6] Gregory Fedynyshyn, Mooi Choo Chuah, and Gang Tan. “Detection and Classification of Different Botnet C&C Channels”. In: (2011), pp. 228–242.

[7] Jignesh Vania, Arvind Meniya, and HB Jethva. “A Review on Botnet and Detection Technique”. In: *International Journal of Computer Trends and Technology* 4.1 (2013), pp. 23–29.

[8] Jonell Baltazar, Joey Costoya, and Ryan Flores. “The real face of koobface: The largest web 2.0 botnet explained”. In: *Trend Micro Research* 5.9 (2009), p. 10.

[9] Jonell Baltazar, Joey Costoya, and Ryan Flores. *The Heart of KOOBFACE*. 2009.

[10] Brett Stone-Gross. *The Lifecycle of Peer-to-Peer (Gameover) ZeuS*. 2012.

[11] Dennis Andriesse et al. “Highly resilient peer-to-peer botnets are here: An analysis of Gameover Zeus”. In: *Malicious and Unwanted Software:" The Americas"(MALWARE), 2013 8th International Conference on*. IEEE. 2013, pp. 116–123.

[12] Federal Bureau of Investigation. *GameOver Zeus Botnet Disrupted*. 2014. URL: `http://www.fbi.gov/news/stories/2014/june/gameover-zeus-botnet-disrupted` (visited on 05/15/2015).

[13] Shishir Nagaraja et al. “Stegobot: a covert social network botnet”. In: *Information Hiding*. Springer. 2011, pp. 299–313.

[14] Kaushal Solanki, Anindya Sarkar, and BS Manjunath. "YASS: Yet another steganographic scheme that resists blind steganalysis". In: *Information Hiding*. Springer. 2007, pp. 16–31.

[15] Guofei Gu, Junjie Zhang, and Wenke Lee. "BotSniffer: Detecting botnet command and control channels in network traffic". In: (2008).

[16] *User Authentication with OAuth 2.0*. URL: `http://oauth.net/articles/authentication/` (visited on 03/02/2015).

[17] Internet Engineering Task Force. *The OAuth 2.0 Authorization Framework*. 2012. URL: `https://tools.ietf.org/html/rfc6749` (visited on 03/02/2015).

[18] Dropbox. *Online statistics fact sheet*. 2015. URL: `https://www.dropbox.com/static/docs/DropboxFactSheet.pdf` (visited on 04/20/2015).

[19] Dropbox. *Core API best practices*. 2015. URL: `https://www.dropbox.com/developers/core/bestpractices` (visited on 04/20/2015).

[20] Arbor Networks. *Twitter-based Botnet Command Channel*. Aug. 13, 2009. URL: `http://www.arbornetworks.com/asert/2009/08/twitter-based-botnet-command-channel/` (visited on 01/15/2015).

[21] Cui Shuai et al. "S-URL Flux: A Novel C&C Protocol for Mobile Botnets". In: Communications in Computer and Information Science (2013), pp. 412–419.