Protecting Web Services From Botnet Exploitations

By

Hanh Nguyen VO

A thesis submitted to Macquarie University
for the degree of
Doctor of Philosophy
Department of Computing
01 September 2012

MACQUARIE
UNIVERSITY
FACULTY OF SCIENCE

This thesis is submitted in fulfilment of the requirements of the degree of Doctor of Philosophy at Macquarie University and has not been submitted for a higher degree to any other university or institution. This thesis represents my original work and contributions. I certify that to the best of my knowledge, all sources and assistance received in the preparation of this thesis have been acknowledged.

Hanh Nguyen Vo

Acknowledgements

My sincere and deep gratitude goes to my supervisor, Professor Josef Pieprzyk, for his kind guidance and consistent support. He has guided me towards the right direction, influenced me as an active thinker, and provided me with insightful advice.

I would like to thank Dr Christophe Doche and Associate Professor Paul Watters for their help, encouragement and valuable inputs to my research.

I would particularly like to thank my family. My parents, who went overseas to assist me in difficult times, have inspired me to pursue my study. My brother and sister have never stopped supporting me in my life. Finally, there are no words to express my love and appreciation to my husband and my little baby. They make everything worthwhile.

Abstract

Botnets have attracted a significant attention and have become a primary "platform" for attacks on the Internet. As Web 2.0 evolves, especially with social networking sites becoming a dominant medium, it is no surprise that a new type of malware, Web 2.0 bot, rides on this new means of propagation and communication. For example, Koobface has been successfully leveraged social networking websites such as Facebook and Myspace to infect millions of computers and TwitterBot attempted to use Twitter as a C&C channel for communicating. Since these are legitimate sites and the communication is just like normal web traffic, this type of attack is difficult to detect by the current detection software. In addition, Captcha used for verification can be easily bypassed by modern bots as they employ the so-called relay attack. The relay attack involves a human to solve the challenges for the bots.

In this research, we solve the problem by presenting two systems: an API Verifier and an enhanced Captcha design. Since a bot must use an API (Application Programming Interface) to post information, the API Verifier will challenge a user with a Captcha if it detects that the API call is from a new computer. To recognize if the API call is from a new computer, we use the Media Access Control (MAC) address of the computer, which is globally unique. Since a bot cannot solve a Captcha challenge, it will not be able to make API calls to the Web service. Our enhanced Captcha is resistant against the relay attack as we change the static answer, which is used by majority of Captcha forms, to a dynamic one by asking the user *"where"* is the answer rather than *"what"*.

We also employ animation with a defined delay time to prevent the human solver from telling the bot where the answer is. The systems are evaluated by operating against various attacks from a modified version of Koobface, the most popular existing Web 2.0 botnet. The results show that Koobface bots fail to break our enhanced Captcha using relay attacks. Also our API Verifier successfully detects requests from Koobface bots and denies their access.

Contents

List of Figures

List of Tables

1

Introduction

A botnet is a network of compromised computers (called bots) that can be remotely controlled by an attacker through a predefined command and control (C&C) channel such as Internet Relay Chat (IRC), Peer-to-Peer (P2P) or Web-based protocols [1]. Botnets have been recognized as the biggest threat to the Internet because of the magnitude and potency of attacks afforded by their combined bandwidth and processing power. Attacks from botnets include DDoS attacks, spamming, traffic sniffing, key logging, identity theft and spreading malware [2].

These types of attacks used to happen before due to the appearance of computer viruses and worms, however, with the emergence of botnets, the scope, sophistication as well as damage of these attacks have been raised to a much higher level. For example, since 2006, email account holders have experienced an unprecedented growth of unsolicited emails (also called spam). It is interesting to learn why spam has grown so quickly despite extensive efforts of the leading IT companies (such as Microsoft) that have produced a myriad of anti-spam software tools. One reason seems to be that spam is profitable. It is a cheap and efficient way of advertising and luring unexpected users to give out their private data and/or download malware. Even during the economic downtime in 2009, the group of Russian spammers behind most Viagra spam still raked in an average of $4000 a day [3]. Spam has gained a prominence because it is a crucial component in many cyber-crime activities.

An interesting kind of spam is phishing. Phishing tries to coax naive users into giving up access to their bank accounts or credit card information. The stolen bank credentials are used to transfer money or withdraw money from victims' bank accounts while stolen credit card information is used to buy products online. Collections of stolen bank credentials are themselves products that are being sold online. In general, according to data presented in [4], a single phishing attack costs a US financial institution approximately $50,000, which gives the loss of more than $50 per account.

The dramatic increase of spam is due to the distributed nature of botnets. A large collection of bots working independently (although co-ordinated by their botmasters) is able to produce billions of spam emails per day. At the same time, it is very difficult to block the influx of spam. The designers of malware use different techniques to make the bots stealthy. Once a computer has been infected by a bot, it behaves in the same way as before the infection. This is quite different from an infection by a virus, when the performance of the infected machine deteriorates dramatically. To minimise a chance of detection, bots are active only when they perform necessary actions and most of the time are inactive.

There is a well developed anti-spam technology that is based on content filters, black lists (blocks emails coming from addresses or IPs in the list), white lists (only accept emails from addresses or IPs in the list) and challenge/response system (eg use

Captcha to verify if the sender is not a bot). Unfortunately, traditional anti-spam tools fail when spam is generated by botnets. Besides using different techniques to avoid content detection and break Captcha to pass challenge/response system, spammers now use thousands of other people's computers (botnet) to send spam without the knowledge of the owners. Many computers are only used to send spam once or once in a while [5] that makes it very hard to detect and shut down the spam sources. Since there are so many computers involved in sending spam now, there is no blacklist able to list them all. Even worse, anti-spam software with a high rate of false-positives create a lot of false alarms and block emails coming from legitimate sources.

The designers of malware started to use botnets to launch distributed denial of service (DDoS). This is a variant of denial of service (DoS) attack that uses all bots within the botnet to flood an attacked server with requests [6, 7]. Note that because bots are scattered over the Internet, it is very difficult to identify the source of the attack. The traffic in the Internet looks normal except in the neighbourhood of the server where all requests (generated by bots) converge. As a result, it takes a lot of effort and coordination among security managers of different network domains to identify the source of the attack. Clearly, identifying all bots is an option. A better option, however, is the identification of the botmaster and the person (cyber criminal) behind it. However, it is very hard to identify the bad guy controlling the DDoS attack because he/she uses other people's computers to execute the attack.

How big is the security threat coming from botnets? The number of infected machines in the Internet is difficult to determine because it grows quite rapidly. In 2009, McAfree [8, 9] estimates that 150 thousand machines get infected by bots every day, which is a three-fold increase compared to the rate reported by Symantec [10]. More recent works [8, 11] report that in 2010 the BredoLab botnet alone consists of more than 30 millions bots located on compromised machines. It is evident that the presence of bots alone poses a serious threat to the global Internet community.

In another research, Spamhaus.com [12] found that in 2010, botnets contributed 95% of the total spam. The king of spam was the Rustock botnet which produced 200 billion pieces of spam a day [13, 14]. According to Mailshine.com [15], estimation of global productivity cost of spam in 2010 was about $103 billion dollars. Loss of productivity was calculated based on:

- the average time spent by the recipient per spam message was half a second
- the average wage of the recipient is $10 per hour.

Other malicious activities of botnets such as DDoS, identity theft and click fraud also cause a lot of financial losses. During the last decade, the production and dissemination of botnets has grown into a multibillion dollar business. In the last survey conducted by Ponmone [16] for 50 US campanies that were the victims of botnet attacks in 2011, the average annual loss was just below 6 million US dollars for each company. According to Ponmone [16], damages caused by malware activities especially botnets together with the costs of preventive measures are likely to grow significantly in the coming years.

Botnets are already a part of the Internet landscape and their impact on the Internet (in)security is difficult to estimate. Unfortunately, the development of security countermeasures against botnets drags behind the progress in the botnet technology. Some experts claim that we are losing the botnet war [2, 17–19]. There are, however, some successes in the war against botnets. Here we give three examples.

- Honeynet project [20] is a leading international security research organization, devoted for investigating the latest attacks, developing open source security tools to improve Internet security and learning how malicious hackers behave. From a small group of members started in 1999, the project now expands into many chapters around the world. The valuable information collected by the project is not only useful for security research to analyze the latest attacks but also to educate the public about threats to information systems across the world.
- BotHunter [21] is a free detection software created by SRI International to help system administrators detect botnet activity within their network. By monitoring

the two-way communication flows between hosts within the internal network and the Internet and basing on state-based infection sequence models, BotHunter has successfully detected many types of bots including P2P ones.

- Rustock is a bot, whose main task is to distribute spam email. It is probably the most successful spam bot to date. It has infected more than one million PC machines [22]. However, the Rustock botnet was shut down by a group of companies led by Microsoft in 2011 [13]. The group was able to track the botnet C&C servers and identify their IP addresses. It turned out that the C&C servers were located on many servers of five major Internet service providers across the US. Outside the US, Microsoft worked with European law enforcement agencies to identify the C&C servers. Finally, all C&C servers were shutdown putting an end to Rustock activity.

Despite many successes, it is clear that there is a lack of comprehensive countermeasures to combat botnets. Such countermeasures have to be global covering the whole Internet. So apart from technical tools, there is a need for the co-ordination of countermeasures that are applied in different domains and countries.

Botnets work so effectively because they are able to communicate with their botmasters. The communication channel (also called communication and control (C&C) channel) can be built in many different ways. The Internet Relay Chat (IRC) protocol was the first and most popular protocol that was used as a C&C channel. However, since the IRC channel has become too easy to detect, the number of IRC botnets is on the decline [23]. Therefore, designers of bots have shifted to web-based and P2P services to implement C&C channels. With the growing usage of social networks, it has been observed a shift in the design methodology. So, designers of bots start using Web 2.0 social networks as a vehicle for the botnet construction. Bots that are using Web 2.0 services are called Web 2.0 bots.

An example of a Web 2.0 bot is Koobface. It is the most popular bot that has infected millions of machines. It targets popular social networks such as FaceBook, MySpace or Twitter, to propagate itself and to communicate with its botmaster. These networks as well as a number of Google services are also being eyed by cybercriminals not only to steal user data, but to use their storage and bandwidth for certain botnet command-and-control capabilities. Examples of such bots are:

- TwitterBot [24] was the first bot found to use a Twitter profile as a rendezvous point for delivering commands from the botmaster to her bots. It also uses the Twitter RSS to alert bots about new commands sent by the botmaster.
- Whitewell Trojan [25] uses a Facebook account as a coordinator for the C&C server. The bots receive some configuration data from its Facebook account. Commands are delivered through a third party Web server.
- Google Group Trojan [26] uses the Google Groups newsgroups to distribute commands. The botmaster creates an account (eg Escape[REMOVED]@gmail.com) and a page containing commands to the bots. A command consists of an index number, a command line to execute, and optionally, a file to download. Once the Trojan is executed, it logs onto the specified account to retrieve a command.

The new generation of bots and botnets (we are going to call them Bot 2.0 or Botnet 2.0, respectively) poses a new challenge for the computer security community. Gavin Gorman, a security expert from Symantec, summarizes the challenge by saying "By integrating C&C messages into valid communications, it becomes increasingly difficult to identify and shut down such sources" [26].

1.1 Research Challenges and Scope

The first and crucial step in the fight against botnets is their detection. The detection of malware (or in more general terms intrusion) can be classified into the two following categories:

- anomaly detection - the process behaves abnormally. This kind of detection is going to work if we know the signatures of the process normal behaviour,

- misuse detection - the process behaves illegally. In this case, the detection works if we can tell apart illegal behaviour from legal.

Note that bots are by definition processes that behave illegally. Consequently, the anomaly detection techniques do not apply to bots.

Intrusion/malware detection systems may also be considered as:

- host-based systems [1, 27–30], where the system inspects processes and their activities locally and tries to identify the processes that behave illegally,
- network-based systems sequences [5, 21, 31–37], where the system controls a domain of the network and inspects processes within the domain mainly by the traffic analysis.

The approach taken by us in this thesis is different. First, we target the specific type of bots, namely Bots 2.0. Second, we design a security solution that is based on the application program interface (API) that is specific to Web 2.0 services. The work has been inspired by the following research challenges:

- There is a lack of an appropriate approach to detect Web 2.0 bots. The approaches taken from the intrusion detection are not working efficiently as bots are stealthy processes that most of the time are inactive. They reside on infected machines for a long time dormant until they are instructed by their botmaster to act. To avoid being detected, they do not attempt to be harmful or to aggressively consume the machine resources (such as memory, CPU or communication bandwidth). Even if active, the bots use the popular social network APIs to blend the traffic generated by them with the normal traffic of the social network service. Thus, the bot traffic is just a small "background noise". The Web 2.0 also uses the standard port 80 that cannot be blocked unless the corresponding social network service is also blocked.
- There is a relatively small body of publication on protecting Web services from bots abuse. The thesis highlights the need to study the real security problems.

The appearance of Botnet 2.0 changes the landscape of the Internet security. In a sense, Botnet 2.0 is an example of an intelligent malware that sits silently and can be used for many purposes including gathering private information about users (their login names and passwords, for instance).

- The currently used detection techniques cannot distinguish human activity from bot activity. They normally look at the contents of communication or storage and use some heuristic rules to filter out suspicious contents [38]. In our work, we propose a security solution, in which if there is a reasonable suspicion that the sender of the contents is a bot, then the sender is asked to solve a Captcha challenge. Once the challenge has been correctly and timely solved, then the sender is accepted as a human user. Otherwise, the content is rejected and the sender is identified as a bot. Note that a standard off-shelf Captcha solution is not good enough as bots can be equipped with Captcha breaker components. We are going to discuss an extension of Captcha in the thesis as well.

- Majority of Captcha designs are subject to the relay attack. In the attack, a bot attempts to get a human user (with or without their agreement) to solve the challenge for it. As Captcha is designed to be easy for human, the relay attack works well with a very high success rate. In our work, we modify the Captcha design to address the security threat coming from the relay attack.

Having the above research challenges in mind, the main goal of this work is to examine new approaches in protection of Web services from botnet abuse. In particular, we propose the design and implementation of the following two systems: API Verifier and enhanced Captcha. The Captcha is built in such a way that it is resistant against the relay attack. In our considerations, we ignore distributed DoS attacks because of the following reasons:

1. There is a significant body of work that discusses how to cope with DDoS attacks [6, 7, 39].

2. Popular Web services such as Google, Facebook and Twitter can cope with large flow of traffic from millions of users. Hence, DDoS attacks seem to be not a significant threat against these services.

1.2 Our Research Goals

Our main research goal is twofold. First, we would like to design system that is able to prevent Web services being exploited by Web 2.0 bots. Secondly, we intend to define security measures of the system and check the level at which the measures are satisfied. To achieve these goals, we are aiming to accomplish the following tasks:

1. Overview of the current bot and botnet technology together with description of main security countermeasures. A special emphasis is going to be placed on bot detection techniques. To be able to detect bot and botnet, it is vital to understand how bot works, botnet characteristics and communication between bots and their botmaster. A special attention is put on Web 2.0 bots. We review and study the existing detection techniques in view of using some of them in our work.

2. Propose a system that is going to be able to distinguish a human request from a bot request to Web services. Our system should be general and customizable. By general we mean that our system should not only detect a bot that uses a specific Web 2.0 service but also work for bots based on any Web 2.0 service. Our system is customisable in the sense that it can be adjusted to specific requirements (eg configurable parameters such as session time and number of users' attempts on Captcha challenge or portable Captcha). In addition, we will provide practical prototype systems that can work in a real-world application.

3. Define features of our system that cannot be changed or spoofed by bots. These features are going to be used to uniquely define the machine on which bots or user processes are running. In particular, we are going to use the permanent media access control (MAC) address to uniquely identify a machine. We also

use a Captcha to distinguish bots from human users. Note that the MAC of a machine cannot be changed by bots and also a well designed and frequently updated Captcha is able to distinguish bots from human beings with very high probability. It is also interesting to compare the features we have chosen for our security system with the ones used in other solutions. The other anti-malware software is based on behaviour signatures that needs to be updated frequently to keep up with the evolution of malware. More importantly the behaviour signatures are very malware specific as they work only for a one type of malware.

4. Define a collection of security measures further on called attacks to check the security of our proposed system. The system security is going to be analysed against the defined attacks. For the API Verifier, we will cover all attacks that attempt to spoof the permanent MAC address, create API Verifier fraud as well as bypass the Captcha verification. Since we will use images to deliever challenge in our Captcha design, three main attacks including dictionary, image recognition and relay attacks will be covered in the security analysis of the Captcha. In addition, the prototypes of our systems will be tested against a modified version of a real bot, Koobface, to evaluate their security capability.

1.3 Bot 2.0 Modus Operandi and Solution Overview

Botnets 2.0 harness the Web 2.0 infrastructure to build their command and control channels and to store stolen data and instructions for bots. A typical Botnet 2.0 consists of two components:

- a bot - a program that performs some specific tasks on behalf of the botmaster, and
- a botmaster - a malicious hacker that co-ordinates the work of bots across the network.

A bot and its botmaster communicates over the C&C channel that is implemented on top of Web 2.0 services. The setup of the C&C channel proceeds as follows:

1. the botmaster creates a profile on a public blog service or a social network site. It uses the profile to post commands and updates for her bots,
2. bots are being alerted by really simple syndication (RSS) that informs users of new updates to blog or website,
3. after receiving RSS feeds, bots fetch the data and execute the command accordingly,
4. bots upload data harvested from the victim host to the profile
5. the botmaster is alerted by RSS and collects the stolen data.

As the Botnet 2.0 uses public services available to everybody, the communication between the botmaster and its bots is a part of regular web traffic. Thus, most anti-malware software is not able to detect bots [40]. Also the exchange of messages between bots and botmasters are not direct but via a public registry storage. This means that it is quite difficult to identify the communicating parties as the receiver collects the message when the sender is no longer active.

Users can update their profiles either manually via their browsers or alternatively they can write auto-update programs using API tools. As bots are computer programs, they may interact with Web services by either

- script programs for web browsers that are available on the infected machines (this option is not too attractive for bots as this activity may be discovered by the owners of the machines and as the result, bots may be removed) or
- API programs that update the contents of profiles stored on a web service.

Considering the structure of Botnets 2.0 and their inner working, we propose a detection system, named API Verifier, which is able to check whether the entity requesting access to Web 2.0 services is a human being or a bot. Our API Verifier can provide a very first line of defence for the Web 2.0 services against bots. All API calls made to

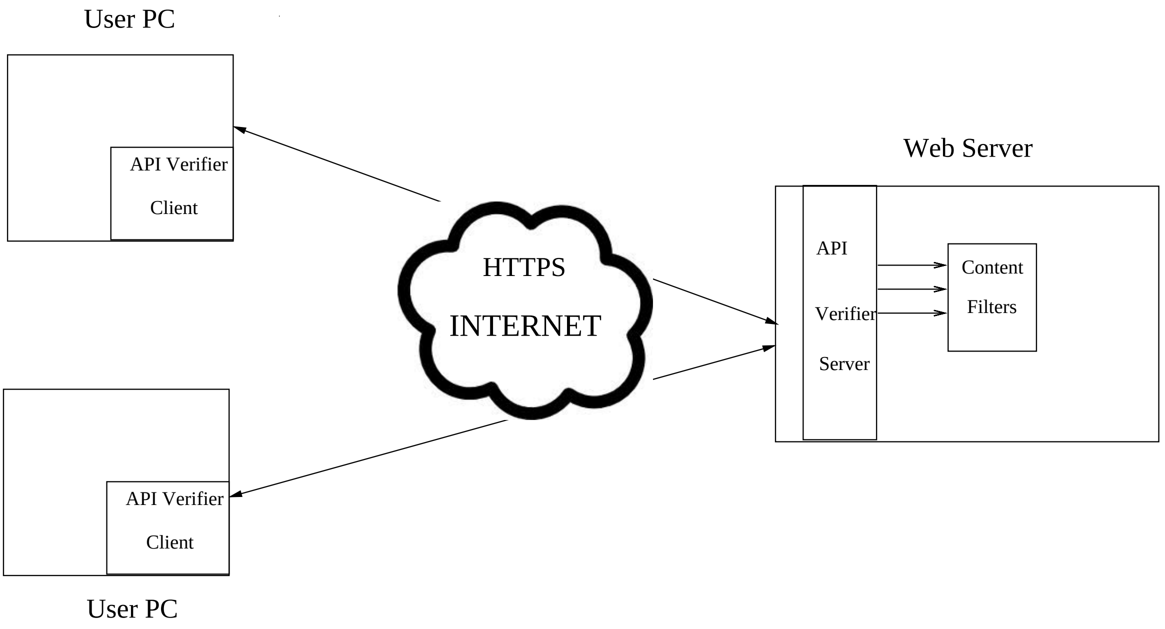


FIGURE 1.1: Our proposed solution fitted in Web services defence system

Web services are filtered by an API Verifier. If an API call comes from a bot, it will be denied. It is known that most Web services defence systems are usually looking at the posted text contents and filtering suspicious posts using heuristic rules [38]. However this method usually fails to detect malicious posts as advanced botnets tend to hide their communication and fake links by using encryption and obfuscation. This method is also expensive as it demands a lot of resources for monitoring thousands or millions posted contents a day. By using an API Verifier, most of the API calls from a bot will be filtered out so it requires much less resources to check the remaining contents coming from API calls. The proposed solution is practical, inexpensive and delivers a reasonable level of security. It is illustrated in Figure 1.1.

The API Verifier consists of two following components:

- API Verifier client (APIV client) and
- API Verifier server (APIV server).

A copy of the APIV client is installed on a machine as a part of the API library of the Web service. All API calls generated by users of the machine have to use the APIV client. The web server installs a copy of the APIV server that monitors and filters the user access requests. APIV clients talk to their APIV servers. We assume that users are not able to bypass the API Verifier.

The API Verifier has been designed using the following principles:

- any API call coming to the Web service has to pass through its APIV client. Any API calls that are not coming from APIV clients will be denied,
- any API call coming from a new computer goes through a checking process to identify whether the entity making the call is a human being or a bot. This is done by using Captcha,
- any machine in the network has a unique (unforgeable) identifier. We use media access control (MAC) address of the Ethernet/Wireless card of the computer as the identifier,
- communication via channels between APIV clients and their APIV servers is protected by an encryption (the AES algorithm with 128-bit cryptographic key).

Though we propose our solution based on the Bot 2.0 model, it is most likely to work on Web 2.0 bots in general as these bots all need to use the API to update content to the Web services. This matter will be discussed in Chapter 6 describing implementation and testing of the proposed system. Details about the system design will be described in Chapter 4. As part of our solution, we also propose an enhanced version of Captcha. The enhanced Captcha addresses the recent advances in algorithmic solutions of Captchas. More importantly, the new Captcha is used to study its resistance against the relay attack, in which a bot sub-contracts human beings to solve Captcha challenges on its behalf.

To mitigate the relay attack, we modified the design of Captcha. Instead of static challenges, where the expected answer is *"what"*, we are going to use challenges, for which the expected answer changes over time. In other words, expected answers to challenges are *"where"*. Thus we use animation to change the positions of objects in a Captcha challenge. We also define a reasonable delay time between two animation based on a simulation mimicking a real relay attack on the proposed Captcha.

1.4 Thesis Contribution and Organisation

In this thesis, we make the following main contributions:

1. We have proposed a novel approach in protecting computer networks against bots. The proposed solution not only detects bot activities but also filters out API calls made by bots to a public Web service. The solution can be easily upgraded so it can be used to automatically notify the administrators of infected machines about the bot activity. There is a drawback that is a logical consequence of our design principles. If a user chooses to use a script program to automatically update their web profile, they need to solve a Captcha challenge for the first API call as it is classified as an API call from a new computer. Once the user passes this verification, the call with following API will be accepted without the need to solve another Captcha.

2. We have designed an enhanced Captcha system that is able to mitigate the relay attack. The enhanced Captcha turns away from a static to a dynamic drag-and-drop design. It uses animation combined with noisy or distorted images to distinguish bots from human beings. The design is also dynamic in a sense that it needs to be adjusted to the current advances in the image recognition algorithms. Our Captcha is portable and can be applied in many applications such as registration, online votes and stopping spam.

3. We have implemented and tested both the API Verifier and enhanced Captcha.

> The prototypes have been tested and the experiments have shown that the systems perform as expected and are able to cope with the attacks that reflect the real-life threats.

The remainder of the thesis is organized as follows. Chapter 2 provides descriptions about bot and botnet including their characteristics, activities and evolutions. Bot is also compared with other malware to be able to detect and differentiate a bot with other malware on the infected hosts and networks. In Chapter 3, Web 2.0 bots are studied further to investigate their ways of exploiting Web services for their malicious activities. We also define Botnet 2.0, which is a network of Web 2.0 bots that leverage Web services as their C&C servers and storage, and present the model of attack. Also in this chapter, we review some popular botnet detection techniques and relate them to our work. Chapter 4 introduces the API Verifier system including its design and security analysis on some major attacks on the system. Its limitations and possible improvements to the system are also discussed in this chapter. Chapter 5 presents a brief literature review about Captcha as well as introduces our enhanced design and analysis on its ability to be resistant against some possible attacks including relay attack. In Chapter 6, we evaluate our prototypes of the API Verifier and the enhanced Captcha by conducting some main attacks from a modified version of an existing botnet, Koobface. Finally, Chapter 7 concludes the thesis and describes directions for future work.

2 Botnet

Understanding botnet characteristics is essential for botnet investigators. This chapter describes bots and botnets in details including botnet evolution and their main characteristics. Section 1 starts from the definition of botnets. It is then followed by a description of botnet developments over time and the future trends in protection techniques against botnet activities. Section 3 elaborates on characteristics that are unique to botnets that can be used to tell them apart from other malware. Other sections discuss main botnet characteristics including its life cycle, possible architectures, communication protocols and evasion tactics.

2.1 Botnet Overview

2.1.1 Botnet Definition

Botnet is a network of compromised computers, which are called bots (or zombies or drones). "Bot" is derived from the word "robot" which is an automated process that interacts with other network services. A typical use of bots is to gather information (such as web crawlers), or interact automatically with instant messaging (IM), Internet Relay Chat (IRC), or other web interfaces. The hacker controlling the botnet is called a botmaster or a botherder. It is also important to distinguish between a computer that is a bot (or botnet client) and a computer that has been compromised by a hacker. According to Schiller et. al. [1], a collection of computers that meets the following characteristics is a botnet:

i. a computer that can be accessed by a botmaster without the need to log into the computer operating system.

ii. the computer acts in a co-ordinated manner determined by the botmaster to accomplish a common goal. Once such goal is set, the botmaster does not need to interact with botnets.

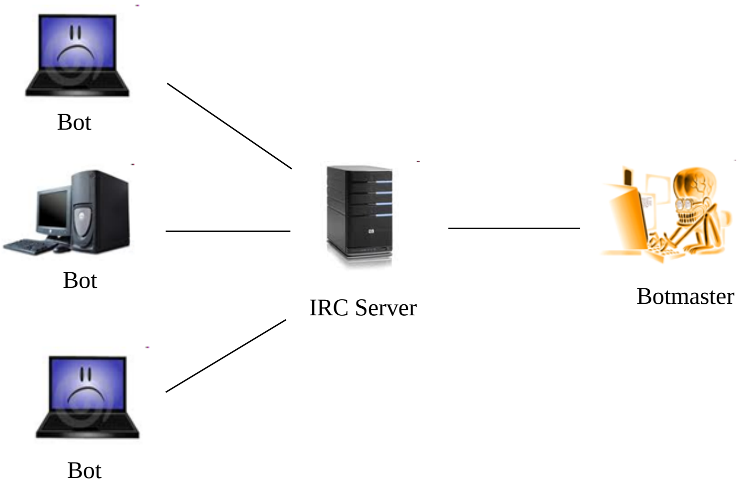


Figure 2.1: A typical botnet

A typical botnet consists of a bot server (usually IRC server), one or more bot controllers (optional) and hundreds or thousands of bots. Each bot after joining the botnet is sitting on the IRC channel and listening to commands. When making an attack, the botmaster issues a command to a bot server and the server relays it to all bots. The bots then execute the commands and report back the results to the server. See Figure 2.1.1

2.1.2 The Botnet Life Cycle

The life of a bot begins when a computer has been compromised. A computer is compromised by a malicious code downloaded by a careless user or by a software vulnerability or by a Trojan horse malware. The bot then initiates contact with the botnet command and control (C&C) server to report the exploitation results. At this point, the bot may get updates. The updates can be other modules or exploit software or list of targeted IP addresses etc.

The next step undertaken by the bot (and instructed by the botmaster) is to make sure that the downloaded malware is not identified and removed by antivirus tools. The bot may download software from the C&C server and run it to shut down antivirus tool, hide from it or render it ineffective. After that, the bot will install utilities that are needed for its activities. Then it uses these utilities to “investigate” the computers to collect information such as user information, log files or find tools of other hackers previously installed in the computer and send a report to the botmaster. The botmaster may need to compete with other hackers to maintain his control over the computer.

When the bot is secured and controlled by the botmaster, it listens to C&C communication channel and waits for orders. After receiving a command, the bot retrieves new module(s) if needed and executes the command, erases all traces or evidence of the attack and then reports back the results. Depending on configuration, some bots are even able to automatically search for vulnerable computers in the neighbourhood and infect them so they become new bots. The botnet life cycle can be summarised in

Figure 2.2

2.2 Botnet Evolution

Within the last decade, many infamous botnets have attracted considerable media coverage. Table 2.1 lists several examples of such well-known botnets and briefly describes their features. In this section, we are going to trace the evolution of the bot technology.

Internet Relay Chat (IRC), invented in 1988 by Jarkko Oikarinen, provided an opportunity for group (many-to-many) communication in real time. The IRC channel allows uni-cast communication, in which a user can monitor a conversation between multiple users and can participate in the conversation [41]. The creation of bots, short form of robots, came shortly after that. The IRC bot is a process (client) that has been programmed to respond to various events and to help maintain the channel moderator on a particular channel [42]. Unlike today's botnet clients, these bots were created as a benign assistant to IRC channel management. For example, GM (Game Manager) bot, created in 1988 by Greg Lindahl, would play a game of Hunt the Wumpus with IRC users. From the simple GM bot, bots gradually have been developed into a comprehensive tool which operates as an IRC channel operator. For example, the Eggdrop bot was written (in 1993) to assist channel operators that were handling tedious 24 hours per day requests from users. The main task of the bot was to keep the channel open and secure against malicious users who may be trying to take over the channel. Around this time, some IRC servers and bots began offering the capability to make OS shell available which permits users to run commands on the IRC host. This also invited malicious activities such as killing the IRC channels and removing users from these channels.

The first known malicious bot called Pretty Park was written in Delphi and was launched in 1999 (see [43, 44]). Pretty Park has an ability to self-propagate via email

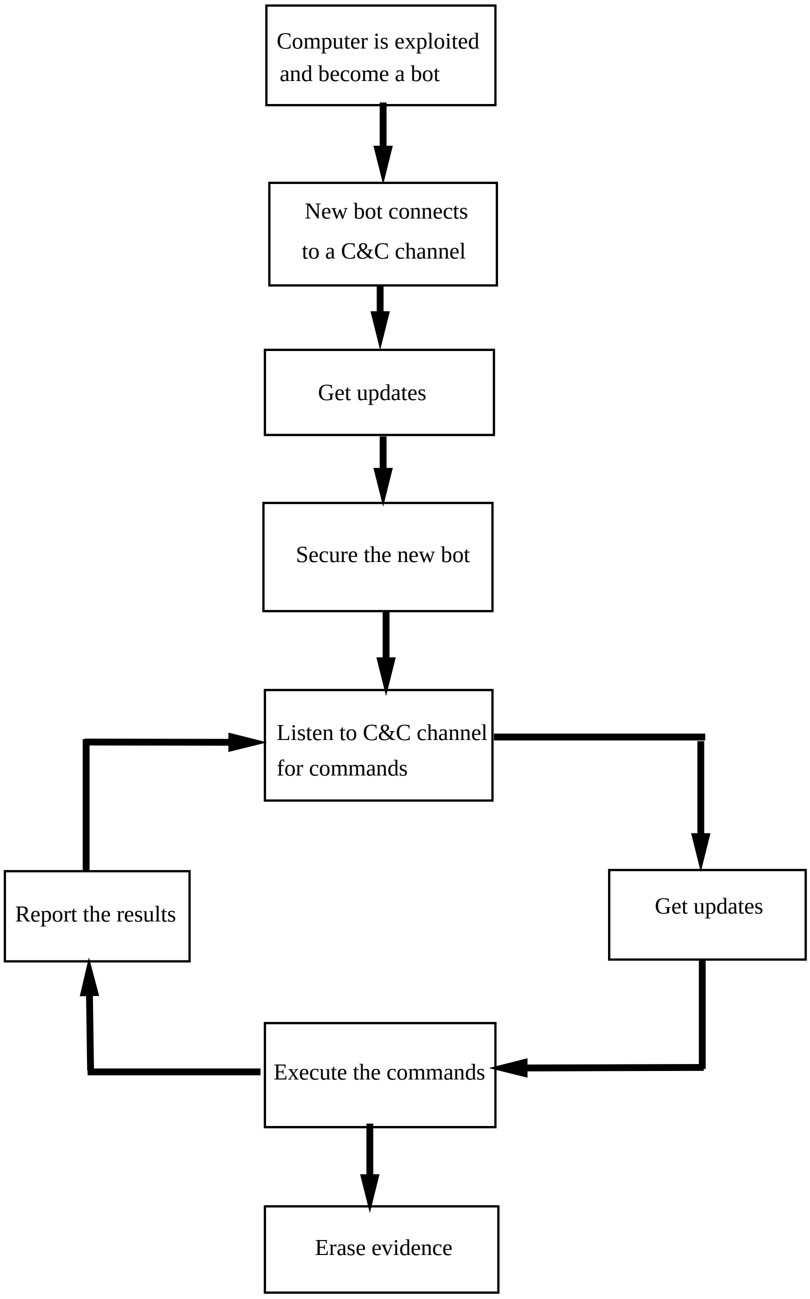

Computer is exploited and become a bot
New bot connects to a C&C channel
Get updates
Secure the new bot
Listen to C&C channel for commands
Report the results
Get updates
Execute the commands
Erase evidence

Figure 2.2: The Botnet Life Cylce

Date	Name	C&C	Structure	Description
04/1998	GTBot	IRC	Centralized	First IRC bot
04/2002	SDBot	IRC	Centralized	First stand-alone and open source IRC bot
10/2002	Agobot	IRC	Centralized	Very flexible and first modular designed bot
09/2003	Slapper	P2P	Decentralized	First peer-to-peer bot
05/2004	Bobax	HTTP	Centralized	First spambot using HTTP as C&C
04/2006	Nugache	Self-defined	Decentralized	First P2P bot connecting to predefined peers
2006	Rustock	HTTP	Centralized	Botnet using rootkit technique and is responsible for 60% of spam
01/2007	Storm	Kademlia (P2P)	Hybrid	Famouse and large scale spamming botnet
11/2008	Waledac	P2P	Hybrid	Decedent of Storm botnet with larger scale, mostly involved in email-spam
2007	Zeus	HTTP	Centralized	Probably the most powerful US based botnet with more than 3,5 million bots (US only) mainly targeting online-banking
12/2008	Koobface	HTTP	Centralized	Welknown web 2.0 botnet using social network sites for propagation
2011	TLD4	P2P	Decentralized	First kernel mode rootkit compatible with x64 Windows and using bootkit technique to conceal

TABLE 2.1: A collection of some well-known bots

and attaching itself to the email body. Unexpected and careless users infect their computers when opening mails. The bot then installs its own IRC client on the computer, connects to the IRC server and initiates the file transfer to send users private information (such as password and credit card information).

In 2000, a new bot called Global Threat (GT) which based on mIRC client for Windows was released. mIRC is a user-friendly IRC client that offers users many options and tools [45]. The GT bot is still popular as it exploits mIRC powerful scripting language, which includes support for raw socket connections to create a bot that is easily controllable via IRC [1]. GTBot, however, does not spread itself directly like Pretty Park but via malicious links. If the user clicks on the link provided in the spam email, they will be taken to a website that delivers the bot to the victim.

A next generation of bots arrived in 2002 with the appearance of SDBot. It started a new trend in bot technology. The source code of SDBot is written in C++, is small and easy to modify and maintain. Therefore, though this is one of the oldest bots, SDBot is still very popular now. The main characteristic of the SDBot is the inclusion and use of remote control backdoors. A backdoor is a method of bypassing normal authentication to remote access to a computer while attempting to remain undetected. SDBot originally propagated itself primarily via network using blank or common password such as “admin”, “guest”, “ 123456” or “password”. Modern variants of SDBot are spreading themselves by other means including malicious scripts hidden in emails or in downloads taken from phishing sites. One of prominent variants of SDBot is SpyBot. It differs from SDBot as it is equipped with extra spy-like capabilities such as keystroke logging, email address harvesting and more. Both GFBot and SDBot are not modular - the code is a single piece of instructions.

Agobot (or Gaobot) arrived in 2002 and was the first bot with modular design and significant functionalities, which makes Agobot the most popular bot in history. Agobot has 3 modules and is designed to be both easy to utilise and update. Agobot does not download entire code to victim system at the same time but one after another as follows:

1. The initial module contains the IRC bot client and the remote access backdoor.

2. The second module disables the computer defences or alternatively hides the bot from anti-virus software.

3. The third module prevents the user from accessing a list of websites (usually antivirus vendor sites).

Consequently, the hacker is able to update modules 2 and 3 in case victim computers have a new more effective anti-virus software installed. Modular design allows hackers to modify Agobot to add new capabilities and upgrades. This obviously has led to creation of thousands of Agobot variants. Another interesting feature of Agobot is the separation of the C&C channel (using IRC) from the propagation channel (using P2P file sharing applications).

To avoid intrusion detection tools applied on the network level and prevent bots against being hijacked by competing botmasters and law enforcement agencies, since 2003 the bot technology has started to use compression and encryption. The first example of such a bot is RBot which also has a modular structure. In addition to spreading via weak security on network shared resources, RBot also uses software vulnerability exploits in the Windows OS and common software applications to propagate.

In 2005, research conducted by the Honeynet Project (a leading international security research organization dedicated to investigating the latest attacks and developing open source security tools) revealed that more than a million computers were compromised and were members of different botnets [20]. More advanced bots have been created since then with more sophisticated functionalities. It is worth noticing that hackers prefer to modifying existing bots to designing new ones from scratch. Therefore, it is believed that today bots and their families are developed based on the above "original" bots [1, 46, 47].

However, the improvement of countermeasures against botnets or more specifically a better detection of botnets has a dramatic impact on development of new botnet technologies. In 2002 the traditional IRC communication protocol has been replaced by a variety of new protocols. Most of them are based on the Web-based HTTP protocol or the P2P protocol. Also starting from 2003, botnets use different techniques to propagate themselves. These techniques exploit such vulnerabilities as buffer overflow, and dictionary attack [1, 47]. A buffer overflow occurs when a program or process tries to store more data in a buffer (temporary data storage area) than it is intended to hold. The buffer overflow bug allows hackers and automated network worms to overwrite portions of a computer's memory that are supposed to be off limits with their code to install malware or trigger specific bot actions. Droppers are programs that are used to install malicious software in a target host while remaining undetected. Droppers have ability to get malicious content past gateway security checking practice, such as scanning downloads and email. Bots also use dictionary attacks on administrator passwords to gain access and spread themselves to computers in the neighborhood.

The biggest weakness of the IRC protocol is that it is centralized. If a security investigator manages to discover and eliminates the central C&C server, the entire botnet will be useless and ineffective. Therefore, to overcome this weakness, botnets have recently moved to P2P technology. The Slapper bot (2003) was the first that based its communication on the P2P protocol [48, 49]. Other examples of these kind of bots include Sinit, Nugache and Storm [48–51], each has its own advanced design and weakness. Slapper maintains a list of known bots for each compromised computer to remove the bootstrap process; hence it is harder for defenders to detect and shutdown. Bootstrap is a procedure of a new infected host getting the information about other bots which can be connected through other systems. For example, the Phatbot obtains the other bots' information through the Gnutella cache servers [52]. Though bootstrap is efficient in finding peers, it is easily exploited by defenders. One weakness of Slapper is that it does not perform authentication, so it is easy for other attackers to hijack it. The designers of Sinit have learnt the lesson from Slapper and applied public key cryptography for Sinit authentication. A Sinit bot uses random probing to find other Sinit bots to communicate with. In particular, Sinit tries to reach other Sinit infected hosts by sending special `discovery` packets to port 53 of random IP addresses on the Internet. When Sinit receives a `discovery` response packet, a connection between Sinit and the hosts sending the response is established [53]. However, the extensive probing generates a lot of traffic that can be used to detect bots. On the other hand, Nugache has implemented an obfuscated C&C channel to improve its resilience. However, Nugache's weakness lies in its reliance on a seed list of 22 IP addresses during its bootstrap process which make it an easy target for detection. To overcome the weaknesses of Sinit and Nugache, a new bot called Storm was designed. It uses sophisticated encryption algorithms for its C&C channel, hence, the bot existence is difficult to detect. At the time (2007), it was the world most active and infamous botnet and was a preferred choice for sending spam. Its decedent, Waldelac appearing in 2008, enhanced the Storm features with a larger scale [49]. In 2011 a new bot called TLD4 was discovered. It is dubbed as "indestructible botnet" because it is difficult to detect. It infected more than 4.5 millions of computers. According to Golovanov et al [54], TLD4 is one of the most sophisticated and decentralized botnets that combine encryption and rootkit capabilities to make it practically invisible to infected machines. A rootkit is a collection of tools that is used by the attacker to gain administrator access privileges on a host while actively concealing its presence from administrators by subverting standard operating system functionality or other applications [55, 56]. TLD4 is the first that uses a kernel mode rootkit and it is compatible with X64 Windows.

Another alternative of IRC is the HTTP protocol. The first HTTP bot appeared in 2004 and was named Bobax. It was followed by another bot called Rustock, which was discovered in 2005. Rustock became very popular and was responsible for 60% of all spam. Another example of HTTP botnets was Zeus that appeared in 2007. It was designed to steal banking credentials of bank customers. Zeus was also well designed making it easy for anyone to create a custom version of the malware. Therefore, the Zeus authors not only gain revenue from stolen bank information but also by trading their Zeus toolkit [57].

The evolution of communication paradigm (from IRC to HTTP) was triggered by advances in exploit kit technology. An exploit kit is a software package that contains malicious programs that target specific weaknesses of machines and their operating systems. Once the exploit kit has been installed on a machine, it will automatically determine which malicious program to activate to compromise the machine. The switch to the HTTP protocol is a clever move by bot designers. The communication between botmaster and her bots is performed via HTTP port #80. This communication is very difficult to distinguish from the normal traffic generated by a large number of web browser users. Most of the firewalls allow HTTP traffic as the Internet use is the most popular service. Closing HTTP communication by firewall will dramatically reduce the usability of Internet services and in some circumstances this is not an option.

As exploit kits are becoming more accessible and the botmaster can control the bots with easier user interface for HTTP based botnets (usually using a regular Web browser), it seems that Web protocols, both encrypted and clear, are widely used as the command replacement for the traditional IRC protocol. Figure 2.3 illustrates the number of botnets tracked by Team Cymru in 2010 [58]. The green line shows the prevalence of IRC based C&C botnets while the blue line represents HTTP (Web-based) C&C's. The red lines represent the trends of each (averaged over 30 days). It is clear from the graph that IRC botnets have remained steady whilst HTTP based botnets have been trending upward and doubling in number over 6 months.

Since Web 2.0 evolved, malware writers have also rapidly adopted these new technologies to increase the sophistication of their attacks. According to various reports from security research institutions [14, 59, 60], Web 2.0 botnet has become a new trend in

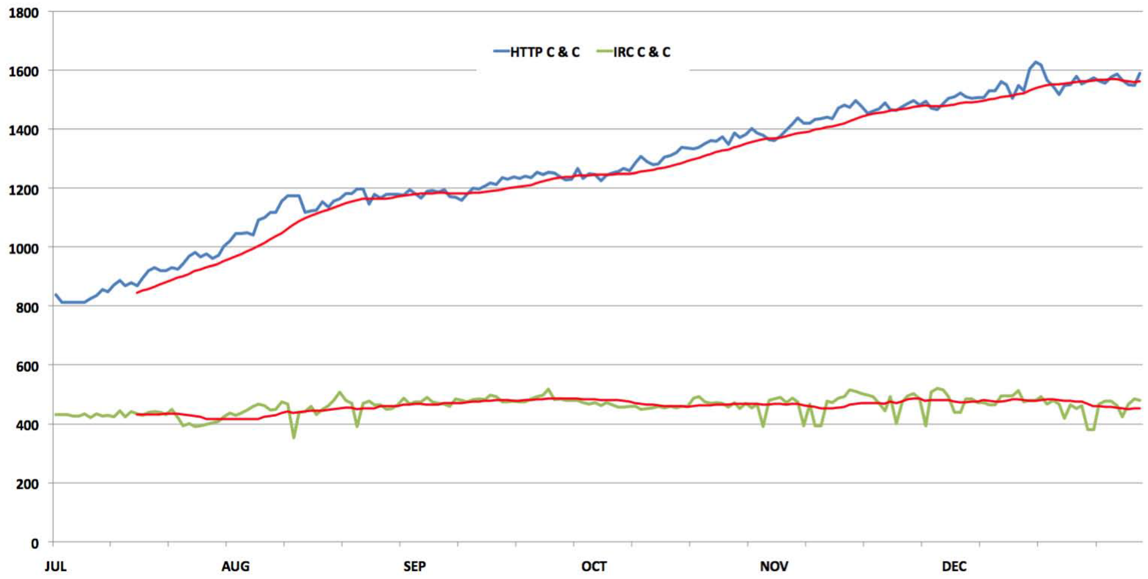


Figure 2.3: Rise of the web based botnets

recent years. More about Web 2.0 botnet will be described in chapter 3.

2.3 Bot characteristics

Internet is facing different types of malicious attacks generated by malware - malicious software designed with an intent to damage, disrupt, steal, or in general inflict some other harmful or illegitimate action on data, hosts, or networks. Malware is the general term covering all types of threats to computers security such as virus, Trojan horse, rootkit, malicious bot, spyware and so on. In order to detect botnet, it is important to understand and differentiate bot behaviours from other types of malware. In this section, we are going to compare botnet characteristics with the characteristics of other types of malware such as viruses, Trojan horses and worms. The summary is given in Table 2.2

Malware	Propagation Methods	Goals	Malicious Activities	C&C	Concealment
Virus	Attached to an executable file and need a user to run the file	Fun-driven or Vandalism	Damage host	No	Normal
Worm	Self-propagation and social engineering	Fun-driven or Vandalism	Damage host, spam, DoS	No	Normal
Trojan horse	Social engineering	Fun-driven or Vandalism	Pop up Ads, spy, damage host	No	Normal
Bots	Self-propagaton and social engineering	Profit-driven	DDoS, spam, identity theft, click fraud	Yes	High

Table 2.2: Malware Comparison

2.3.1 Definition of Malware and their Propagation Methods

A computer virus is a malware program that, like a biological virus, is dependent upon the "life" of a host in order to reproduce. Computer viruses do not exist and function independently, rather they attach themselves to other executable programs, and are run when those programs are run [56]. In other words, the virus may exist on a system but will not be active or able to spread until a user runs or opens the infected file or program. When the infected program is executed, the viral code is executed as well. Viruses spread to other computers when the software or document they are attached to is transferred from an infected computer to another using the network, a disk, file sharing, or infected e-mail attachments.

Similarly, worm spreads autonomously like a computer virus, but needs no host program that it can infect. There are two methods of worm propagation [61, 62]. The first are worms that are capable of total self-propagation. That means the worm requires no direct assistance from the computer user in order to spread. In particular, these worms perform network scans for hosts with exploitable vulnerabilities, and then copy themselves to new hosts and activate themselves through the exploit. The second method involves some type of user intervention to help the worm spreading. Instead of seeking out vulnerable machines, this type of worm uses some kind of social engineering to trick users into downloading and executing them. A typical ploy is to send out phishing emails to try to get an unsuspecting user to download and execute a program from a website or open a malicious attachment. Some worms use the web to spread by exploiting browser vulnerabilities. The second method may be slower but it depends on the popularity of the websites used for propagation.

Unlike computer virus and worms, Trojan horse only propagates by using social engineering (therefore, some types of worms using the second method of propagation are also referred to as Trojan horses). Trojan horse, named after the wooden horse the Greeks used to infiltrate Troy, is a piece of harmful program that looks legitimate and is hidden inside another program that serves a useful purpose [56, 62]. A common method for spreading Trojans is to place them on games or pornography websites and trick users into visit these traps by using phishing emails or pop up advertisers. In addition, by using an iframe attack, in which the attacker injects malicious code or redirect links to a malicious website to an inline frame of a webpage [63], legitimate websites can be used for malware propagation.

Bots have all the advantages of worms and Trojan horses in their infection capability. For example, they propagate like worms through network shared resources, file-sharing platforms, peer-to-peer networks, backdoors left by previous worms and/or exploit vulnerabilities. They also use social engineering for propagation but generally in a much more versatile way. Recently, in one emerging trend, botmasters use compromised Web services to infect those who visit the websites through drive-by download [63]. For example, the Koobface botnet has used popular social networks such as Facebook and Twitter as a means of propagation [60]. The bot firstly steals username and password of the victim users then posts fake “funny” videos (eg Youtube) or image links to the victim profile to lure victim’s friends . Once clicked, these links will ask users to download an update of a program (eg update flash version) to be able to view the “funny” content. Consequently, they may download a bot to their computer.

By using multiple propagation methods, the botmaster can recruit many victims. Currently, a botnet typically contains tens to hundreds of thousands of bots, but some had several millions of bots such as Zeus, Conficker and Koobface botnets [60, 64]. Distinct from all other types of malware, the bot may need to wait for the botmaster command before searching for new victims and it may be updated or new components can be installed.

2.3.2 Malicious Activities

Viruses and worms are involved in similar malicious activities causing damages whose severity ranges from annoying to catastrophic. The following list gives a typical collection of damages [56, 61, 64]:

- Some or all files in the infected computer are deleted or modified. More seriously, the computer hard drive may be reformatted and making the computer unusuable.

- The computer runs out of memory because the malware has consumed the available free memory. This is a DoS attack on the computer. Some viruses/worms plant monitoring software or change security settings that allow hackers to enter the computer without the owner’s knowledge so they can steal private information or/and control the computer.

- Some worms steal confidential information such as (bank, email) account usernames, passwords or credit cards information and email it to the worm writer.

- Unlike viruses, worms are capable of self-propagating across network. They also tend to consume memory resources and bandwidth causing the internet services to slow down or stop responding.

Trojan horses are hidden in applications that appear to be useful. Once activated, they can launch a number of attacks on the host computer. The attacks can include a variety of malicious actions starting from reading and deleting your files, reading your keystrokes including passwords, reading your emails, sending a lot emails, distributing viruses to manage the victim's bank account [56]. A common example is a fake login program (e.g login to Internet banking), which collects account information and passwords by asking for this info just like a normal login program does. Some Trojan horses are created for annoying purposes such as changing desktop icons or popping up advertisers. Others can cause serious damage by deleting files and destroying information on the system. Trojan horses are also known to create backdoor that may allow a malicious person to access to the infected system.

Today's bots combine features of other types of malware. For example, a bot is capable of self-propagating like a virus, can send spam like a worm, is able to create a backdoor for malicious access like a Trojan horse, can spy on the victim computer as a spyware, may steal login credentials by using keystroke loggers and may use rootkits to hide itself from malware detection software. Therefore, the bot can perform any malicious activity that other malware can but its actions are much more sophisticated. Once instructed by its owner, the bot performs its activities autonomously. Usually, many (hundreds or thousands) bots conduct the attack simultaneously and independently.

While other types of malware are mostly designed for fun (by hackers who would like to show their technical skills) or to annoy users (by displaying irritating adds) or to damage the victim computer, bots are designed with a specific illegal activity in mind that provides a financial gain. So bots are tools of choice for organised cybercrime groups. There are four main types of botnet malicious activities:

- **Distributed Denial of Services (DDoS):** The earliest and still most popular utilization of botnet is to launch DDoS attacks against channel operators or Web servers [20]. DDoS attacks attempt to overwhelm targets with an immense volume of traffic in order to exhaust system resources required to provide a service, which results in legitimate users being unable to access the service [6, 39]. Different from DoS attacks, which operate on an individual basis, DDoS attacks recruit hundreds or thousands of bots to flood the victim system. Diversity of botnets allows the attacker to launch DDoS attacks from different source addresses, which make it difficult to shutdown or filter. The measures to counter these attacks that include moving sites around the network, replication of services, data recovery plans, etc., are very expensive and according to the analysis given in [65], they may cost up to several millions of dollars per incident. According to a McAfee report [66], published in January 2008, botnet DDoS costs an average enterprise around 6.38 millions US dollars in lost profit for a 24-hour outage. At the same time, the botmaster might earn several thousands of dollars for leasing their bots. Originally, DDoS attacks were intended to disrupt business of a competitor. Recent developments show that they may also be used as a political tool. For example, in 2007 and 2008 a well organised DDoS attack against one of Eastern European countries grounded most of the Internet services for days. This illustrates the power of botnets as a weapon in the cyberwar.

- **Email/Instant Message Spamming:** Botnets are also used as a tool for sending spam emails. The addresses of receivers are harvested by bots. Spam can also be delivered in the form of popup advertisements or as instant messages [47, 67].

  Fastcompany.com reported in 2010 that 107 trillions of emails were sent and 89% of the volume was spam [68]. Apparently 95% of all spam was distributed by botnets [12]. A lot of the well known bots have been used as tools for spam delivery. Bobax [69] was one of the early bots that was designed for spam delivery using the HTTP protocol for the C&C communication. The Storm worm (aka Peacomm) [49] is another example of a bot used for spam distribution. Rustock (aka Spambot) [70] is probably the most popular bot used for spamming and therefore called the king of spam. According to Symantec's report, published in March 2011, Rustock generated more than 200 billion spams each day [13, 14]. A remarkable kind of spam is phishing. Phishing lures users to fake websites that pretend to be original and steals the user logging information (user name together with PIN or password). In 2010, in the USA alone, FBI estimated annual losses of 1 billion dollars that were caused by phishing. Consumer Reports projected over $2 billion lost to phishing scams [71].

- **Identity theft:** Apart from phishing, there are also other methods used by bots to steal confidential information from users. Examples include key loggers and traffic sniffers [67, 72]. Bots can be programmed to filter specific keywords of user sensitive information such as "username", "password", "bank", "paypal.com" or email contacts. In addition, bots are able to sniff the packets that the victim sends out over the Internet, therefore, revealing some important personal and financial information such as credit card details, license keys for games, music and software. Bots can also scan the victim's hard drive searching for files, or transfer files without the knowledge of the victim. The collected data is reported back to the botmaster who analyses and target specific data. The usable data will become a commodity to botmasters, often selling off data to third parties [20].

- **Click Fraudulence:** A botmaster can profit from making their bots periodically access specific links to artificially increase the number of clicks or to manipulate outcomes of online polls. A good example is the case of the AdSense program powered by Google. AdSense allows websites to display the Google advertisement and pays them money for the number of clicks on the commercial [73]. Google Click Quality and Security Team [74] reported that a bot called Clickbot.A controlled over 100,000 machines and excecuted a low-noise click fraud attack. According to a report of the IDC firm [75], in 2009 US companies paid a record $14,2 billion for keyword driven contextual ads. Google dominated with

55% of this revenue followed by Yahoo 9% and Microsoft 6%. The report argues that 42% of fraudulent clicks were generated from botnet infected machines.

2.3.3 Command and Control

Bots differs from other malware forms by their ability to establish a command and control (C&C) channel. The channel is then being used by a botmaster to update the bot software and to instruct the bots. In other words, while other malware solely execute a pre-defined set of instructions, a bot is also designed to accept instructions remotely from the botmaster. As the bot program propagates throughout the Internet, the collective group of bots form a botnet, providing the botmaster with an arbitrary large number of machines that are under the botmaster control. Using the C&C channel, the botmaster can instruct her bots to attack a specific target at a desired time. The botmaster can also use the C&C channel to upgrade her bots by letting the bots download and install updates so the bots are kept resistant against new antivirus software and other security countermeasures. There are several implementations of the C&C channels. They will be described later on.

2.3.4 Concealment

Concealment is a feature of a bot that allows it to be un-detected by its host and the host anti-virus software. Compared to other malware, bots apply more sophisticated hiding methods and therefore are harder to detect. Following are some main differences between bot and other malware concealment methods.

- Many viruses and Trojan horses can escape anti-virus detection, however, the designer of the malware may choose to reveal the presence of malware by displaying popup messages or ads. Some malware exhausts the victim computer memory or shuts down the computer or damages the host files to challenge the victim to remove the malware. In these cases, the intention of the attacker is to challenge the victim or to demonstrate to the victim his/her hacking skills. Some attackers may prefer to keep their malware hidden but thanks to relatively high efficiency of anti-virus software, even a user with a basic knowledge of computers is able to detect malware. Unlike other malware, bots rarely announce their presence. Instead, they infect hosts in by stealth so they can remain hidden.

- Once installed on a victim computer, viruses, worms and Trojan horses start working immediately. Bots, on the other hand, are inactive waiting for instructions from their botmasters. Some bots try to avoid detection by using slow-spreading infection techniques. Some use multiple levels of indirection (eg the botmaster does not contact the bots directly but through multiple layers of relay points) to make it harder to understand the botnet structure.

- The interesting feature of bots is their structure, which can evolve over time. The designers of bots equip them with enterprise-quality patching and update systems so bots are kept updated and resilient against detection or removal. As bots are able to download and install themselves, botmasters are able to update their malware so it can bypass new signatures, start a new spamming campaign, or new encryption key for communication. For example, if a detection tool successfully detects a bot based on their behaviour, the botmaster will upgrade her army with a new operation routine such as random active timing or changing some particular functionality to improve the bot conceal ability.

Summing up, bots are collections of malware blocks, where each block provides some specific functionalities. Consequently, capabilities of bots may vary significantly but still they preserve some common characteristics such as the ability to propagate, persistence and stealth. Bot can self-propagate like worm or spread via social engineering like Trojan horse. They can silently reside on the victim host by frequently changing their processes name with random active timing or hide their processes using rootkit technique to configure operating system. Most importantly, the unique remote access ability allows a bot to download and upgrade itself and to join the botmaster's army to simultaneously attack a target victim at a desirable time.

2.4 Botnet Architectures

A first step in the fight against bots is their detection. The next steps may be the removal of bot software, shutdown of infected by bots computers or hijack of control over bots. Clearly, designers of bots trying to make each of the step in the fight as difficult as possible by applying a variety of bot architectures. There are however three basic ones, namely centralized, decentralized and hybrid, each of which have a mixture of advantages and drawbacks [43, 76]. In this section, we are going to describe these structures, their strengths and weaknesses, which are summarized in Table 2.3. The following metrics are used to evaluate a botnet structure:

- **Complexity:** Efforts involved in designing, implementing and operating the botnet
- **Resiliency:** An ability to cope with a loss of members (zombies) or servers.
- **Latency:** Reliability in message transmission. A low latency indicates high speed transmission and network reliability.
- **Enumeration:** A capacity to accurately estimate a botnet size.
- **Re-sale:** A possibility to carve off sections of the botnet for lease or resale to other operators.

Architecture	Complexity	Resiliency	Latency	Enumeration	Re-sale
Star	Low	Low	Low	Easy	Hard
Multi-server	Medium	Better	Low (optimal)	Easy	Better
Hierarchical	Hard	High	High	Hard	Easy
P2P	Hard	High	High	Hard	Hard
Hybrid P2P	Very hard	Very high	High	Hard	Better
Random	Low	Very high	Very high	Very hard	Hard

Table 2.3: Properties of Botnet Architectures

2.4.1 Centralized Architecture

The centralized architecture consists of one or several selective hosts that are connected via appropriate C&C channels with bots. The C&C servers are usually the compromised computers as well and run certain network services such as IRC (IRC server) and HTTP (Web server). The majority of known botnets ultilized a centralized control structure such as SDBot, RBot, Rustock, Koobface, Zeus and so on. Centralized architecture can be represented in different shapes, often seen are star, multi-server and hierarchical.

Star Architecture

A star architecture applies a single centralized server that is used to communicate with all bot clients (see Figure 2.4). After installation, bot clients are programmed to connect to the single server (that uses a specific implementation of C&C channel - either based on IRC or HTTP protocols) to receive commands from the botmaster. The C&C server usually contains IP lists of all botnet members for the ease of query and communication with bots. All the “original” bots including SDBot, GTBot and Rbot employed this architecture.

Advantages :

1. low latency because each bot is issued commands directly from the single server.

2. easy to construct and maintain the botnet as the structure is simple.

Disadvantages :

1. single point of failure. The single control server is the Achilles’ heel of the centralized botnet. If the server is detected and blocked, the whole botnet is effectively neutralized. The C&C servers can be easily detected based on the incoming traffic from a large number of bots, or simply by the backward trace from a single captured bot.

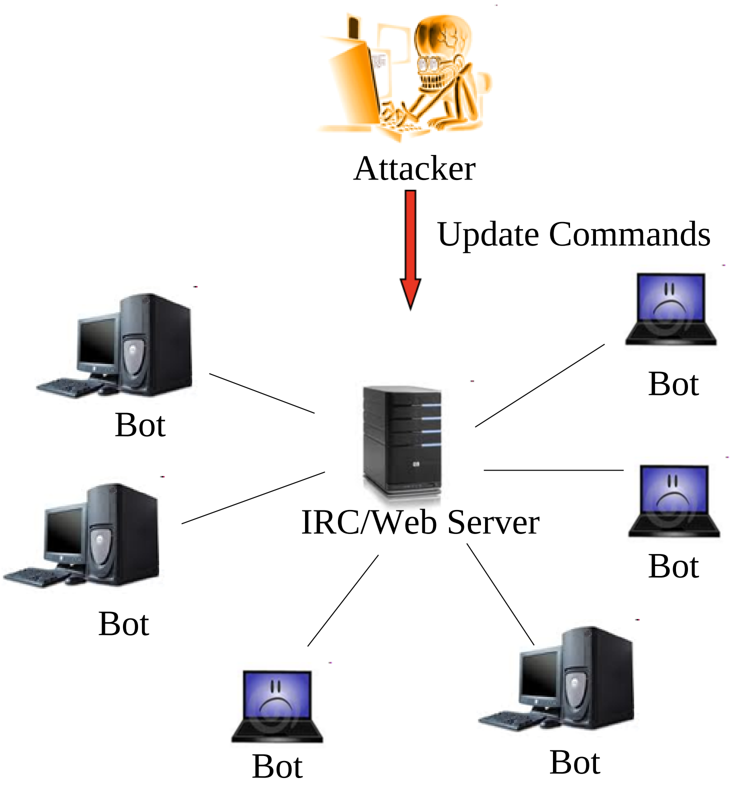


Figure 2.4: Star Architecture

2. very low resiliency. The IP lists of all bots contained in the server will reveal all bots' locations and make it easy to enumerate the number of bots in the botnet.

Multi-server Architecture

Multi-server architecture is a logical extension of the star architecture. Instead of one server, multiple servers are used to feed instructions to the individual bots as illustrated in Figure 2.5. The architecture of the Zeus botnet can serve as an example of multi-server architecture. Actually, the out of the box do-it-yourself Zeus botnet kit applies a simple star architecture. However, botmasters tend to upgrade Zeus so it becomes multi-server [76].

Advantages:

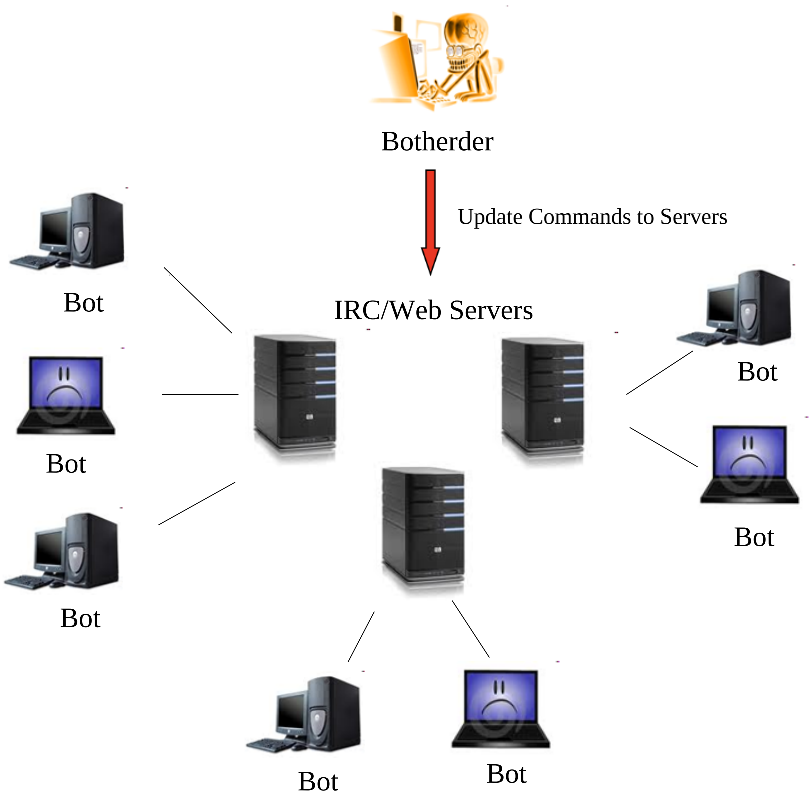


FIGURE 2.5: Multi-Server Architecture

1. better resiliency. Should an individual sever fail or be permanently removed, commands from the remaining servers maintain control of the botnet.

2. optimal latency. Wise distribution of the C&C servers in different geographical locations can speed up the transmission of commands and data. For example, bots are usually grouped according to areas or countries as communication among a C&C server with those bot clients within its neighbourhood is faster and more reliable than with those geographically far away from the server.

3. may be resistant to legal shutdown. C&C servers hosted in multiple countries can also make the botnet more resistant to legal shutdown requests.

Disadvantages:

1. high complexity. The main drawback is that it requires an advance planning

and more effort on the part of the botnet's operator to be able to construct and maintain the network.

2. easy to enumerate. As each C&C server maintains a list of IP addresses, it is easy for a security investigator to estimate the size of the botnet (or sub-botnet) once one or more C&C servers are analysed.

Hierarchical Architecture

For hierarchical botnets, the botmaster builds several layers of bots. The hierarchical configuration is shown in Figure 2.6. The botmaster does not communicate directly with the bots but communication passes through one or more bot controllers. In this case the communication network creates a tree with leaves (child nodes) and intermediate nodes (bot controllers). Bot controllers are also bots but are trusted by the botmaster. To be more secure, the botmaster also try to keep their bot controllers mobile by using dynamic DNS (DDNS) [1, 43]. By this way, bots are generally not aware of the server location as the bot controllers play as proxy nodes to hide the real locations of C&C servers. Likewise, no single bot agent is aware of the location of the entire botnet.

Advantages:

1. better resiliency thanks to multiple stages of proxy. Interception or hijacking of bot agents will not enumerate all members of a botnet and is unlikely to reveal the C&C server.

2. ease of re-sale. The botmaster can easily split a botnet into sub-botnets for sale or lease to other attackers.

Disadvantages:

1. high latency. Due to multiple layers, communication within a botnet suffers latency issues which make some real-time attacks difficult to launch such as DDoS.

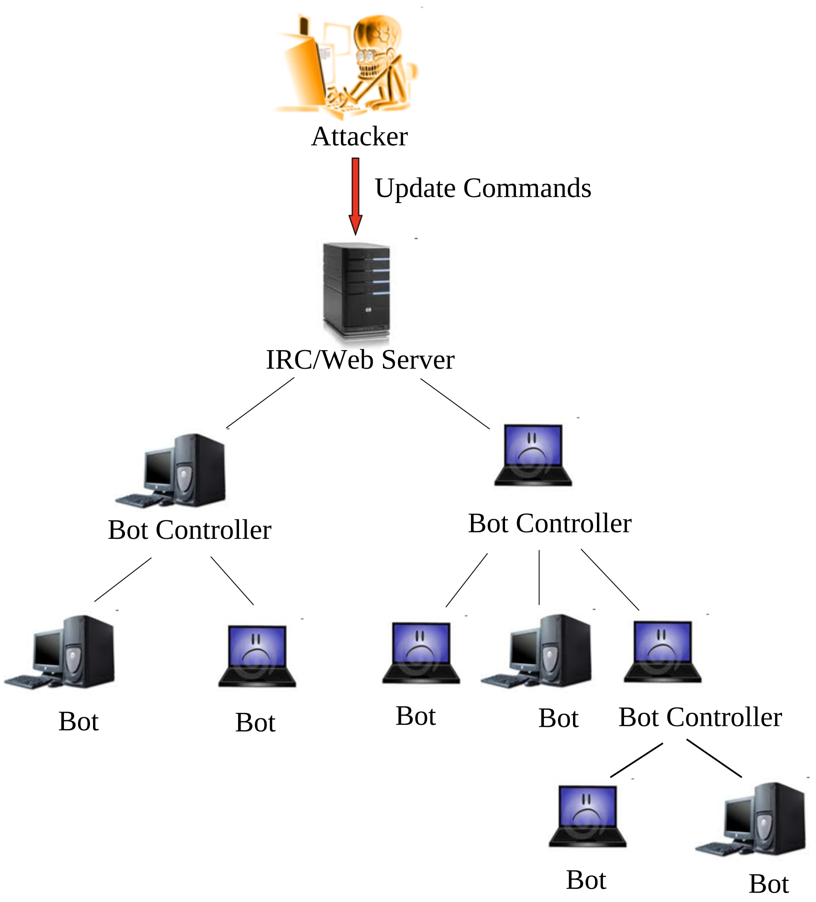


FIGURE 2.6: Hierarchical Architecture

2. high complexity. It is harder to construct and maintain a hierarchical botnet than a star one as it requires a good planning for each layer of the botnet.

The architecture of the Koobface botnets [77] is a good example of hierarchical, although the original design of Koobface employed a star architecture. The evolution toward hierarchical architecture was a defence against a single point of failure, which is the bot server in case of the star architecture.

2.4.2 Decentralized Architecture

Botnets with decentralized architecture (also known as P2P botnets), on the other hand, have no centralized C&C infrastructure. The decentralized architecture is illustrated in Figure 2.7. Each bot in the network can act as both a client and a server.

When acting as a client, a bot performs the attack against the victim computer. If the bot assumes the role of a server, it distributes messages to the client bots whose addresses are on its peer list. To issue a command to a P2P botnet, the botmaster will inject the command to several bots that she trusts. These bots will execute the command and pass it to all bots listed in its peer list and so on.

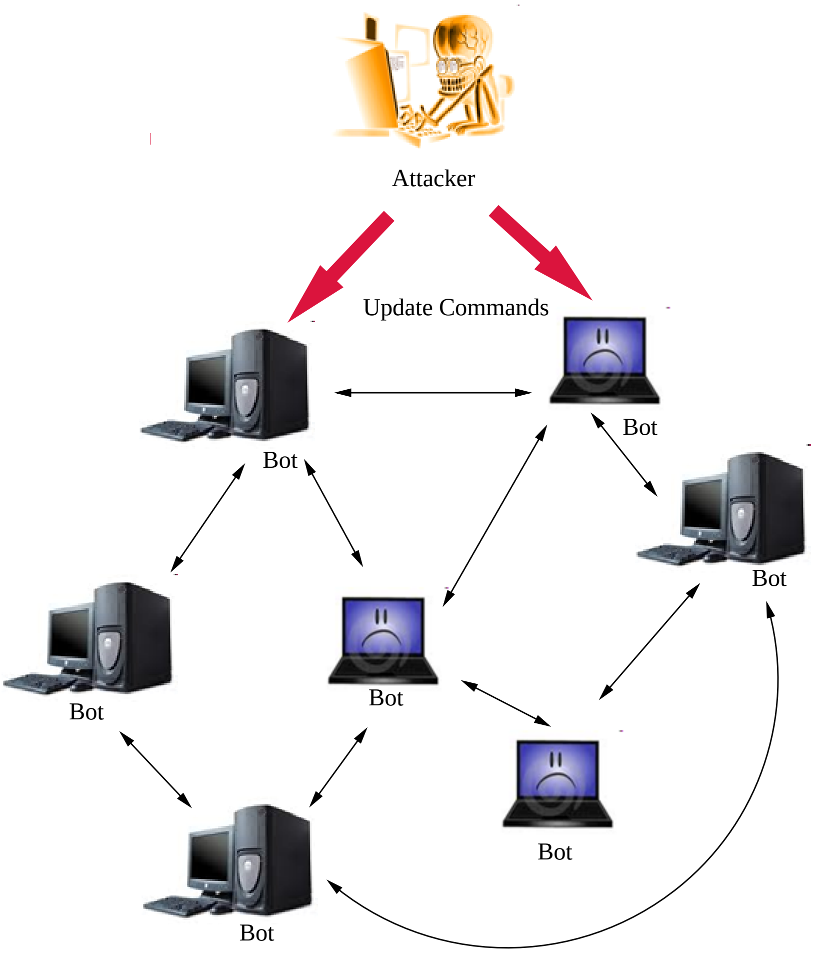


Figure 2.7: Decentralized Architecture

Advantages:

1. high resiliency. Due to the lack of centralized botnets and multiple communication paths between bot clients, the P2P botnets are highly resilient to shutdown and hijacking. Should a bot client is detected and monitored, it will only reveal

at most other N bots in its peer list.

2. hard to enumerate. It is often hard for security researchers to gage the overall size of a P2P botnet.

Disadvantage

1. high latency. Command latency is a problem for decentralized architecture botnets. The delivery of commands is not guaranteed and could be delayed. For example, if some bots are down or not available at the time of delivering commands, the botmaster may lose control of a significant part of the botnet, which cause difficulty for real-time activities.

2. very high complexity. P2P structure is very complicated and requires a lot of efforts to plan, implement and operate well the botnet.

Because of the above constraints, the botnets that use P2P based C&C are still very few, among those are Slapper and Nugache. In 2008, Arbor Networks estimated that known P2P botnets only comprised 5% of the current botnet population [78]. However, it is predicted that more and more botnets will move to P2P structure since it is more robust than centralized C&C communication [43, 79].

2.4.3 Hybrid Architecture

To surmount the limitations of the above architectures, botmasters may combine different architectures to increase survivability and sophistication of their attacks. Such architectures are called hybrid P2P architectures. Like the traditional architecture, the hybrid P2P one consists of multiple tiers of bots and each tier constitutes an instance of a P2P topology - see Figure 2.8 [80, 81].

The bottom nodes, also called workers or slave bots, are usually responsible for sending spam and other malicious activities. The upper nodes, also called servant nodes, create a P2P network and behave as both client and server bots. The servant nodes are also

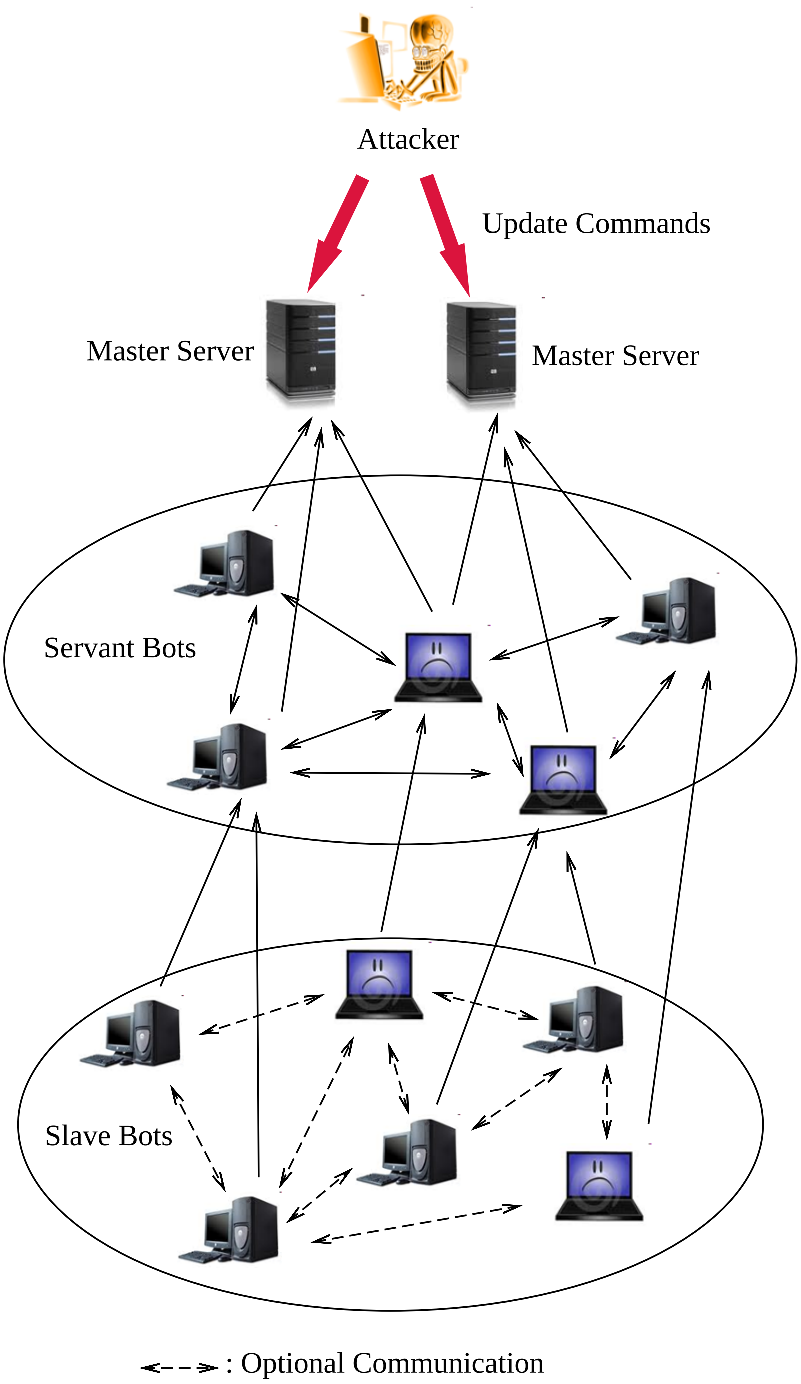


Figure 2.8: Hybrid P2P Architecture: slave bots also form a P2P network

known as proxy nodes as they form a proxy layer to obfuscate the addresses of the next tier hosts (usually master servers) and marshal communication between the next tier and the slave bots. To send commands to bots, the botmaster injects commands to master servers or some trusted servant bots. In turn, the servant bots can retrieve commands from upper tier nodes and forward the commands to other peers (servant bots) and slave bots. Once infected, a victim host can join the botnet as either a servant or slave bot. The hosts that are not able to access the Internet (because they are behind a firewall) become slave bots. A network of P2P slave bots may not necessarily exist. Usually, slave bots connect to one of the servant bots to get updates, in that case, the servant bot is a centralized server of one or many slave bots as illustrated in Figure 2.9. Only when a slave bot is not able to connect to the upper tier to get updates, it starts probing the local area network to search for peers and retrieve commands, hence form a P2P network as seen in Figure 2.8.

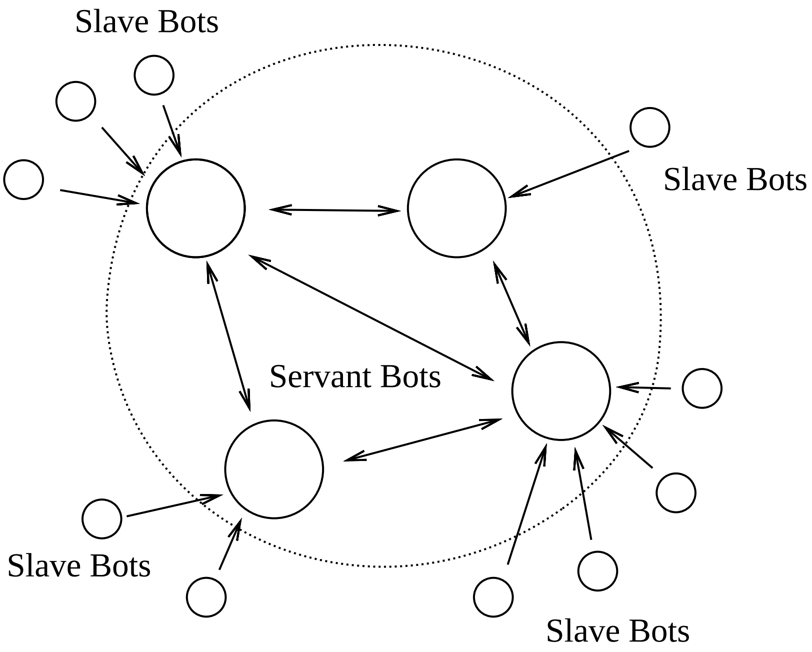


FIGURE 2.9: Hybrid P2P Architecture: each servant bot is a centralized control server

Advantages:

1. high resiliency. This architecture is very complicated with layers of proxy, therefore, increases the resilience ability.

2. hard to enumerate. It is hard for a security investigator to estimate the scale of the botnet.

Disadvantages:

1. very high complexity. It requires an advance planning as it may consist of many layers of sub botnets. It also requires a lot of efforts to construct and maintain the botnet.

2. very high latency. This botnet architecture suffers high latency issue since the communication must traverse multiple paths to get to destination.

Only a few botnets employ hybrid architecture due to its high complexity. Among those are infamous Storm and its descendant Waledac [80].

2.4.4 Random Architecture

An alternative to previous topologies is random architecture of C&C channels. In this model, the list of addresses of bots is not maintained and at a particular time, a single bot knows no more than one other bot [37, 43]. Unlike other topologies in which bots actively connect to other bots or a control server to get commands, in random architecture, a bot will listen to incoming connection from the botmaster or from another bot. To inject a command, the botmaster will scan the Internet to look for her bot. Once a bot is detected, the botmaster will issue a command to this bot. In turn, this bot will probe for another (only one) bot and send the received message.

Advantages:

1. high resiliency. Each bot only knows at most another bot at the time of probing so if a bot is detected and monitored, it will not reveal the whole botnet.

2. low complexity. It is quite easy to implement random architecture botnet as it does not require maintaining a structure.

Disadvantages:

1. very high latency. As the botmaster and each bot need to randomly scan for other bots, there is no guarantee that the message is delivered.

2. low scalability. This model is unlikely to be used in a large scale botnet because it has no structure which causes difficulty in coordinating a real-time attack.

There has no known botnet employing this architecture, however, future botnets would utilize this model to achieve high survivability.

2.5 Botnet Rallying Mechanism

Rallying mechanisms are methods for new bots to identify a rendezvous point, where they can retrieve commands from their botmaster. The most commonly rallying mechanisms are hard-coded IP addresses, dynamic DNS domain names and distributed DNS services [1, 43].

2.5.1 Hard-coded IP address

A hard-coded address is the simplest and most obvious method to rally bots. After a bot has infected a host, it connects to its C&C server using the server IP address. The address is hard coded in the bot files. This method can be easily and efficiently implemented. However, its main drawback is that the IP addresses can be easily detected and blocked by putting the address of the server bot on the blacklist. Once black listed, the bot becomes useless as it cannot communicate with the botmaster. Because of this drawback, modern bots do not employ hard-coded addresses.

2.5.2 Dynamic DNS Domain Name

Instead of hard-coded IP addresses, modern bots use hard-coded domain names assigned by a DNS provider. This allows the botmaster to relocate her botnet.

Assume that a C&C server was shutdown by putting its IP address on the blacklist. To resume control over bots, the botmaster creates a new C&C server bot, assigns its new IP address to the hard-coded domain name and asks DNS to update the appropriate entry. When a bot tries to connect to the old compromised C&C server and fails, it sends a query to DNS. DNS replies with the correct IP address. The bot can now establish communication with the new C&C server.

To make the detection of bots harder, botmasters may randomly update the DNS entries to periodically change locations of the C&C servers. It is also easy to obtain a dynamic domain name. For example, "dyndns.com" provide a free service to create your own domain "yourname.dynds.com" and assign a dynamic IP to this name [82].

2.5.3 Distributed DNS service

The new and sophisticated mechanisms for rallying apply distributed DNS service. This solution can be found in a new generation of bots. The bots run their own DNS servers in carefully chosen places usually beyond the reach of the law enforcement authorities. For example, DNS server can be located in a country, whose law has a very narrow definition of computer crime. Bots contact their own DNS servers to retrieve the IP addresses of the C&C servers. Botnets utilizing this mechanism are hardest to detect and remove.

2.6 Botnet Communication Protocols

Communication protocol is a standard that defines the syntax and structure of data exchanged between two entities. Bots creating a botnet exchange information using

C&C channels. The designers of botnets usually do not design their own protocols for C&C channels. They tend to use existing communication protocols such as IRC, HTTP and P2P. Although the functionality of bots usually changes over time, the communication protocol is rarely modified. Thus, it is convenient to classify botnets according to the communication protocol they use. So we have IRC botnets, HTTP botnets and P2P botnets. Understanding how the communication protocol works is crucial for understanding internal workings of botnets. In particular, the following aspects seem to be important

- the protocol characteristics provide information about the botnet origin and about software tools that are being used by bots,
- the monitoring of a specific communication protocol may provide information about bots and their master. This may include information about locations of the C&C servers and bots.

Apart from the three main communications mentioned above, there are also other alternatives such as the Instant Messenger (IM) protocol, the voice over IP (VoIP) protocol or one of the communication protocols from the DNS protocol suit. However, the population of botnets using such protocols is relatively small.

2.6.1 Botnets with C&C based on IRC protocol

The Internet Relay Chat (IRC) protocol was used in the first generation of botnets. Although there are many bots that use other communication protocols, the overwhelming majority of current botnets uses the IRC protocol. The popularity of IRC can be attributed to the following characteristics of the protocol [1, 43]:

- It is easy for implementation. Building an IRC server is pretty straightforward. Free tools and tutorials are readily available via Internet, just type keyword to your favourite search engine to get an extensive list of IRC resources. Alternatively, one can use one of many existing IRC servers to register and run a server. A list of such IRC servers can be found in [83].

- It is interactive. IRC provides a very responsive communication channel between clients and servers.

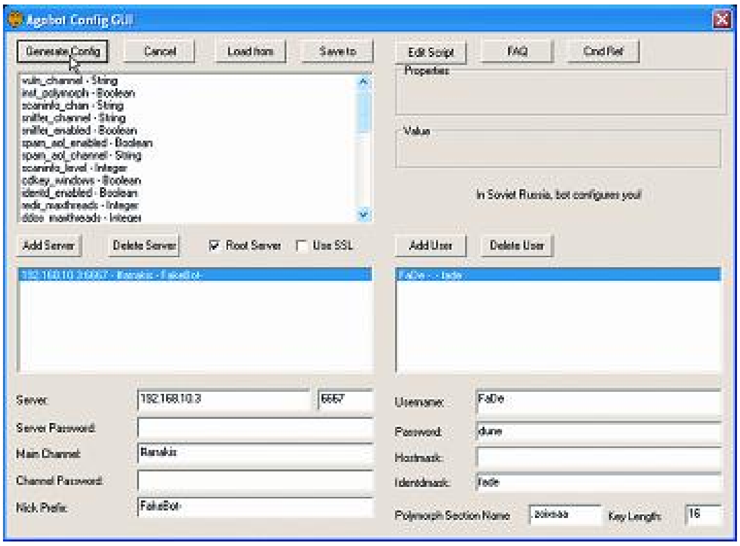
- It is flexible. The IRC protocol provides not only point-to-point communication but also broadcasting and multicasting options. This is very convenient for a botmaster who may send a command to all bots (via broadcast channel) or target a group of bots (via multicast channel).

- It provides anonymity. Bots are able to hide their identities by using anonymous proxies or/and fake IP addresses. Due to this, it is difficult to pinpoint the source of bot attacks.

- It is centralised. This characteristic can be seen as an advantage but also as a weakness. On the positive side, it offers a botmaster the ability to control a large number of bots from a single server. Moreover, a botmaster may use the same server to control many independent botnets. On the negative side, if an IRC server fails then all bots controlled by it become inactive (see Section 2.2).

A botmaster creates a private communication channel. Access to the channel is normally via password. The channel is going to be used by all bots to receive instructions from the botmaster. To provide confidentiality, data sent via the channel can be encrypted using one of the SSL encryption algorithms. Additionally, bots usually contain scripts that obfuscate message sent. For example, a message “Hello Alice 154.102.8.10” after parsing becomes “ddos syn 154.102.8.10”. This can be translated as “ddos attack target 154.102.8.10 using SYN flood ”.

Let’s examine a sample attack scenario of Agobot [84]. The first step is to configure the bot (see Figure 2.10). The information entered included name and port of the IRC server, name of the channel, a list of users with master passwords, and finally - filename and directory in which the bot is to be installed. Plugins have also been activated such as sniffing support and polymorphic engine. The result of this stage is a `config.h` file, fundamental for bot compilation.



```
vuln_channel - String
inst_polymorph - Boolean
scaninfo_chan - String
sniffer_channel - String
sniffer_enabled - Boolean
spam_aol_enabled - Boolean
spam_aol_channel - String
scaninfo_level - Integer
cdkey_windows - Boolean
identd_enabled - Boolean
redir_maxthreads - Integer
ddos_maxthreads - Integer
```

☑ Root Server ☐ Use SSL

192.168.10.3:6667 - #arrakis - FakeBot-

FaDe - - fade

Server: 192.168.10.3 6667

Server Password:

Main Channel: #arrakis

Channel Password:

Nick Prefix: FakeBot-

Username: FaDe

Password: dune

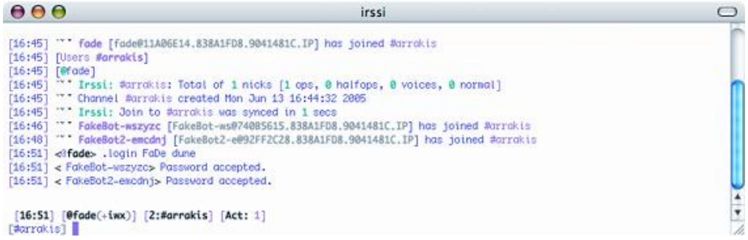
Hostmask:

Identdmask: fade

Polymorph Section Name: .zoixaa Key Length: 16

FIGURE 2.10: Agobot configuration interface

Once the bot has been compiled, the host has been infected and tried to make communication with the IRC server. In order to gain control over the bots, authentication is needed. This is done by simply sending a command to the channel (see Figure 2.11). Once authentication is completed, the botmaster can control the bot by issuing different commands over the IRC channels. Some typical commands are listed in Table 2.4 [84, 85] . In the examples, Ago is the botmaster and AgobotN is an instance of Agobot.



```
[16:45] -!- fade [fade@11A06E14.838A1FD8.9041481C.IP] has joined #arrakis
[16:45] [Users #arrakis]
[16:45] [@fade]
[16:45] -!- Irssi: #arrakis: Total of 1 nicks [1 ops, 0 halfops, 0 voices, 0 normal]
[16:45] -!- Channel #arrakis created Mon Jun 13 16:44:32 2005
[16:45] -!- Irssi: Join to #arrakis was synced in 1 secs
[16:46] -!- FakeBot-wszyzc [FakeBot-ws@74085615.838A1FD8.9041481C.IP] has joined #arrakis
[16:48] -!- FakeBot2-emcdnj [FakeBot2-e@92FF2C28.838A1FD8.9041481C.IP] has joined #arrakis
[16:51] <@fade> .login FaDe dune
[16:51] < FakeBot-wszyzc> Password accepted.
[16:51] < FakeBot2-emcdnj> Password accepted.

 [16:51] [@fade(+iwx)] [2:#arrakis] [Act: 1]
[#arrakis] 
```

FIGURE 2.11: Agobot username and password authentication

Summing up, the IRC protocol provides a simple and efficient communication tool to control thousands of bots. It has, however, weaknesses that in some circumstances

Command	**Description**	**Example**

`command.list`	List of all the available commands	`<Ago> .commands.list`
		`<AgobotN> -[ command list ]-`
		`<AgobotN> 1. / ‘‘commands.list" / ‘‘Lists all available commands"`
		`<AgobotN> 2. / ‘‘cvar.list" / ‘‘prints a list of all cvars"`

`bot.dns`	Resolves an IP/hostname	`<Ago> .bot.dns ago.bastart.net`
		`<AgobotN> ago.bastart.net -> 90.0.1.55`
		`<Ago> .bot.dns 90.0.1.55`
		`<AgobotN> 90.0.1.55 -> ago.bastart.net`

`bot.execute`	Runs an .exe file on a remote computer	`<Ago> .bot.execute 1 notepad.exe` (Victim executes notepad.exe visible)

`bot.about`	Displays the info the author wants you to see	`<Ago> .bot.about`
		`<AgobotN> AgobotN (0.1.N Alpha) "Release" on "Win32" by Ago (theago@gmx.net). homepage: http://none.yet/`

`bot.id`	Displays the bot id which is used to identify which version is running, and only update the bots that need it during an update.	`<Ago> .bot.id`
		`<AgobotN> DC0M0R17`

`bot.remove`	Completely removes the bot from the system.	`<Ago> .bot.remove`
		`<Agobot3> removing bot...`
		`<-- Agobot3 has quit (Read error: 104 (Connection reset by peer))`

Table 2.4: Some Agobot C&C commands

outweigh the benefits and forcing botmasters to explore alternative protocols. Its main drawback seems to be a single point of failure as discussed in Section 2.2. Other weaknesses include the ease, in which the communication can be eavesdropped. This also means that a command (a message) from the botmaster can be intercepted or/and modified. Consequently, bots can be detected, hijacked or immobilised.

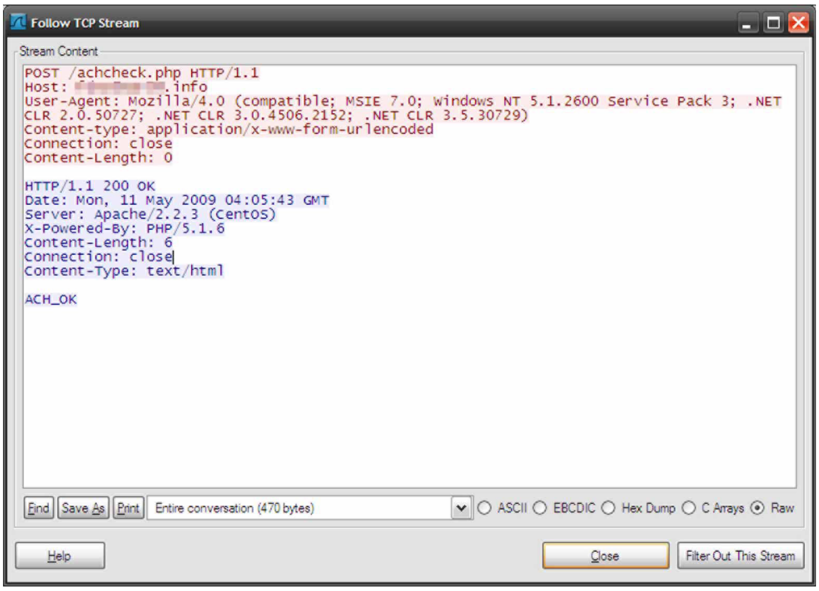
As a security precaution, many corporate networks do not allow any IRC traffic. Also many firewalls are configured to block any IRC traffic. To circumvent these precautions, IRC bots tunnel their communication via the HTTP protocol. Because of the wide use and popularity of the HTTP protocol, the IRC traffic is much harder to detect.

2.6.2 Botnets with C&C based on HTTP Protocol

The HTTP protocol is an application-layer network protocol built on top of the TCP protocol. This protocol is the second choice of bot designers. HTTP provides a standard for web browsers and servers to communicate. Note that a web server is now used to facilitate C&C communication channels. Now we describe the C&C communication for HTTP-based (aka Web-based) botnets.

Web-based bots are programmed to instruct the compromised machines to connect to the web C&C server providing the machine IP address, port, and machine identification. The botmaster applies a web interface to send commands to an individual bot or to the whole botnet by using appropriate HTTP commands such as GET and POST. One example of Web-based botnets is the well-known Koobface botnet. The Koobface C&C and infected nodes communicate with each other through a series of HTTP GET and POST transactions. These transactions contain either C&C commands or data stolen by the malware components from infected nodes. Most of the transactions are not encrypted or written in plain text. However, in information-stealing transactions, a simple encryption method is utilized. Koobface first use a predefined list of C&C domains embedded within the malware file and check if there is C&C available. To check if the C&C is available, Koobface issues an HTTP POST request to the file `achcheck.php`. A reply `ACH_OK` as shown in Figure 2.12 signifies that the C&C domain is available. Once an available Koobface C&C domain is found, an HTTP request to either `first.php` or `gen.php` is issued. This transaction reports the following information to the Koobface C&C:

- the infected machine volume serial number,
- what Koobface component is reporting to the C&C,
- the version number of the Koobface component,
- the social networking site cookies found on the infected machine.



```
POST /achcheck.php HTTP/1.1
Host: [redacted].info
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1.2600 Service Pack 3; .NET
CLR 2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729)
Content-type: application/x-www-form-urlencoded
Connection: close
Content-Length: 0

HTTP/1.1 200 OK
Date: Mon, 11 May 2009 04:05:43 GMT
Server: Apache/2.2.3 (CentOS)
X-Powered-By: PHP/5.1.6
Content-Length: 6
Connection: close
Content-Type: text/html

ACH_OK
```

Entire conversation (470 bytes) ○ ASCII ○ EBCDIC ○ Hex Dump ○ C Arrays ● Raw

Figure 2.12: `achcheck.php` transaction packet capture

Depending on the Koobface component connecting to the C&C, the component version, or the social networking site cookies found in the system, the C&C can either instruct its component to:

- Download additional components.
- Download updated components.

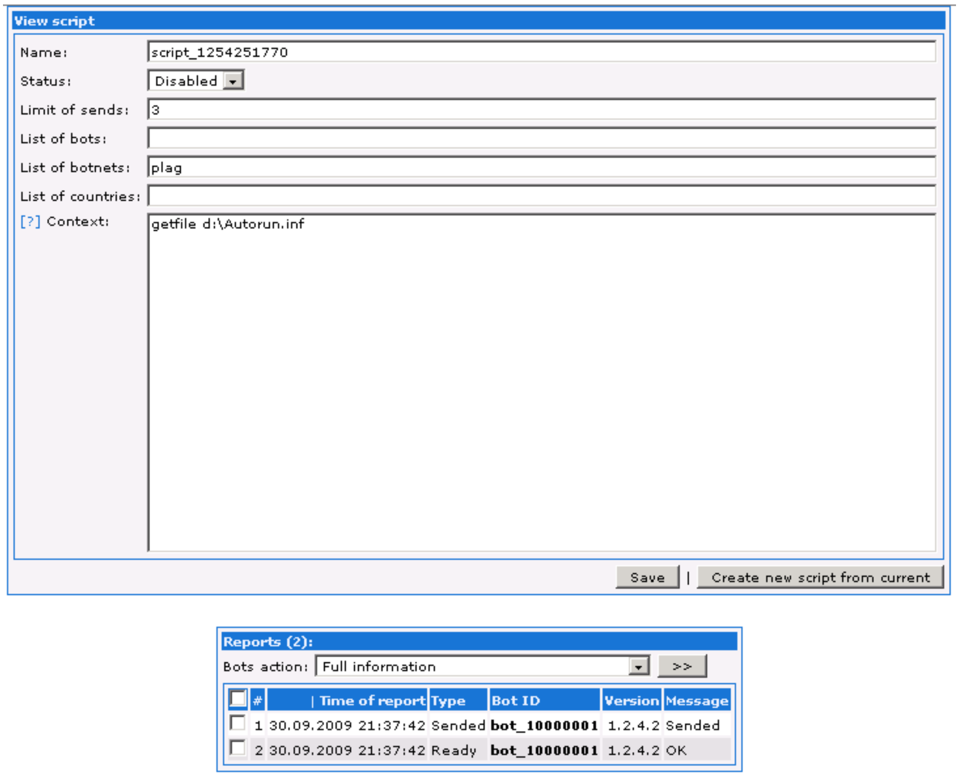
- Perform social networking site propagation.

Table 2.5 [77] lists some C&C commands from the Koobface C&C server.

Command	**Description**	**Example**
`UPDATE`	Download and execute an updated version of the loader component.	`UPDATE http://some.octopus.com/1/ld.12.exe`
`START`	Download and execute the file specified in the URL argument.	`START http://some.domain.com/1/CAPTCHA6.exe`
`WAIT`	Wait for N minutes before executing the file.	`WAIT 5`
`BLOCKIP`	Block access to websites hosted in an IP range defined by the C&C. This routes all traffic bound to the blackholed IP address range back to the victims PC.	`BLOCKIP 92.122.0.0`
`EXIT`	Terminate the current process or program.	
`RESET`	Ignores all the other commands it received and retrieves a new set of commands from the C&C.	

TABLE 2.5: Some Koobface C&C commands

The HTTP protocol is interactive, flexible and suits well the centralised control model. Moreover, HTTP bots are easier to operate since botmasters can control botnets by simply using the point-and-click interface of web browsers. Figure 2.13 shows an example of the command and control of graphical user interface (GUI) of the Zeus botnet. In addition, it is much more difficult to detect botnets that use HTTP for communication because the traffic usually hides amidst huge amounts of normal HTTP traffic, hence usually bypasses firewall. As a result, the centralised botnet technology is adopting the HTTP protocol as a preferred C&C communication tool (see Figure 2.3 in Section 2.2).



View script

Name:	script_1254251770
Status:	Disabled
Limit of sends:	3
List of bots:	
List of botnets:	plag
List of countries:	
[?] Context:	getfile d:\Autorun.inf

Reports (2):

Bots action: Full information

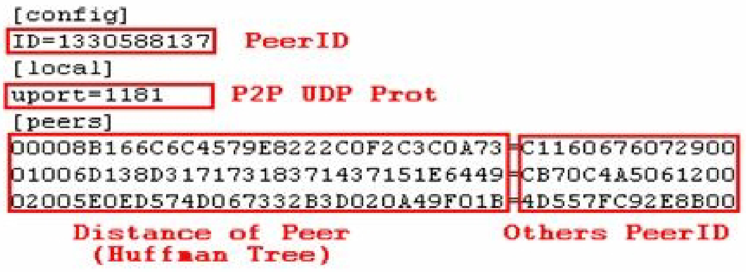
□	#	Time of report	Type	Bot ID	Version	Message
□	1	30.09.2009 21:37:42	Sended	**bot_10000001**	1.2.4.2	Sended
□	2	30.09.2009 21:37:42	Ready	**bot_10000001**	1.2.4.2	OK

Figure 2.13: C&C GUI: Zeus Control Panel - Script Edit

2.6.3 Botnets with C&C based on P2P Protocol

Unlike the traditional client/server model, peer-to-peer (P2P) networks consist of nodes or members (peers) who have the same privileges [86]. The P2P protocol provides a suit of tools that allow a peer to communicate with any other peer in the network. In other words, each peer in the network can assume the role either client or server. Note that there is no central authority in the P2P network. Therefore, the P2P protocol comes with decentralized architecture, which also called P2P architecture.

Observe that P2P botnets do not have any central C&C server. However, P2P bots can contact other bots whose addresses are stored in their peer lists. A botmaster can contact her bots by injecting appropriate commands to one (or more) of the (peer) bots. The commands are next forwarded to other bots using the P2P protocol. To avoid detection, the commands may also be obfuscated.

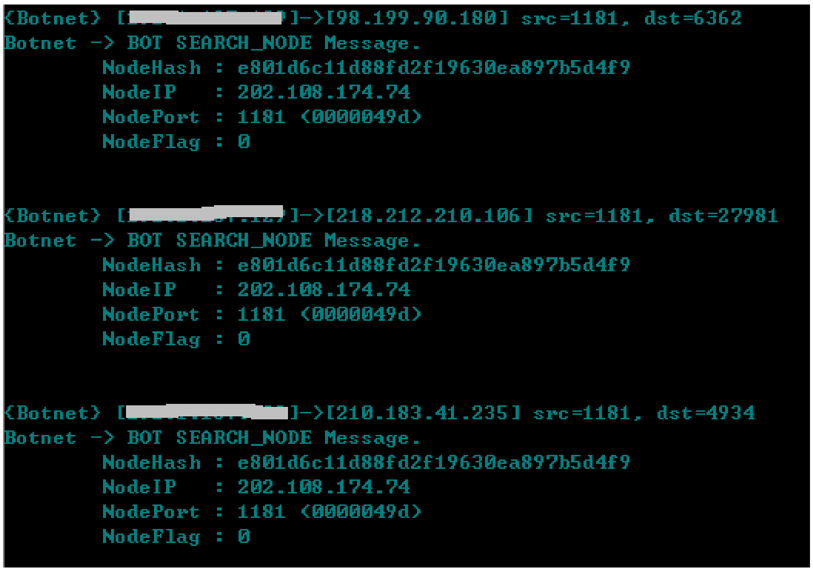


```
[config]
ID=1330588137
[local]
uport=1181
[peers]
00008B166C6C4579E8222C0F2C3C0A73=C1160676072900
01006D138D31717318371437151E6449=CB70C4A5061200
02005E0ED574D067332B3D020A49F01B=4D557FC92E8B00
```

FIGURE 2.14: Example of a bot information in a peer list

One typical example of P2P botnets is the Storm botnet. After infecting the host, the bot starts contacting other peers to retreive commands. There is a peer list in the bot's configuration file for them to contact their peers residing in the botnet. Figure 2.14 [87] shows one example of a peer list. Based on this peer list, the bot searches for other peers as shown in Figure 2.15 [87] and receives reply from the peers as shown in Figure 2.16 [87]. All the exchanged messages and information are encrypted to bypass bot detection tools.

Based on the commands received, the bot can exchange its peer list as shown in Figure 2.17, other information such as SMTP servers and spam/adware contents. It also tries

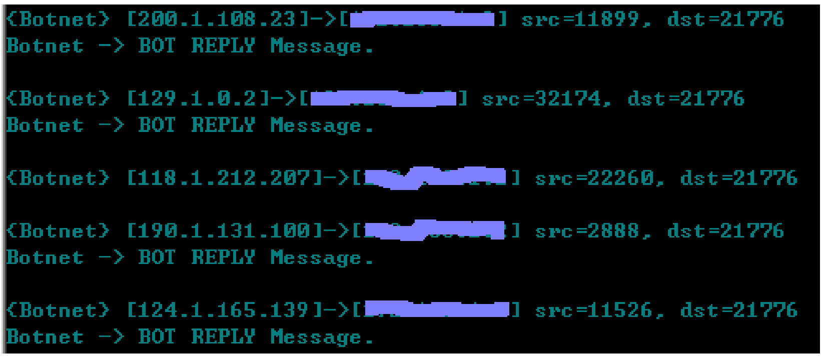


```
<Botnet> [[redacted]]->[98.199.90.180] src=1181, dst=6362
Botnet -> BOT SEARCH_NODE Message.
        NodeHash : e801d6c11d88fd2f19630ea897b5d4f9
        NodeIP   : 202.108.174.74
        NodePort : 1181 (0000049d)
        NodeFlag : 0


<Botnet> [[redacted]]->[218.212.210.106] src=1181, dst=27981
Botnet -> BOT SEARCH_NODE Message.
        NodeHash : e801d6c11d88fd2f19630ea897b5d4f9
        NodeIP   : 202.108.174.74
        NodePort : 1181 (0000049d)
        NodeFlag : 0


<Botnet> [[redacted]]->[210.183.41.235] src=1181, dst=4934
Botnet -> BOT SEARCH_NODE Message.
        NodeHash : e801d6c11d88fd2f19630ea897b5d4f9
        NodeIP   : 202.108.174.74
        NodePort : 1181 (0000049d)
        NodeFlag : 0
```

Figure 2.15: Searching other (peer) bots

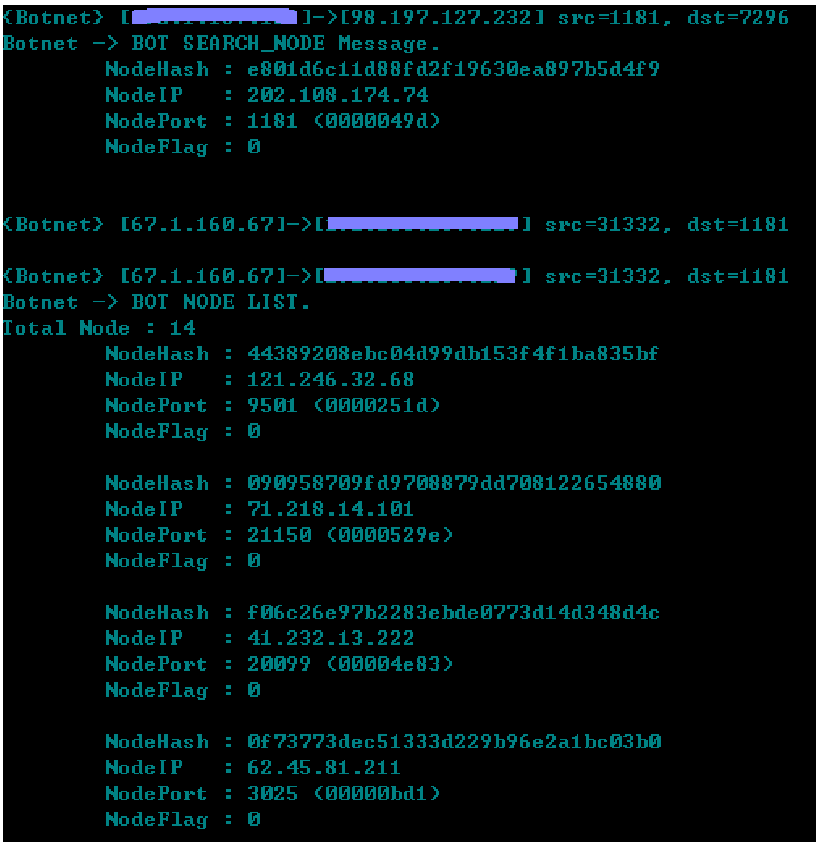
to connect to some SMTP servers to send spam. Figure 2.18 shows the bot's attempts to send spam via Google SMTP [87].

P2P botnets can utilize established P2P protocols such as WASTE (Phatbot [88, 89]) and Overnet (Storm [49, 90, 91]) or implement their own P2P protocol such as Nugache [91]. Comparing to IRC and HTTP, the P2P protocol is more complicated and requires



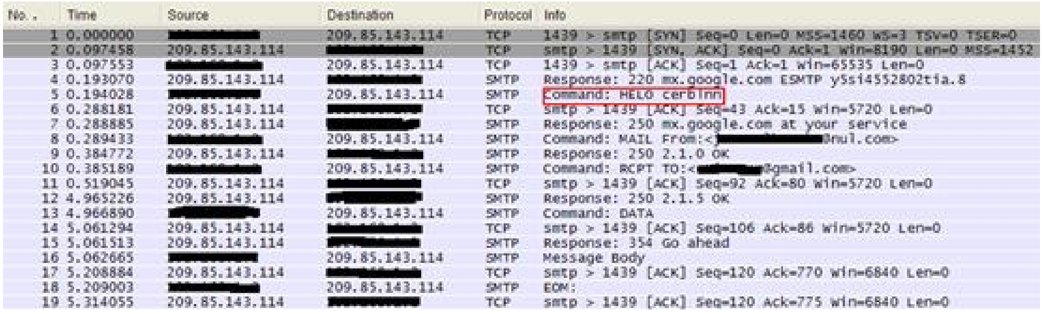
```
<Botnet> [200.1.108.23]->[[redacted]] src=11899, dst=21776
Botnet -> BOT REPLY Message.

<Botnet> [129.1.0.2]->[[redacted]] src=32174, dst=21776
Botnet -> BOT REPLY Message.

<Botnet> [118.1.212.207]->[[redacted]] src=22260, dst=21776

<Botnet> [190.1.131.100]->[[redacted]] src=2888, dst=21776
Botnet -> BOT REPLY Message.

<Botnet> [124.1.165.139]->[[redacted]] src=11526, dst=21776
Botnet -> BOT REPLY Message.
```

Figure 2.16: Reply messages from other (peer) bots



```
<Botnet> [[redacted]]->[98.197.127.232] src=1181, dst=7296
Botnet -> BOT SEARCH_NODE Message.
        NodeHash : e801d6c11d88fd2f19630ea897b5d4f9
        NodeIP   : 202.108.174.74
        NodePort : 1181 (0000049d)
        NodeFlag : 0


<Botnet> [67.1.160.67]->[[redacted]] src=31332, dst=1181

<Botnet> [67.1.160.67]->[[redacted]] src=31332, dst=1181
Botnet -> BOT NODE LIST.
Total Node : 14
        NodeHash : 44389208ebc04d99db153f4f1ba835bf
        NodeIP   : 121.246.32.68
        NodePort : 9501 (0000251d)
        NodeFlag : 0

        NodeHash : 090958709fd9708879dd708122654880
        NodeIP   : 71.218.14.101
        NodePort : 21150 (0000529e)
        NodeFlag : 0

        NodeHash : f06c26e97b2283ebde0773d14d348d4c
        NodeIP   : 41.232.13.222
        NodePort : 20099 (00004e83)
        NodeFlag : 0

        NodeHash : 0f73773dec51333d229b96e2a1bc03b0
        NodeIP   : 62.45.81.211
        NodePort : 3025 (00000bd1)
        NodeFlag : 0
```

FIGURE 2.17: Peer lists exchange



No.	Time	Source	Destination	Protocol	Info
1	0.000000	[redacted]	209.85.143.114	TCP	1439 > smtp [SYN] Seq=0 Len=0 MSS=1460 WS=3 TSV=0 TSER=0
2	0.097458	209.85.143.114	[redacted]	TCP	smtp > 1439 [SYN, ACK] Seq=0 Ack=1 Win=8190 Len=0 MSS=1452
3	0.097553	[redacted]	209.85.143.114	TCP	1439 > smtp [ACK] Seq=1 Ack=1 Win=65535 Len=0
4	0.193070	209.85.143.114	[redacted]	SMTP	Response: 220 mx.google.com ESMTP y5si4552802tia.8
5	0.194028	[redacted]	209.85.143.114	SMTP	Command: HELO cerbinn
6	0.288181	209.85.143.114	[redacted]	TCP	smtp > 1439 [ACK] Seq=43 Ack=15 Win=5720 Len=0
7	0.288885	209.85.143.114	[redacted]	SMTP	Response: 250 mx.google.com at your service
8	0.289433	[redacted]	209.85.143.114	SMTP	Command: MAIL From:<[redacted]@nul.com>
9	0.384772	209.85.143.114	[redacted]	SMTP	Response: 250 2.1.0 OK
10	0.385189	[redacted]	209.85.143.114	SMTP	Command: RCPT TO:<[redacted]@gmail.com>
11	0.519045	209.85.143.114	[redacted]	TCP	smtp > 1439 [ACK] Seq=92 Ack=80 Win=5720 Len=0
12	4.965226	209.85.143.114	[redacted]	SMTP	Response: 250 2.1.5 OK
13	4.966890	[redacted]	209.85.143.114	SMTP	Command: DATA
14	5.061294	209.85.143.114	[redacted]	TCP	smtp > 1439 [ACK] Seq=106 Ack=86 Win=5720 Len=0
15	5.061513	209.85.143.114	[redacted]	SMTP	Response: 354 Go ahead
16	5.062665	[redacted]	209.85.143.114	SMTP	Message Body
17	5.208884	209.85.143.114	[redacted]	TCP	smtp > 1439 [ACK] Seq=120 Ack=770 Win=6840 Len=0
18	5.209003	[redacted]	209.85.143.114	SMTP	EOM:
19	5.314055	209.85.143.114	[redacted]	TCP	smtp > 1439 [ACK] Seq=120 Ack=775 Win=6840 Len=0

FIGURE 2.18: Send spam through Google's SMTP servers

more efforts to effectively plan the nodes in the network. In addition, the P2P protocol works better for small groups of bots, therefore, large scale botnets rarely employ this protocol. Motivated by financial gains, botmasters tend to recruit as many bots as possible. As a consequence, centralized architecture with the IRC and HTTP protocols are more popular over the P2P protocol. However, recent advances in anti-bot software may force botmasters to move towards the P2P protocol or alternatively, try other new emerging protocols to increase survivability of their botnets.

2.7 Evasion Tactics

Majority of botnets are designed with a specific intend of gaining a financial reward. Botnets are an industry providing tools for rapidly growing cybercrime groups. Clearly, botmasters try to stay undetected and anymous. To achieve this goal, they apply a variety of evasion tactics. In general, the evasion tactics can be divided into the following two categories:

- hiding the presence of botnets from anti-virus and intrusion detection systems,
- concealing traces of communication between bots.

2.7.1 Hiding presence techniques

Botnets become more and more sophisticated by day as their designers try to be one step ahead of the anti-bot technology. There is a variety of techniques that are used to evade anti-virus and IDS systems. The following list provides the main ones.

- **Rootkit:** A rootkit [55, 56] is a collection of tools that is used by the attacker to gain administrator access privileges on a host. Many bots use rootkits to hide themselves by interfering with operating system. Some bots attempt to disable anti-virus software. Storm and TLD4 are good examples of such bots [49, 54].
- **Low presence:** To avoid detection in a large networks, botmasters may activate a small part of a botnet that generates relatively small traffic. Smaller botnet

traffic is less visible and more difficult to detect. Storm, TLD4 and Waledac [54, 80] are examples of botnets that apply this tactic.

- **Random Connection Interval:** Detection tools use specific behavioural characteristics of bots to detect them. One of such characteristics is a measure of connection persistence from bot to C&C server. Such a measure can be a length of connection interval. If the connection interval of bots is set to be random or very long, then this characteristic fails to be an effective discriminator for detection. For example, the connection interval of the Srizbi bot is random in the interval from 60 to 1200 secounds. The Bredolab bot has a long interval equal to 20 minutes [92].

2.7.2 Hiding Traces of Bot Communication

Hiding traces of bot communication is essential for a bot to survive when it is being tracked. Otherwise, the communication traces may reveal locations of C&C servers and in the extreme case even the location of the botmaster. The following list provides a non-exhaustive list of evasion mechanisms that are employed by bots.

- **Fast-flux network:** The goal of fast-flux technique is for a domain name to have multiple IP addresses assigned to it. These IP addresses are swapped in and out of flux with extreme frequency, using a combination of round-robin IP addresses and a very short Time-To-Live (TTL) for any given particular DNS Resource Record (RR). Website hostnames may be associated with a new set of IP addresses as often as every 3 minutes [93].

  Bot designers quickly adapt this technique to hide bot trace of communication. The attacker's fast-flux networks are a group of compromised (hacked) computer systems which have public DNS records [92]. These records change very fast (by setting very small value for TTL), which makes the detection of those networks harder. The fast-flux networks, which are assigned tens or hundreds of IP addresses, play as proxy for phishing and malware delivery sites of botnets. When using fast-flux technique, the bot communication trace will lead security investigators to IPs of the proxy hosts, not the real IP address of a C&C server. Storm and its variants are typical examples of botnets using this technique [80].

- **Data Encryption:** The communication between bots and their C&C servers contains sensitive information about the botnet. To protect confidentiality of this information, many bots use encryption. Bots can apply the existing encryption algorithms or design their own encryption scheme. For example, the Karen botnet encrypts communication using its own algorithm while the Waledac bot employs AES, RSA and Base64 encryption algorithms [80, 94].

- **C&C Server Migration:** Modern botnets employ multiple C&C servers to make sure that botnet is operational even after few C&C servers have been detected and shutdown. Instead of a fixed C&C server, a bot chooses at random one of available C&C servers for each connection. This makes it difficult to identify the collection of C&C servers that are communicating with the bot. Botnets implement the C&C server migration using a hard-coded IP address or DDNS techniques (see Section 2.5).

- **Noise traffic:** To evade detection, bots can inject noise packets to their communication. The idea behind this technique is to make the traffic between bots and their C&C servers very similar to the normal web traffic. This technique is widely used by botnets for sending spam. For instance, the Rustock bot [70] applies this technique.

- **Using legitimate services:** There have been some attempts to use legitimate and popular Web 2.0 services such as Twitter and Google blog for delivering commands in botnets [23–26]. This works as follows. The botmaster sends a command by posting an appropriate message to a new entry of Twitter. Alerted by the Twitter RSS, bots access the website and retrieve the message. Next they parse it, recover the command and execute it.

This arrangement makes the detection of communication between bots and their C&C servers very difficult. Fortunately, this technique is applied by a small number of bots. But it is expected that the popularity of the technique is going to grow.

2.8 Summary

In this chapter, we have provided an in-depth understanding of botnet natures from their life cycle to their characteristics, architectures and survival tactics. We have also compared bot and other types of malware to be able to distinguish bot activities from others. The main characteristics of bot and botnet can be summarized as follows:

- The unique characteristic of the botnet is that it is remotely controlled by different mechanisms including IRC, P2P and HTTP protocols. Though the most common C&C structure is IRC, it has been observed a shift to P2P and HTTP protocols for better resilience. Using the C&C channel, the botmaster can instruct her bots to conduct malicious activities as well as upgrade her bots to keep them resistant against new security countermeasures.

- Botnet can be constructed in different architectures, each has its own advantages and drawbacks. Three main structures are centralized, decentralized and hybrid P2P architectures. Though centralized has low resilience issue, it is still the most popular architecture used thanks to its advantages such as large scale, easy to maintain and low latency.

- Bots use variety methods to spread themselves. Some examples are exploiting vulnerabilities, using spam containing infected files and phishing links to malicious websites and recently, using social networking to propagate. With the dominant of social networking services, propagating via social networking is expected to rise.

- Bots malicious activities are for financial gains. Bots can be traded to conduct DDoS attacks or used to send advertising and phishing spam or instructed to steal personal information such as bank account credentials and credit card information. The stolen information is used to steal money or sold to third parties.

- Motivated by financial rewards, botmasters apply variety evasion techniques to erase traces of botnet communication and hide the bots presence from detection systems. To hide their presence, bots use rootkit technique, generate small traffic and connect to C&C servers in random intervals of time. To erase the communication trace, botmasters may use fast-flux to rapidly change IPs and hide the real IP of C&C servers. Other common methods are using encryption for communication and using legitimate websites to deliver commands.

Botnet evolves to adapt technology advancement. Web-based botnet, especially Web 2.0 one, is predicted to rise thanks to the heavy use of the Internet and Web 2.0 services in today life and business. Some social networking websites such as Facebook, Twitter and Myspace have been abused as a means of propagation and communication by some Web 2.0 botnets. In the next chapter, we will study further on Web 2.0 botnet and describe a special type of Web 2.0 botnet that may escape notice of security detection thanks to its prospective way of communication. We will also propose a solution to prevent Web 2.0 botnet to spread and communicate via Web 2.0 services. Instead of preventing a bot from infecting a computer as most detection do, our solutions will prevent a bot from communicating with Web 2.0 based C&C servers. Communicating with the botmaster is an important step in the bot's life cycle as without being able to communicate, the bot becomes useless. That means the bot is not able to update, execute command as well as spread itself. As our scope is Web 2.0 services, our contribution is mostly relevant to the centralized botnet architecture and the HTTP C&C protocol. The solutions and the experiments will be presented in Chapters 4, 5 and 6

Nowadays, most of the bots target Windows users. In the future, we expect bots target other platform users such as Linux, Mac and especially mobile phone users. As smartphones have become everybody wallet and powerful like personal computers

(PCs), holding everything from banking to social network information, they have become targets for hackers, scammers and criminals. People now tend to use their mobile phones to access emails, social networking sites or even bank transactions. While people use their mobile phones more and more often, their devices are rarely protected from malware intrusion (eg using anti-virus software). The reasons can be:

- majority of mobile phone users are not aware of malware threats to their devices.
- anti-virus program can consume a lot of battery and the phone memory that may slow down their devices.
- people tend to change their mobile phones much more frequently than their PCs. Therefore, they may not want to spend money on purchasing anti-virus software for short-time-usage devices.

Therefore, it is not a surprise if mobile phones will be the main targets of bots in the near future.

3

Web 2.0 Botnet and Current Botnet Detection Techniques

As our goal is to investigate and detect Web 2.0 bots activities, it is essential to understand Web 2.0 techniques, characteristics as well as possible exploitations that botnet can take advantage on Web 2.0 services. In the first part, we describe Web 2.0 services, their APIs and Web 2.0 bots and their main attacks on Web 2.0 services. We also define Botnet 2.0, a recent type of Web bot, and its model of communication. In the later sections, we review some existing detection techniques and their relations to our work.

3.1 Web 2.0 botnet

3.1.1 Web 2.0 and Web 2.0 API

Web 2.0

Web 2.0, the second phase in the Web evolution, is also called the wisdom Web, people-centric Web, participative Web and read/write Web because of its emphasis on the ability for people to collaborate and share information online [95]. In contrast to its predecessor, Web 1.0 where users passively view websites contents created by web masters, Web 2.0 is more dynamic and interactive letting users both access content from a website and contribute to it. Examples of Web 2.0 services include:

- social networking websites: are online services that usually consist of representation of each user (known as profile), their social links and other additional services such as games and chatroom. Typical examples of social networking websites are Facebook, Twitter, Myspace and Google+.

- blogs: are online services that host personal journals such as Yahoo blog, Google blogpost and Livejournal.com.

- wikis: are websites which contents are created collaboratively by multiple users. Wiki allows users to create, update and edit contents via a web browser using simplified markup language or rich text editor. Examples of wikis are Wikipedia and WikiAnswers.

- video sharing sites: allow individuals to upload and share their own video clips on a website such as Youtube, Veoh and Dailymotion.

Web 2.0 comes with new technologies such as RSS/ATOM and API to assist users with ability to quickly get updated and post information to their profiles. RSS stands for Really Simple Syndication which is a family of Web feed formats used for syndicating content from blogs or Web pages. It informs users of updates to blogs or websites they are interested in so that users are free from manually keeping track the blogs/websites by themselves. Atom is another syndication specification aimed at resolving issues of multiple incompatible RSS versions.

Web 2.0 API

Some public Web 2.0 services, such as Google blogpost, Facebook, Twitter and so on, allow users to use API to automatically post information to their websites. API, an acronym for Application Programming Interface, is a source code interface that an operating system, library or service provides to support requests made by external systems. Using API, users are able to auto-update contents to their profiles without the need of manually logging on to the account and doing the update via a web browser. For example, a user can set up RSS and use API so that whenever a piece of information that he is interested in or is relevant to his business is posted on the website that he subscribes, his profile will be auto-updated with that new information without the need of his interactions.

One of the most used Web 2.0 API is Facebook Platform API. According to Facebook, their API enables anyone to "build social applications on Facebook and the Web." [96]. The core Facebook Platform API is the Graph API that allows users to read and write data to and from Facebook. Graph API uses object-oriented method of reading and writing data to and from Facebook and their relationships or connections with each other. Objects and connections in Graph API are user profiles, friends, posts, photos, likes and so on.

Graph API objects are assigned a unique ID and are easily addressable using a URL that can be further qualified to address a specific object/connection. The general structure of an object URL is as follows:
`https://graph.facebook.com/OBJECT_ID/CONNECTION_TYPE`
where `OBJECT_ID` is the object's unique ID and `CONNECTION_TYPE` is one of the connection types supported by the object. For example, a page supports the following connections: feed/wall, photos, notes, posts, members, and so on. With the Graph API, users

can retrieve an object, delete an object, and publish objects. They can search, update objects, filter results, and even dynamically discover the connections/relationships of an object.

By default, applications have access to the user's public data. To access private data, applications must first request the user's permissions, which are called extended permissions. At a high level, we need to get an access token for the Facebook user. After obtaining the access token for the user, we can perform authorized requests on behalf of that user by including the access token in the Graph API requests: `https://graph.facebook.com/220439?access_token=...`. Facebook defines a large number of permissions which can be found in [97].

To post information to Facebook, we use HTTP POST request to the appropriate connection URLs, using an access token. For example, we can post a new wall post on Alice's wall by issuing a POST request to `https://graph.facebook.com/alice/feed`:

```
curl -F 'access_token=...'
-F 'message=Hello, Alice.  How is your day?.'
https://graph.facebook.com/alice/feed
```

We can comment on or like any object that has a `/comments` or `/likes` connection by posting to `https://graph.facebook.com/OBJECT_ID/comments` and `https://graph.facebook.com/OBJECT_ID/likes`, respectively:

```
curl -F 'access_token=...'
https://graph.facebook.com/123456789/likes
```

Table 3.1 [96] lists some methods of publishing objects. Other APIs can be found in the Facebook development resource page [96].

Method	**Description**	**Arguments**
`/PROFILE_ID/feed`	Publish a new post on the given profile's feed/wall	`message, picture, link, name, caption, description, source, place, tags`
`/OBJECT_ID/comments`	Comment on the given object (if it has a `/comments` connection)	`message`
`/OBJECT_ID/likes`	Like the given object (if it has a `/likes` connection)	none
`/PROFILE_ID/notes`	Publish a note on the given profile	`message, subject`
`/PROFILE_ID/links`	Publish a link on the given profile	`link, message, picture, name, caption, description`
`/PROFILE_ID/events`	Create an event	`name, start_time, end_time`
`/EVENT_ID/attending`	RSVP "attending" to the given event	none
`/EVENT_ID/maybe`	RSVP "maybe" to the given event	none
`/EVENT_ID/declined`	RSVP "declined" to the given event	none
`/PROFILE_ID/albums`	Create an album	`name, message`
`/ALBUM_ID/photos`	Upload a photo to an album	`message, source, place`
`/PROFILE_ID/checkins`	Create a checkin at a location represented by a Page	`coordinates, place, message, tags`

Table 3.1: Methods supported in Facebook Publishing API

3.1.2 Web 2.0 Risks

Although Web 2.0 has improved our ability to freely communicate and share via the means of the Net, it has brought some unimaginable dangers and as a result it is insecure. The following list shows top threats against the Web 2.0 services [98, 99]:

- **Cross-site Scripting Attack**. A user goes to a website and unwillingly downloads a malicious java script. It gets executed on a user browser. The script may compromise the security causing a leakage of the user private information. For example, a forged link displayed on a website redirects users to a phishing site. Any download from the site contains a phishing malware. One typical example of cross-site attack was the Samy worm that exploited MySpace cross-site scripting flaw. There was incorrectly written script running on MySpace client side that was exploited by an attacker. The attacker is only required to craft a malicious link to coax unsuspecting users to visit a certain page from their Web browsers. We also once tried a simple cross-site scripting on the popular Google blogspost. Unexpectedly, Google blogspot allowed embed JavaScript as well as an iframe in the post. As a result, by writing a small JavaScript that redirect users who are visiting our blog to a malware websites or using iframe to embed a malware website to our blog post, those users' computers will get infected by the malware.

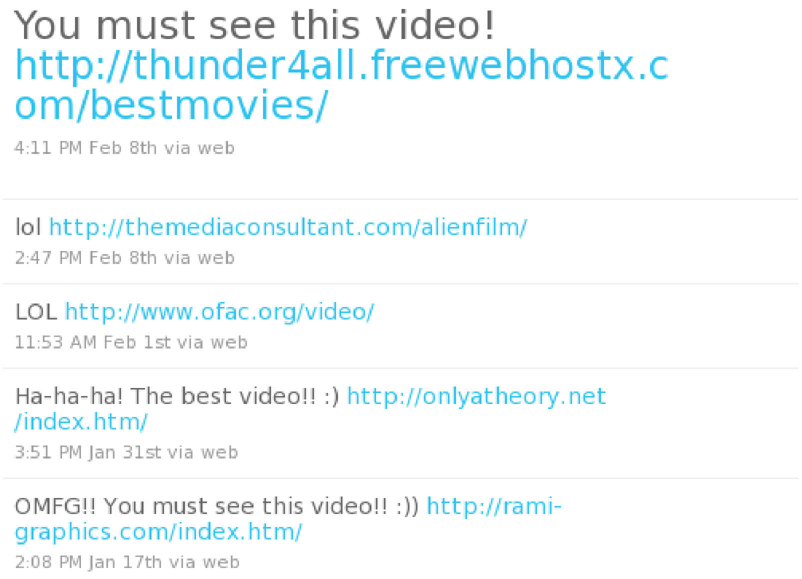
- **RSS/ATOM Injection**. An adversary injects a JavaScript into the RSS feeds. The script contains malware. When a user visits this website, they load the page with the RSS and the malicious script. The script then gets executed on the user machine. For example, Jeremy Moeder [100] discovered that when Yahoo's RSS aggregator allows a person to add an RSS feed to its website, it doesn't properly check the XML file to make sure it doesn't contain possibly malicious code. Therefore, it could be disguised with JavaScript to look like a link to add a valid news feed, eg CNN feed. The malicious links could also be made to look like real headlines, or even grab headlines from CNN RSS and replace the links with malicious code.

In addition, with the aid of RSS, Search Engine and Splogging (Spam and Blogging), malware can spread faster. For example, by spamming on the blog or bookmark pages, a hacker can advertise his webpage and gain higher pagerank, hence, will appear in top pages searched by search engines. As a result, users are more likely to visit the hackers' websites and their computers get infected.

- **API abuse**. An adversary uses the public application programming interface (API) to post spam to the Web services and/or create a communication channel between a botmaster and a bot. The purpose of Web 2.0 API is to facilitate users with ability to update information automatically and on schedule. However, this also offers botnet a new means of communication. In particular, besides sending phishing emails, attackers now can instruct bots to post malicious links to Web 2.0 services via the provided API. For example, Koobface bots propagate by posting malicious links to social networking websites such as Facebook and Twitter. Another example is TwitterBot. The botmaster creates a Twitter account and update commands to that account. Her bots once alerted by RSS of Twitter will log into that account and retrieve the commands by using API.

3.1.3 Web 2.0 Botnet

As Web 2.0 services evolve, so do the efforts of botnet writers. Web 2.0 offers all users freedom to contribute to the web content and provides a public application programming interface (API) allowing users to automatically update content and retrieve information. Malware writers also rapidly adopt these new technologies to increase the sophistication of their attacks and create a new generation of botnet, the so-called Web 2.0 botnet. They have been targeting social networks and Web 2.0 sites such as Twitter.com, FaceBook.com, YouTube.com and Google/Yahoo blogs because these sites are not only legitimate, with verifiable SSL or EV-SSL certificates, but also heavily used by millions of users [101]. Thus, the interactions between bots and their botmasters are difficult to notice when they are a small part of a heavy traffic generated between the server and its users. Similarly, by observing the traffic between the (client) machine and Internet services, it is difficult to detect sporadic and short communication sessions between a bot (installed on the machine) and its botmaster.



You must see this video! http://thunder4all.freewebhostx.com/bestmovies/
4:11 PM Feb 8th via web

lol http://themediaconsultant.com/alienfilm/
2:47 PM Feb 8th via web

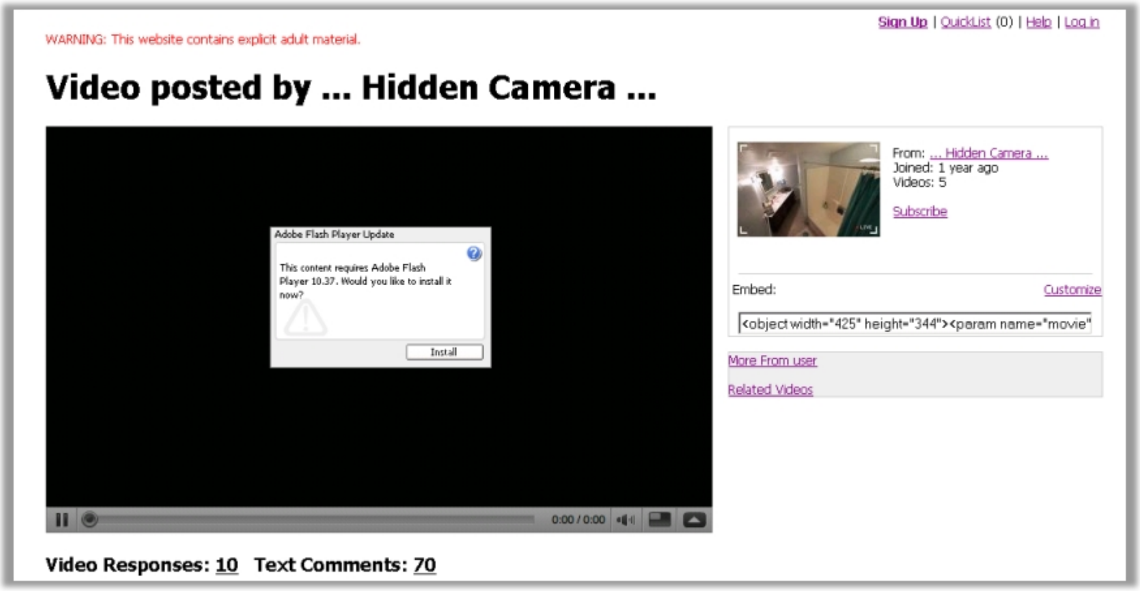
LOL http://www.ofac.org/video/
11:53 AM Feb 1st via web

Ha-ha-ha! The best video!! :) http://onlyatheory.net/index.htm/
3:51 PM Jan 31st via web

OMFG!! You must see this video!! :)) http://rami-graphics.com/index.htm/
2:08 PM Jan 17th via web

Figure 3.1: Sample Twitter Koobface Spam

It is not a surprise to see that botnets are using social networks as a tool for bot propagation. For instance, the Web 2.0 bot called Koobface (anagram of Facebook) appeared in 2008 and is using social networks (Facebook, Twitter, Myspace, etc.) to distribute the malware [60, 102]. A typical Koobface infection starts with a spam sent through Facebook, Twitter, MySpace, or other social networking sites containing a catchy message with a link to a “video”. Figure 3.1 is an example of Koobface spam on Twitter. Clicking the link will redirect the user to a website designed to mimic YouTube (but is actually named YuoTube), which asks the user to install an executable (.exe) file to be able to watch the video. For example, the user is required to update flash player as shown in Figure 3.2 [77]. Once the user downloads and installs the bot, it becomes active. It will start communicating with C&C servers for commands and try to propagate itself.



WARNING: This website contains explicit adult material.

Video posted by ... Hidden Camera ...

Adobe Flash Player Update

This content requires Adobe Flash Player 10.37. Would you like to install it now?

From: ... Hidden Camera ...
Joined: 1 year ago
Videos: 5

Embed: <object width="425" height="344"><param name="movie"

Video Responses: 10 Text Comments: 70

Figure 3.2: Fake Youtube page that leads to download Koobface worm

3.1.4 Botnet 2.0

A recent and greater threat to Web services is the ability of botmasters to use these services as an intermediate to control bots without having a direct communication channel. As illustrated in Figure 3.3 [40], attackers can use popular and public access Web 2.0 services such as Google or Yahoo blogs and social networking websites to communicate with her bots and store stolen information.

In particular, the C&C flows is described as follows:

1. The adversary posts commands and updates to a public blog service.

2. Thanks to RSS, her bots are alerted and get commands and updates from the blog service.

3. The bots then run the commands/updates and subsequently send responses or

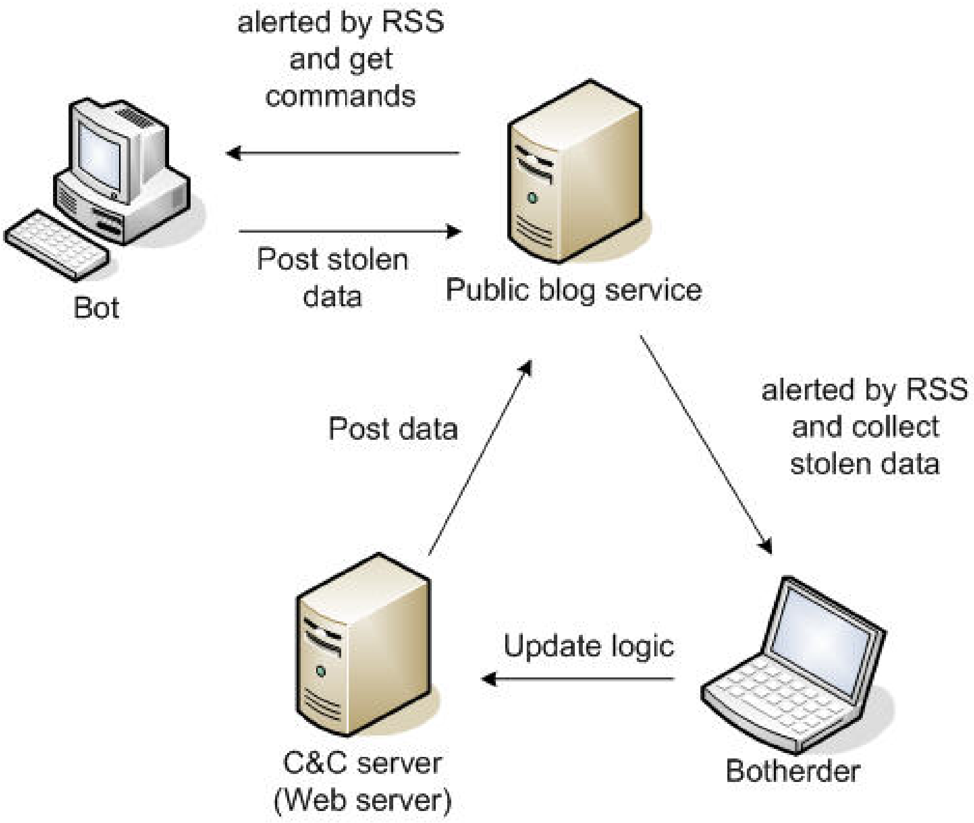


Figure 3.3: Trojan 2.0 Command and Control

stolen information such as user's bank accounts or sensitive information to the blog service.

4. The attacker will collect the stolen data when alerted by RSS.

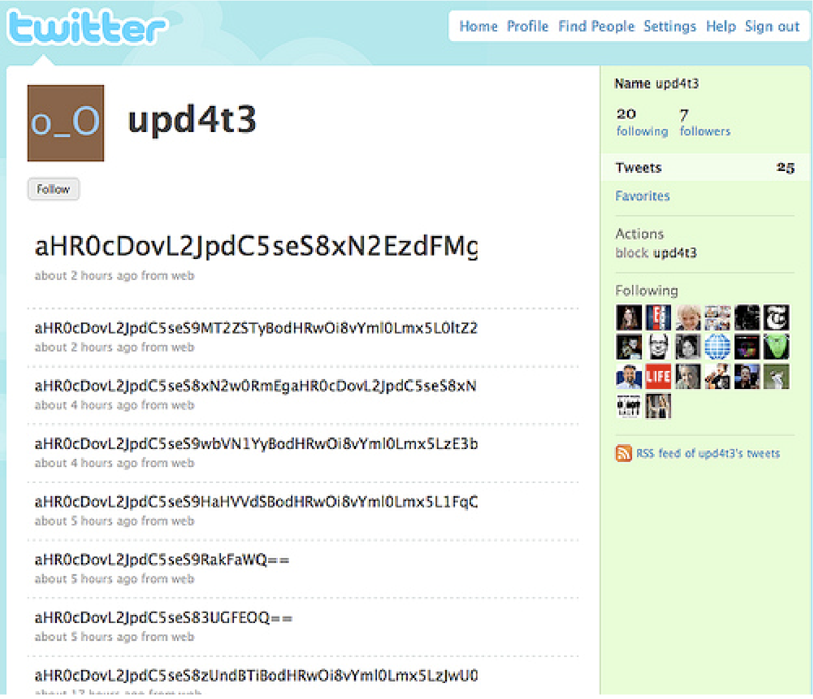
So the attacker uses public blog service as a temporary storage for C&C and stolen data. Consequently, the attacker can hide her identity as well as can bypass current botnet detection tools since the communication is just like a HTTP traffic and the blog service used is a popular service, which is white-listed by most detection tools. This type of malware is named a Trojan 2.0 by Finjan [40], a provider of secure web gateway solutions for the enterprise market. In this study, we refer the bots and botnets that use this communication method as Bots 2.0 and Botnets 2.0.

```
<entry xmlns=‘http://www.w3.org/2005/Atom’>
/*-- id of the bot ---*/
   <title type=‘text’>News from "AUS1234"</title>
   <content type=‘xhtml’>
      <div xmlns="http://www.w3.org/1999/xhtml">
   /*--- post content:  bank account information --- */
           <p>Commonwealth</p>
           <p>Joe Smith</p>
           <p>123456</p>
           <p>9876 5432 1987</p>
      </div>
   </content>
   /* --- categories of the post ---*/
   <category scheme="http://www.blogger.com/atom/ns" term="bank"/>
   <category scheme="http://www.blogger.com/atom/ns" term="AUS1234"/>
</entry>
```

Figure 3.4: An example of post entry from a bot

For example, a botnet can use Google blog service as a temporary storage of stolen information. To automatically manage blog entries such as creating, editing and deleting, a user needs to use a Google API called Google Blogger Data API. The following text presents a segment of code showing how a bot can post a new entry (eg bank account of the victim) to the attacker blog [103]. Assume that the botmaster has already registered a Google account and the Google blog server is at http://www.blogger.com. First, the bot needs to create a piece of XML code to define post entries. This code might look like one in Table 3.4. Then the bot use the attacker account credentials to authenticate and publish the new blog entry to the blog host at http://www.blogger.com using the HTTP POST method

In September 2009, researchers at Symantec Security Response discovered command-and-control data streaming through Google Groups, Google online discussion forums. The Trojan was coded to login to Google Groups and then redirected to a webpage containing encrypted commands. Symantec said the Trojan then posted the data it collected from victim's machines to the newsgroup [23, 26].



upd4t3

Name upd4t3

20 following 7 followers

Tweets 25

aHR0cDovL2JpdC5seS8xN2EzdFMg
about 2 hours ago from web

aHR0cDovL2JpdC5seS9MT2ZSTyBodHRwOi8vYml0Lmx5L0ltZ2
about 2 hours ago from web

aHR0cDovL2JpdC5seS8xN2w0RmEgaHR0cDovL2JpdC5seS8xN
about 4 hours ago from web

aHR0cDovL2JpdC5seS9wbVN1YyBodHRwOi8vYml0Lmx5LzE3b
about 4 hours ago from web

aHR0cDovL2JpdC5seS9HaHVVdSBodHRwOi8vYml0Lmx5L1FqC
about 5 hours ago from web

aHR0cDovL2JpdC5seS9RakFaWQ==
about 5 hours ago from web

aHR0cDovL2JpdC5seS83UGFEOQ==
about 5 hours ago from web

aHR0cDovL2JpdC5seS8zUndBTiBodHRwOi8vYml0Lmx5LzJwU0

FIGURE 3.5: Sample commands posted by TwitterBots

Among a few existing Botnet 2.0 networks, TwitterBots are the most prevalent examples of how cybercriminals can use social networks as the new platform for botnet command, control, and attack. Twitter-controlled bots, uncovered by Jose Narario in 2009 [24], were operated by Brazilian thieves who created a Twitter account (`upd4t3`) for a sole purpose of sending commands to its associated bots. The communication between the criminals and their bots is done as follows. Each command is posted as an update of a status on a Twitter account. The Twitter RSS alerts the bots and then they retrieve the new command. Figure 3.5 gives a sample of such commands. The commands are encrypted.

3.2 Current Botnet Detection Techniques

Although botnets appeared more than 10 years ago, they have only recently sparked an avalanche of research. Main research effort has been directed into a development of new tools for detecting bots and botnets. Most of the techniques used are based on existing tools already applied for detecting viruses, Trojan horses, and spam software (eg traffic analysis technique). New techniques apply a blend of various techniques such as honeypots and tracking techniques.

3.2.1 Traffic analysis

One of the most common methods for detecting bots is analyzing network traffic flows. Some attempts have been made to examine traffic contents for finding IRC botnet commands then filtering them out or triggering alerts. Livadas et. al. [31], proposed machine learning-based classification techniques to identify C&C traffic of IRC-based botnets. The authors have attempted to label real and botnet chat flows based upon historical data and the difference of length and content between real chat text and botnet commands. However, this method appears to have a high false detection rate because obtaining an accurately labeled data set for machine learning is challenging. In addition, it is not hard for hackers to mimic the real chat flow to avoid detection. Another disadvantage is that, the data set needs to be updated regularly.

By monitoring network traffic flows, on the other hand, several studies focus on examining traffic behaviors such as bandwidth, duration of flow (by recording flow start and end times) and packet counts looking for evidence of botnet C&C activities [31, 32, 104]. The authors examine real bot traffic flow and try to classify the chat traffic into two groups: non botnet traffic and likely botnet traffic. According to Strayer et. al. [32], non-botnet traffic flow likely has the following properties:

- do not contain TCP packets. Note that the authors aim for IRC botnet communication,
- high bandwidth flows such as software updates and rich web page transfers,
- short-lived flows - flows of only a few packets or a few seconds

However, again, this method relies heavily on historical data, which quickly gets obsolete if the data set is not updated on time when the hackers use different techniques to mimic legitimate traffic flows. This approach is also likely to produce a high false positive detection rate [105].

BotHunter

While the traditional traffic analysis inspects independently the uploads and downloads for signs of malware, the BotHunter [21] software correlates the two-way traffic between client machines (that may be infected by bots) and server machines (that may host botmasters). BotHunter uses malware packet sensors that are tuned to detect (1) port scanning activity, (2) specific exploit patterns, (3) signatures of bot coordination commands, (4) outbound attack propagation. BotHunter also is able to learn the pattern of the normal traffic by observing communication within a trusted network, where there is no bot. After detection of bot traffic, it alerts users about the botnet activity. Suspicious bots typically match an infection sequence model. For example, BotHunter declares a host is infected when there is an incoming infection warning followed by one of the following activities:

- there is outbound bot coordination dialog
- there are exploit propagation warnings
- there is a minimum of at least two forms of outbound bot warnings such as bot binary download, C&C communications and outbound scan.

With BotHunter, a network administrator can identify suspicious communication between a node (machine) of a network and an external server. The communication can be stopped if necessary. Like other traffic analysis tools, BotHunter most of the time listens to the traffic. From time to time, however, it update its database that stores list of botnet command and control blacklist, malware DNS list and new malware-detection rules. This allows BotHunter to maintain awareness of the latest botnet operator servers, malware-associated DNS lookups, known-bad server addresses and malware back-door control ports. Though BotHunter is considered as one of the most effective tools for bot detection, it has its limitations [35]. They are:

- it is passive and has to observe relatively long sequences of sessions,
- it is not able to detect new bots whose signatures are not stored in the database,
- it is not able to detect an old bot when the bot changes its signature (behaviour),

3.2.2 Honeypots

Honeypot is a well known technique for discovering the tools, tactics and motives of attackers who own botnets. Honeypot is actually a bot that joins a botnet to spy and gather information about the botnet [18, 106, 107]. Many researchers and white-hat (computer hackers intending to improve security) crime fighters have been using honeynets, a network of more than one honeypots, to monitor the local network, to identify the bot activity and if present, to learn about their behaviour and shut them down if possible. For example, the Honeynet Project [20] has done extensive work on capturing live bots and characterizing botnet activities, which are valuable information for research community. Honeypots are not only for gathering information; they can also be employed for detection and prevention. That explains why honeypots have been deployed in many defense systems.

However, maintaining a honeynet is not an easy job. The more interactions a honeypot makes with the botnet, the greater risks it takes. The honeypot not only needs to protect itself from being taken over by the attacker (and becoming a “true” bot) but also needs to secure the operating system as well. This means that we need to always keep the honeypot and the OS up to date. In addition, like a normal bot, a honeypot needs to be updated frequently to cope with new threats and vulnerabilities that are discovered. The effective maintenance of a honeynet requires a support of well-developed policies and activities [1].

Honeypots act like real bots to avoid detection by botmasters. But at the same time, they must not be equipped with any defence mechanisms that may harm other processes or computers. This requirement is imposed by the computer law. The harmless nature of honeypots can be used by botmasters to distinguish between honeypots and bots. Zou and Cunningham [108] have presented various honeypot detection methodologies which can be applied during the botnet’s propagation stage to successfully detect and remove honeypots. This weakness of honeypots needs an urgent attention from the research community.

3.2.3 Detecting software vulnerabilities

Most bots propagate thanks to software vulnerability exploits. Therefore, scanning, tracking and finding vulnerabilities then fixing them before the bots use them to infect computers appear to be a natural and efficient way to prevent bot break-ins. Indeed, recent advances in the defense of networked computers has used such tools and techniques, e.g network detectors and Self-Certifying Alert (SCA), for tracking tainted data and detect attempted break-ins automatically [109]. SCA is known as one of the most promising techniques to defend system against software vulnerability exploits. When a vulnerability is detected, each SCA-registered computer will be alerted with a description of exploits. A patch to fix it is also distributed and run as soon as possible to protect as many computers as possible.

However, this technique can also benefit the attackers. Finding software vulnerabilities is obviously not a trivial task. So hackers can leave this hard job for detectors and just take advantages from the exploits found by detectors. This is called exploits hijacking [109]. For example, the botmaster can register few of her bots to SCA. As described above, when a vulnerability is found, the exploits description is sent to all computers in the SCA network, including the bots. The bots in turn send the information to the botmaster. The information provided by the detector is clear enough for the hackers to attack the computer using the same vulnerabilities. So if the worms (auto-worms) are fast enough, many computers will be infected before they are patched. Raiciu et. al. [109] have conducted simulations in which fast auto-worm and automated defenders are racing to infect and protect networked computers using the same vulnerabilities found by detectors. The results reveal that when SCA is very fast, the auto-worm still infects 5% of the hosts. If SCA verifications take 1 second, the infect proportion is about 20% and approaches 80% when SCA delays 4s. In reality, SCA verification may be quite slow, which leave the auto-worm more chance to win. The authors have attempted to propose some possible defenses against exploit hijacking, however, bundling these ideas into a complete solution is challenging.

3.2.4 Network traceback

So far we have concentrated on techniques that are used to protect the network computers from infection by bots. There is, however, an alternative. Instead of trying to defend computers in the network, one can try to determine the location of bots, C&C servers and botmasters to be able to shut down the whole or large portions of a botnet. This technique is called network traceback [5, 110, 111]. There are two categories, namely IP traceback and connection chain.

IP traceback attempts to locate the source of spoofed IP packets. It is likely motivated by the problem of determining the source of denial-of-service (DoS) attacks, which tend to use spoofed packets to hide the identity of the physical source. There are three main approaches in IP traceback. They are:

- Ingress filtering [112]: placing filters in a few key points in the network to limit

IP spoofing. In this approach, routers (at key points) are configured to block packets that arrive with illegitimate source addresses. This requires a router with sufficient power to examine the source address of every packet and sufficient knowledge to distinguish between legitimate and illegitimate addresses. However, when the traffic load is high, there is no longer enough information to unambiguously determine if a packet arriving on a particular interface has a "legal" source address. In addition, in practice, attackers could still forge addresses from the hundreds or thousands of hosts within a valid customer network [113].

- Packet marking [113, 114]: adding labels to every IP packet (deterministic) or random IP packets (probabilistic) to trace the source of the packets. In this approach, each router inscribes its local path information onto a traversing packet so that the destination node (i.e., victim of an attack) can reconstruct, with high probability, the complete path traversed. The path is reconstructed by inspecting the markings on the received packets, assuming the attack volume is sufficiently high. Because of the limited space available to insert information into the packet headers, these schemes generally require that a large number of packets be sent along the same set of paths to ensure precision. Therefore, none of the packet marking techniques has been successfully implemented in a real system [110].

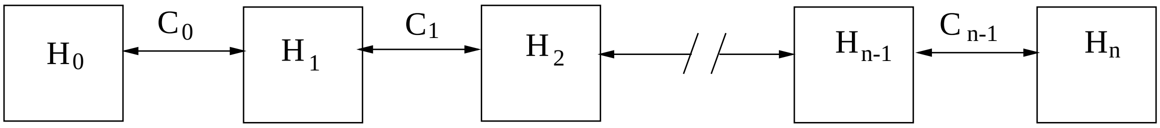


FIGURE 3.6: Connection chain example between H_0 and H_n

A connection-chain C is a sequence of hosts H_i, in which any pair of hosts is connected by a communication channel C_i (see Figure 3.6). To track back a suspicious IP address, the administrator of a host H_i asks the administrator of H_{i-1} (or alternatively H_{i+1}) to investigate their host. This process continues recursively along the chain. Unfortunately, it is done mostly manually so it is expensive and time consuming. In the studies published so far, there are few attempts to automate the tracing procedure. The following list shows the most promising procedures

- using network traffic analysis to match streams in a connection chain [111, 115]. However, the effectiveness of the method has not been demonstrated.
- improving and designing a protocol to support recursive traceback request. The purpose is to gather information about the entire path of a connection [110].

3.3 Summary and Relation to Our Work

We have described Web 2.0 services and main threats against them. The Web 2.0 services provide many attractive features such as easy downloading and publishing, efficient sharing of data (such video, pictures, text, etc.), and supports creativity and interactions. Unfortunately, the new opportunities created by Web 2.0 can be abused by malware designed by adversaries.

A good example of opportunities and challenges created by Web 2.0, are social networking services such as FaceBook, Twitter and Myspace. The unrestricted and public access to these services is very attractive to adversaries who may use it for their own purposes such as:

- propagate malware by tricking users click on malicious links posted on social networking services
- spam (advertising and/or phishing)
- build C&C channels to communicate with bots

The use of social networking sites for propagation works because Web 2.0 botnets have taken advantage of how social networking sites work and how people use them. They constantly trick victims into downloading malware components through its video-sharing scam, which mimics what the majority of social network users do online. They also leverage the implied trust that builds between victim (the sender) and other users in the victim's friend list. As a result, this increases the chance of the victim's friends clicking the malicious URL.

Botnet 2.0 is a collective name of a family of bots interacting with their botmasters via their C&C channels that are built on the top of Web 2.0 services. The C&C channels used in Botnet 2.0 is becoming cybercriminals' choice because of the following characteristics:

1. Abusing trusted popular websites as a C&C server to improve resilience. Social networks and Web 2.0 sites such as Twitter.com, FaceBook.com, Google, and YouTube.com are popular and legitimate that are white listed by majority of antivirus software. In addition, due to the heavy usage by millions of users, light occasional traffic to one or more accounts is unlikely to be noticed compared to a user's actual traffic pattern. This also avoids DNS queries as antivirus tools usually do for non-popular DNS names.

2. Exploiting popular port(s) for C&C communication. Port 80 is standard for web traffic that most network traffic will flow through it. This helps bots blend in with benign traffic.

3. Abusing Web 2.0 application features for automatic C&C. The botmaster uses application features, such as RSS feeds and API, to automatically update bots.

4. Cost Effective: as Web 2.0 services are leveraged to function as C&C servers and temporary storages of stolen information, the botmasters can save money and efforts from hiring or setting up their own C&C servers and storages. Each account like `upd43t` created by TwitterBots is functioning like a C&C server of the botnet so the botmasters can have as many C&C servers and storages as they need for free.

Fortunately, so far, there are a few reports only about bot incidents discovered in social network web services. But it is highly likely that this generation of botnets is going to grow in both numbers and sophistication [59, 116].

In this chapter, we have also reviewed some current detection techniques. Each technique has its own advantages and disadvantages that may or may not be suitable for detecting Web 2.0 botnets. Detecting software vulnerabilities techniques will not work for Web 2.0 bots because they mainly propagate using social engineering method, which tricks users into downloading and executing them rather than exploiting software vulnerabilities. Honeypot and network traceback techniques can be applied to track some Web 2.0 botnets like Koobface because this type of botnet still has its own C&C servers and the bots still need to communicate with these servers for commands and updates. In other words, we can inject some honeypots to the Koobface network or place some filters in the network to investigate the source of the commands in order to discover and take down C&C channels. However, these approaches fail to trace Botnet 2.0 as they are going to find the address of a legitimate website. Note that, in Botnet 2.0, the C&C servers are located on the targeted website and the botmaster never communicates directly with their bots.

As long as traffic analysis techniques are concerned, they are inefficient for detecting Web 2.0 botnet as bot activities are indistinguishable from typical HTTP requests and responses generated by legitimate users and websites. Most social network sites currently monitor the posted contents and employ heuristic rules to differentiate human and bots activities [38, 101]. For example, the detection system looks into each text to search for malicious codes or suspicious links. Those posts that come from IPs in the blacklist are also marked as suspicious. Once the content is marked as suspicious, the user will be asked to solve a Captcha to verify themselves as a human before the content is published. However, this approach not only requires a lot of resources as there are millions of posts everyday on popular websites but also produces high rate of false detection [77]. Advanced bots have different ways to hide their code and fake links such as using encryption and obfuscation. Also, blacklisting IPs will not work since it is easy to mask IP and may also cause annoyance to those genuine users that have their IPs happen to be within blacklisted IP range. In addition, Captcha has been the most efficient Turing test yet the recent relay attack can break almost all current Captcha forms as it involves human to solve the challenge. This explains why Koobface has been able to bypass the defense systems and spread to millions of computers.

In conclusion, most of the current detection techniques are unlikely able to efficiently differentiate activities from human and web bots. In the next chapter, we will present a novel method to filter auto-update request (via API) from bots. We will not base on the text content but attempt to identify if a user uses a new computer to auto-update (via API) their profile the first time. If it is the case, the user will need to solve a Captcha to prove that the auto-update request comes from a genuine user. After passing the Captcha verification, the following API calls from the same computer will be accepted without the needs of another Captcha verification. We also develop an enhanced Captcha design that is resistant against relay attack. The Captcha design will be described in Chapter 5.

4

API Verifier: an application-based approach to detect Bot 2.0

Advances in web technology and more specifically the development of Web 2.0 created new opportunities for web interactions and internet services. Unfortunately, these opportunities have also been exploited by cybercrime groups to design the next generation of bots and botnets with increased abilities and more sophisticated characteristics. The current detection and anti-bot software fails to protect the Internet against the next generation botnets. In this chapter, we present a novel approach to deal with the next generation botnets (i.e. botnets designed using the Web 2.0 technology). Our solution applies the following security components: a unique identifier of the computer, a block encryption algorithm (such as AES) and a verification based on Captcha.

4.1 Motivation

To update information on a Web 2.0 site, a bot must use the API provided by the Web 2.0 service. API stands for Application Programming Interface which is a set of functions or methods for external system/program to access some functionalities provided by the service. In particular, the bot is programmed to contain methods calling the blog API functions to update old and to create new entries. To prevent bots from accessing API, we can verify whether a user is a bot or a human being. If the verification shows that a user is not a human being, the access is denied.

It is known that the Captcha techniques can distinguish a computer program (bot) from a human being (n.b. Captcha stands for **C**ompletely **A**utomated **P**ublic **T**uring test to tell **C**omputers and **H**umans **A**part [117]). Therefore, we will challenge the user to solve a Captcha for verification. This request only happens once a user calls first time API from a given computer. In other words, if the user has passed a Captcha verification on a computer A, the user will not be asked to repeat Captcha verification when they use the same computer A again.

To create a Botnet 2.0, a botmaster signs up to one or many accounts on a Web service. Then the botmaster needs to write a program (bot) to automatically post blog entries to the blog Web service from a victim computer. This means one blog account is shared by many bots (computers). When a victim computer is infected by a Bot 2.0, the bot will do its activity by starting reporting back to the botmaster by posting information to the blog service using API. If the web server learns that the API call is from a new/different computer, it will request the user to verify himself by solving a Captcha. However, because the user is a bot which cannot solve the Captcha challenge, it cannot pass the Captcha verification. Hence, the API request is denied and as a result, the communication channel between bots and the botmaster cannot be established.

To be able to identify if an API call is from a new/different computer, we need to have a unique and permanent identifier of the computer. In this study, we use the permanent MAC address, which is globally unique and identifies the computer. We will discuss further how to guarantee that the permanent MAC address will not be spoofed by the attacker during the verification phase in section 4.4.1.

4.2 API Verifier Design

The API Verifier consists of 2 components: an API Verifier Client (APIV Client) and an API Verifier Server (APIV Server). The main functionality of the API Verifier is to check if a user is a human being, before letting the API call from the user be executed by the web server. The API Verifier at the client side is a program installed on the user's computer. Figure 4.1 illustrates the components of the API Verifier.

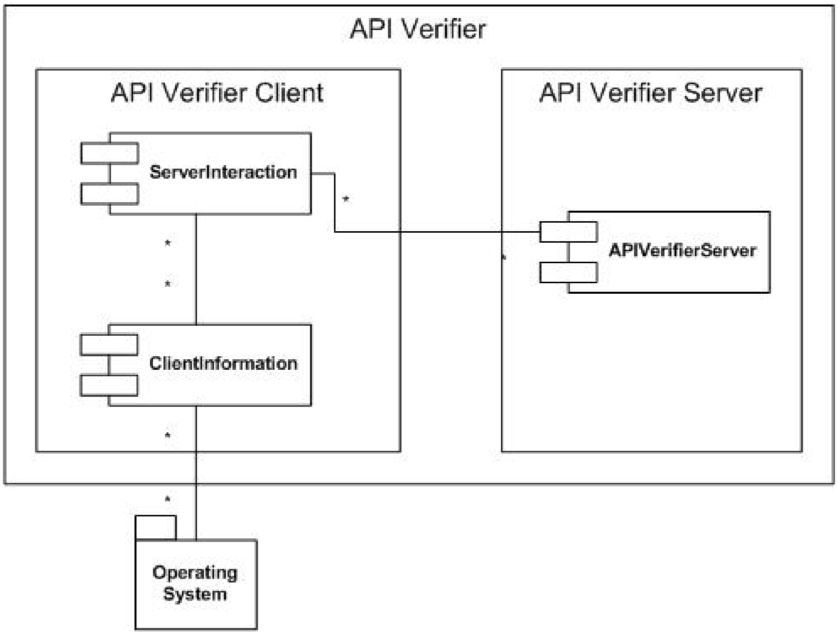


Figure 4.1: Components of API Verifier

4.2.1 API Verifier Client

As mentioned above, the APIV Client is a program running at the client side. The APIV Client is also a library that a user needs to import into their code to be able to use the Web 2.0 service API. It has two main classes: a Server Interaction and a Client Information (see Figure 4.1)

1. *Client Information*: gets permanent MAC address from the computer operating system, encrypts the MAC Address, user's credentials and returns the encrypted message to the Server Interaction.

2. *Server Interaction*: handles all interactions with the server such as making connection, sending/receiving data, displaying Captcha images returned from the server and getting Captcha answer submitted by users. The Server Interaction also communicates with the Client Information to pass tokens to and receive encrypted MAC addresses from the Client Information.

4.2.2 API Verifier Server

The APIV Server is software at the server side which is responsible for interaction with the APIV Client. The interaction is done by performing the following steps: authentication, MAC address verification, Captcha generation and verification and executing API calls. The APIV Server also handles time token generation and updating user's profile when needed.

4.2.3 Authentication

Only valid users are able to make API calls to the server. The Server Interaction component at the client side is responsible for encrypting username/password and sending them to the server for authentication. At the server side, the APIV Server decrypts the username, password using the same key, verifies these credentials and responses to the client. After the authentication step, the APIV Server generates a random token and returns it to the client; otherwise returns an error message. The

APIV Server is also customizable to handle session for a given time. For example, the response from the client has to be received by the server within a fixed time interval (say 3 minutes). Otherwise, the session between the client and the server is terminated.

4.2.4 Session Keys

Session keys are used to encrypt the MAC addresses. Note that this key is different from the secret key used for authentication. A session key is generated independently each time an API call is made. It is created by XOR-ing the secret key contained in the API Verifier Client with a time token returned by the server after a successful authentication. The secret key is generated once and used for all instances of API Verifier, where as the token is randomly generated with respect to time when an API call is made.

4.2.5 Permanent MAC Address

For each ethernet/wireless adapter, there are current and permanent MAC addresses. The current MAC address is the one stored in the operating system registry and displayed as the physical address when you view detailed network configuration information with the `ipconfig` utility. On the other hand, the permanent MAC address is not displayed and is the code burned in the EEPROM of the device [118]. By default, the current MAC address is the same as the permanent one. However, the current MAC address can easily be changed (see [119] for how to change the current MAC address in Windows XP) while the permanent MAC address cannot be modified unless the manufacturer does it for you. In our approach, we only use the permanent MAC address. The reasons for that will be discussed in section 4.4.1. In Windows, one way to obtain the permanent MAC address is to query looking up for an object with the identifier `OID_802_3_PERMANENT_ADDRESS` [120].

4.2.6 Captcha Verification

Captcha verification happens when the MAC address sent from the user's computer does not match the one in the user profile stored in the server database. The APIV Server will generate a random Captcha containing a challenge and send it to the client side. The APIV Client will display the Captcha and get the Captcha answer submitted by the user then send it back to the server for verification. The Captcha challenge must be hard to solve by a computer program but easy for a human being. If the user fails to solve the Captcha, they will be asked to solve another new Captcha randomly generated by the sever. A user is given three attempts to solve the Captcha within a given time interval (which can be adjusted by the server). After three fails, the user will be blocked and, depending on the website policy, waiting for further verification to prove that they are genuine users.

4.2.7 Update of User Profile

A user profile consists of a username, a password, a description of the user and a MAC address. When the user registers for an account, the username, the password and other information except the MAC address are stored in the database. At the beginning, the MAC address field is empty because the MAC address is only used for API verification and majority of users may never use an API to auto-post an entry. During the API Verifier process, only the MAC address can be updated.

To address the privacy concern about the permanent MAC address, the MAC addresses are encrypted while sending via the Internet and protected by using hashing when storing in the server database. The MAC address in the user profile needs to be updated in two situations:

1. When the user uses API for the first time. In this case, the MAC address at the server side is empty so the system needs to store the MAC address after the user has passed the Captcha verification.
2. When the user calls API from another computer. In this case, the MAC address

sent from the APIV Client is different from the one stored in the database. The new MAC address will replace the old one after the user has passed the Captcha verification. If the service allows one user to have multiple MAC addresses, e.g three MAC addresses, up to three MAC addresses will be stored. If all three slots for the MAC addresses are used, the new MAC address will replace the first one, which is the oldest one.

4.3 API Verifier Work Flow

The workflow of the API Verifier can be summarised as shown in Figure 4.2.

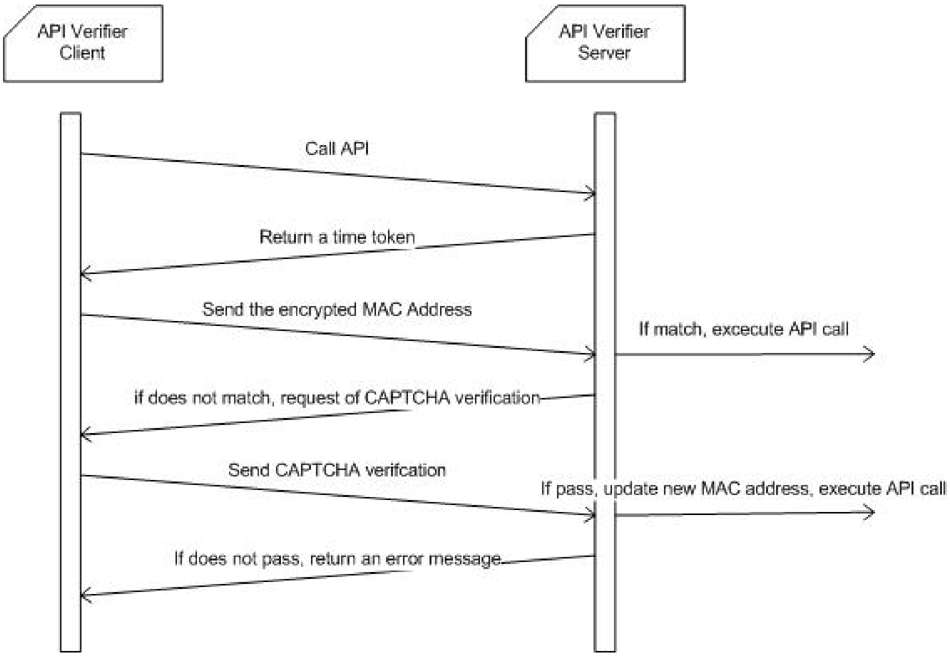

API Verifier Client
API Verifier Server
Call API
Return a time token
Send the encrypted MAC Address
If match, excecute API call
if does not match, request of CAPTCHA verification
Send CAPTCHA verifcation
If pass, update new MAC address, execute API call
If does not pass, return an error message

Figure 4.2: The API Verifier Workflow

When the user makes an API call to the web server, the API Verifier runs a protocol between client side (user's computer) and the web server. The APIV Client sends a request to the web server and gets a time token in return.

```
Authenticate(url=https://example.com/APIVServer, username=abcdefghk,
```

> `password=123456789)`
>
> The username and password are encrypted with a secret key `K`.
>
> Upon the success of authentication, the APIV Server generates a time token `T=GenerateToken()`
>
> `T` is unique as it is randomly generated based upon the time of generating the token.
>
> Returns: time token `T=987654321`
>
> Otherwise, returns an error message “Connection fails” or “Authentication fails”

The APIV Client then gets the permanent MAC address of the computer and encrypts it using a private key cryptosystem. The key is a session key which is combined from the secret key and the time token.

> `S=CreateKey(T, K)` where the session key `S = XOR (K,T)`
>
> `EMAC=Encrypt(MAC,S)` results in the encrypted message `EMAC` of the MAC address using the session key `S`.

The encrypted message of the permanent MAC address is then sent to the APIV Server for verification in each API call. The APIV Server decrypts the message to get the MAC address and compares it with the existing MAC address associated with the user.

> At the server side:
>
> `S=CreateKey(T, K)` where the session key `S = XOR (K,T)`
>
> `newMAC=Decrypt(EMAC, S)` where `newMAC` is the value obtained from decrypting `EMAC` received from the APIV Client
>
> `compareMAC = CompareMACaddress(newMAC, storedMAC)` where `storedMAC` is the existing MAC address stored in the server database.

If they match, it means that, the user uses the same computer as before and the verification is passed and the API call is through. Otherwise, the user is working from this computer for the first time. Consequently, the API Verifier will present a Captcha to the user for verification.

> `If compareMAC==True` , APIV Server returns a success message and the API call is through.
>
> `If compareMAC==False, url=GenerateCaptcha()` and APIV Server returns to APIV Client the `url` of the Captcha object.
>
> The `GenerateCaptcha()` function involves other functions to create a Captcha such as image selection and distortion, question generation and so on. For more information, please refer to the function list of APIV in Table 8.1.

If the user is a human being, they will be able to solve the Captcha and pass this verification stage. The API Verifier will then update user's info with the new MAC address so that next time when the user uses the same computer, they will not encounter another Captcha.

> `If VerifyCaptcha(CaptchaAnswer=xyz) == True, updateMACAddress(username, newMAC)`
>
> and the API call is accepted.

If the user is a bot, it will not be able to pass the Captcha verification, hence, it will not be allowed to make any API call to the web service.

4.4 Security Analysis

There are three main attacks as shown in Figure 4.3.

1. Spoofing MAC Address: this attack can be done in two places: (a): changing the MAC address of the computer before the APIV Client gets the MAC address from the operating system (OS), (b): changing the encrypted MAC address when it is being sent from the APIV Client to the server.

2. API Verifier Client Fraud: the attacker may use his own program to imitate the APIV Client functionalities and make API calls to the server.

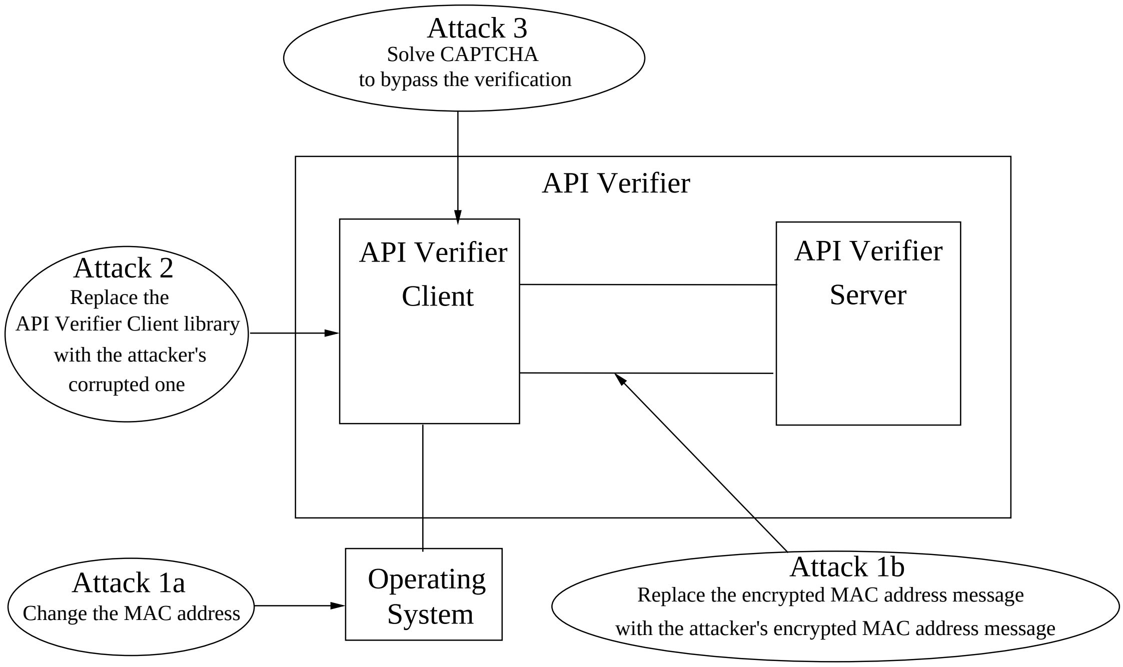


Figure 4.3: Model of Possible Attacks

3. Bypassing Captcha Verification: the attacker may use an automated program to solve the Captcha to bypass this verification stage. Another way is to send a copy of the Captcha to some people to solve the challenge for them.

4.4.1 Spoofing MAC Address

The API Verifier needs to be able to guarantee that the MAC address cannot be spoofed by an attacker, otherwise, she just needs to use the same MAC address for every bot, hence bypassing a Captcha challenge. As mentioned above, the MAC address can be changed in two places.

Changing the MAC address of the computer

It is true, MAC address can be changed quite easily. However, that MAC address is only the current MAC address, which is stored in the OS registry. The permanent MAC address is the code burned in EEPROM of the device and cannot be changed.

Therefore, the APIV Client will only use the permanent MAC Address.

To spoof the permanent MAC address, the attacker needs to hijack the OS kernel and modify the way the OS communicates with the ethernet card, which is expensive and should be detected by firewall or other anti-virus software. Unlike other type of malware, bots' intention is not to destroy or completely take over the computer. Bots rarely announce their presence, instead they infect networks in a way that escapes immediate notice [121]. For bots, the less active they are the better. They silently reside on victim computers, steal data and wait for commands from the botmaster. Therefore, bots rarely hijack the OS kernel because of the high risk of being detected. In the case that the bot is able to take over the kernel, there is no point of using Web 2.0 service as a means of communication because the attacker already completely "owns" the victim's computer and there are other simpler and efficient ways of communication without being traced.

Because the main scope of this study is to address the communication of bots via Web 2.0 service, another research in OS field is required if we want to completely address the matter of how to protect OS kernel from being hijacked or if we want to implement a function to detect when the kernel is compromised by a bot and cannot return a trusted permanent MAC address. In this study, we assume that OS kernel is not taken over by the bot and returns the correct permanent MAC address.

Changing the encrypted MAC address

In this scenario, the bot will replace the MAC address encrypted by the APIV Client (message A) by the desired one (message B) while it is being sent to the server for verification. To protect our system against this attack, we employ an encryption with a session key. Note that using a secret key alone cannot prevent this type of attack because the attacker can use the encrypted message of her desired MAC address for every bot, without the need of knowing the secret key. To get a "valid" encrypted message of a MAC address, the attacker can process as follows:

- use a computer in a public Internet service to make an API call to the server,
- sniff the traffic between the computer and the server,
- analyze the sniffed traffic and get the whole encrypted message of the MAC address (message B) of the computer.

After infecting a victim's computer, a bot sends message B to the server for the MAC address verification. The server will decrypt message B to get the original MAC address. Because the attacker has already passed MAC address and Captcha verification when she used the computer in the public Internet service, that MAC address was already stored in the server database associated with the username. Consequently, her bots will pass the MAC address verification too and are able to post information to the website.

To defeat this attack, we use a session key, which is a combination of the secret key and a time token returned by the server, to encrypt the permanent MAC address. As the attacker does not know the private key and the time token is random each time, the session key is unique for each API call. As the results, the encrypted message of MAC address is different every time. Consequently, the attacker will not be able to spoof the MAC address using the sniffed message B.

4.4.2 Protection of API Verifier Client and the secret key

The second possible attack is to replace the APIV Client with the attacker's own program (bot) that can mimic all API Verifier Client functionalities. Before sending out the bots, the attacker makes an API call to the server, solves the Captcha challenge so that her desired MAC address will be saved in the server database. After infecting a victim computer, the bot makes API calls to the server using the attacker account credentials. When the server asks for MAC address verification, the bot sends the attacker's MAC address rather than the MAC address of the victim computer. Because the MAC address sent from the bot matches the one stored in the user's profile, the

API Verifier will not challenge the bot with a Captcha. Hence, to avoid this attack, we need to ensure that the APIV Client is well guarded against the attackers who may try to replace it with a corrupted one. The API Verifier Client is considered to be secure if the following requirements are satisfied:

1. it is hard to guess/recover the secret key of the APIV Client
2. it is hard or impossible to disassemble the APIV Client (reverse engineering)

For the requirement (1), we need to carefully choose the key length and encryption algorithm so that it is secure as well as does not heavily affect the system performance. Therefore, we use AES (Advanced Encryption Standard) with a 128 bit key. We can always increase the key length to improve the security of the key.

For the requirement (2), we can use obfuscation to protect the API Verifier from being decompiled and to hide the secret key used in the APIV Client. Obfuscation is a technique of making a content very hard to read and understand. Although obfuscation is not provably secure, in practice, it makes the reverse engineering task very costly in terms of time and effort [122]. This technique has been used by many computing leaders such as Microsoft, Oracle and Sun to protect their software. In addition, some coding languages are more prone to obfuscation than others. C, C++ and Perl are most often cited as easily obfuscatable languages [123]. Therefore, a good choice of coding language and an obfuscation tool can improve the protection of API Verifier.

4.4.3 Bypassing the Captcha Verification

Another attack is to bypass the Captcha verification by using one of the following techniques:

1. targeting victims that have already passed Captcha verification, then using the victim accounts rather than the attacker account.
   The attacker targets victims who have passed Captcha verification. The attacker then uses the victim accounts. When the bot calls API using the victim account

and information, it will not be asked to solve Captcha again. However, the information stolen from the victim is useless to the attacker because the information is posted to the victim blog rather than attacker one. The victim will also easily figure out that her computer is infected by a bot once they find an unexpected entry on their blog.

2. solving the Captcha by automatic recognition program, i.e analysing the picture and extracting characters/figures on the image.
   Mori and Malik [124] argue that their algorithm for solving text-based Captcha image is able to identify the text with a high rate of accuracy. However, we can always make a text Captcha harder to be analysed automatically by introducing extra features that make the recognition less efficient. Some popular extra features are distorting the characters of the Captcha string on the image and making segmentation difficult by introducing angle lines on the string. We can make Captcha harder to break by asking the user to solve some simple logical problems, e.g, $\sqrt{25}$ or "Only enter characters in <random color> (e.g red)" or "Only enter the first character of each word" so that the user needs to think rather than only look and read to gain access.

   Besides, we can improve Captcha security by using image-based Captcha instead of text-based Captcha. Image-based Captcha is believed to be much harder to be analysed by automated program [125–127]. So with a clever choice of questions and using image-based Captcha, it will give the bot a hard time to bypass the challenge. In addition, with RIA (Rich Internet Application) being very popular nowadays, advance techniques such as animation Captcha and Drag-and-Drop Captcha are widely supported by many browsers, hence, can be effectively used to enhance Captcha security strength. We will further discuss about these techniques in Chapter 5

3. sending the Captcha to someone (human-being) to solve the Captcha challenge

for the bots.

(a) If the bot sends to the botmaster the Captcha, the bot will definitely get the answer for the Captcha challenge. However, this violates one of the most important rules of the Botnet 2.0 in which the bots in the botnet must not directly communicate with the botmaster; otherwise, her identity can be traced back. In addition, a user only has a given fixed and short time (for example two minutes) to solve a Captcha. Therefore, this attack requires the attacker to monitor each bot and support them on time, which is impossible if a botnet consists of thousands of bots.

(b) Another way is that the bot could send the Captcha to some people to help solving the challenge. The botmasters can lure some unsuspecting users or can hire cheap labours to solve the Captcha for them. This is called a relay attack. Unfortunately, almost all current Captcha designs are broken under the relay attack. There are only few Captcha forms intentionally designed to address the relay attack, however, they are either impractical or still fail in some scenarios of the relay attack. In Chapter 5, we will propose a new design of Captcha which is immune against this attack.

API Verifier can also actively detect the relay attack by monitoring the network traffic right after the Captcha is displayed. If at that time, a high load of sending package is detected, a suspicion is alerted and the API Verifier will generate and display another Captcha. After three attempts, the API Verifier will terminate and only allow another attempt in a certain amount of time. Note that the API Verifier will not always monitor the network traffic, this action only happens when a Captcha verification is required, which is only when the user first time use the computer to call API to the web service. However, there is one downfall is that the detector may produce false alarm if there is legitimate heavy network traffic happening at the same time.

4.5 Usability and Possible Enhancement

API Verifier is user-friendly because it is very easy to use and do not burden users with Captcha verification as it is only required once for the first use. In particular:

- **API Verifier is easy to setup:** only need to download and install the APIV Client library. The users who know how to use API to automatically update their Web 2.0 accounts, should be able to download the software and install it without major difficulty. For convenience, the software may be supported by an installation wizard.

- **API Verifier is easy to use:** APIV Client has a GUI component to display Captcha and messages so it is easy to understand and follow the verification process.

- **Users only need to verify themselves once for each computer used:** the user only need to solve a Captcha the first time they use a specific computer to make API call to the Web 2.0 server. They will never be asked to solve a Captcha again as long as they use the same computer to update their contents.

- **Protection of privacy:** some users may have a concern of giving away their permanent MAC address, though this information can be easily sniffed from their Internet traffic. API Verifier addresses the privacy issue by using encryption. The users data (including the MAC address) is never transmitted in clear form by the verifier. Additionally, the MAC address is stored using cryptographically strong hash algorithm.

There are also some aspects that need to be addressed to increase the tool usability.

4.5.1 Multiple MAC addresses

Sometimes, users may wish to access several computers to automatically post entries to the Web 2.0 sites. So allowing one MAC address per account may cause inconvenience to the users. The API Verifier can be customized to support multiple MAC addresses for one user account. For example, users are allowed to use up to three MAC addresses for the same account. When the user uses the fourth computer to call API, the user will be asked to solve a Captcha. If they pass the verification, the new MAC address will replace the oldest MAC address associated with the user account in the database. Supporting a small number of MAC addresses per account should not have a negative impact on the security of our solution. Although a number of bots may attempt to share a single account, every time a new bot (with a new MAC address) contacts the account, an appropriate Captcha verification procedure takes place. As a botnet usually consists of thousands of bots, it still requires thousands of accounts and Captcha verifications that the attacker cannot afford.

4.5.2 Support Multiple Platforms

A user may use different devices to make API calls to the web services. For example, a user may use a PC at home and a tablet or mobile phone while commuting. Therefore, to improve usability, API Verifier must have multiple versions to be compatible with as many platforms as possible. This requires further research and significant efforts for implementation because each platform requires a different method to extract the permanent MAC address of the machine. This becomes a challenge as far as mobile phones are concerned because they use a large variety of operating systems. Our solution, API Verifier, is implemented for Windows platforms as they are the most popular operating systems used for personal computers. The implementation and experiment will be described in Chapter 6.

4.6 Conclusion and Possible Future Study

In this chapter, we have proposed a novel approach to prevent bot-masters from using Web 2.0 as a C&C communication channel and a temporary storage for the stolen information. Recall, Bot 2.0 abuses technology provided by Web 2.0 services (such as RSS and API) to create a new way of communication which is hard to detect and shut down. With the normal HTTP digest authentication as shown in Figure 4.4 [128], Bot 2.0 can easily bypass the authentication using the stolen username and password. After that, using the provided API, the bot is able to post messages and/or malicious material to the victim's profile as well as to infect other users in the victim's friend list.

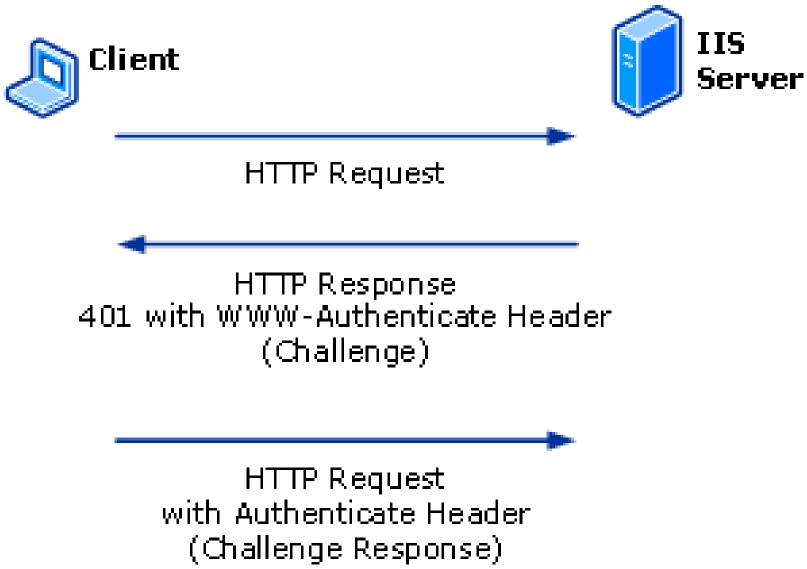


Figure 4.4: Overview of a HTTP Digest Authentication Scheme

Comparing with the current HTTP authentication, our API Verifier can provide a better defence against Bot 2.0 because of the following reasons.

- The API Verifier uses MAC address as an additional credential rather than only username and password to authenticate a user. As MAC address is globally unique, the API Verifier is able to identify if the user uses this computer the first time to make a request to the Web service via API. If it is the first time, the user must answer a Captcha challenge that a bot cannot solve. Therefore, the API Verifier can filter out a majority of bots' requests made to the Web service.

- The API Verifier only accepts API calls from the genuine APIV Client which contains a secret key. In a normal API call (such as those from Facebook and Twitter), an access token is required but it is easy to obtain one and the user is asked to keep the token secret by themselves. Meanwhile, our APIV Client is implemented in such a way that the reverse engineering of its code is impossible. We choose to use obfuscation as described in Section 4.4.2. Hence, it is hardly possible for attackers to make an APIV Client fraud. In other words, the secret key is likely more secured and the MAC address is hard to be spoofed by a bot.
- The API Verifier uses session keys, which are randomly unique each time and unknown to the bot, to encrypt the users' credentials. As a result, even a bot knows the user's credentials, the request made to the Web service is denied as the credentials are not encrypted with a correct session key.
- The API Verifier is also designed so that it is immune against possible attacks including the MAC spoofing attacks and the attacks in which the Captcha verification is being bypassed.

However, there is one downfall is that there is a large diversity of platforms used by users so that it requires many different methods to query the permanent MAC address. As a result, to gain high usability, API Verifier needs to be designed and implemented in many versions to be compatible with platforms that are on the market.

Although this is not a complete solution yet, we believe that the API Verifier is the first system that is able to detect Botnet 2.0 communications. Furthermore, the API Verifier can also prevent Web 2.0 bots in general from abusing the service. This will be discussed further in Chapter 6, where we describe our experiments on a popular existing Web 2.0 botnet.

The key idea behind our approach is to use both the unique and permanent identifier of the computer and the Captcha tests to tell apart a computer program from a human being. A possible future investigations may explore other options for these features.

4.6.1 Alternatives for Computer Identifiers

TPM is a tamperproof device that can be used to store security sensitive information such as cryptographic keys, passwords, digital signature certificates, etc. TPM also provides a suit of cryptographic and security algorithms. TPM stores a 2048-bit RSA public/private key pair called the endorsement key (EK). The key is generated at random during the manufacturing time and burnt into the circuitry. It cannot be changed during the life-time of TPM. EK is unique for each TPM. The secret part of EK is not known to anybody (except TPM and manufacturer). The public part of EK can serve as an identifier of the computer with the installed TPM as (1) it is unique for the computer at least as long as the TPM remains as a part of it and (2) it cannot be changed unless the TPM is replaced. This obviously is possible but the attacker needs a physical access to the machine. If we assume that the kernel and operating system (guarded by TPM) of a computer is secure against external software attacks, then we can allow our API Verifier to use the EK public key of TPM as the unique identifier of the computer. Additional advantage of using TPM is that the secret keys used to protect communication between the computer and a Web 2.0 server can be generated and stored within TPM. This significantly reduces an overhead needed for key management.

Although a majority of computers (especially high-end ones) are equipped with TPMs, the usage of TPMs is not popular. This is mainly due to the fact that there is a lack of a user-friendly graphical interface that would allow users and programmers to interact with TPMs. To our best knowledge, the user still needs to enable TPM via set up options during the start up stage of the computer, which is not always feasible for an average computer user.

4.6.2 Alternatives for Captcha

Captcha is not the only way to tell human being and computer program apart. To verify a user is not a bot, they can use their mobile phone to receive a verification code from the server via SMS. Once the verification code is correctly entered, the user is confirmed as a genuine user. This method works because the attacker cannot steal every user's mobile phone number to receive the verification code. However, mobile phone number is sensitive information that not many users are willing to give away to get a free email or social network accounts. In our opinion, SMS verification should be applied in those services that need very high security and are important to the users such as banking.

Another alternative is using human intervention. In this case, the user is asked to send a short request email to the person handling the website. That person can then verify if the sender of the email is a human. Although this option can be very effective, it is not user friendly as the verification takes a much longer time to complete.

5

Captcha Enhancement

Captcha is a crucial component of our API verifier. It is essential to employ a good quality Captcha solution that guarantees that bots will be told apart from human beings with very high probability. There are many solutions to choose from. However, there are only a few that are resistant against the so-called relay attack. The attack is a variant of man-in-the-middle attack, where an attacker (a program, in our case a bot) forwards a Captcha challenge to a human being so they can solve it on its behalf. The available Captcha solutions that are resistant against the relay attack are too complicated to be practical. In this chapter, we are going to propose an improved Captcha design that combines image animation together with a few sessions of drag-and-drop interactions.

5.1 Background

Expansion of the Internet has created new applications and services that are accessible anytime from anywhere by anybody. The dark side of the Internet advancements is that the ability of potential attackers has increased considerably. The major threat comes from malware that causes multi-million losses. One of the major challenges in a fight against malware is an ability to identify if a user who asks for access is a real person or a computer program (malware). In 2000, researchers from Carnegie-Mellon University [129], came up with a solution. The solution is called Captcha, which stands for Completely Automated Public Turing test to tell Computers and Humans Apart.

In the original Turing test [130], a human judge is allowed to ask questions to two players. One of them is a computer program and the other is a human being. The task of the judge is to identify them correctly. Captchas are similar to the Turing test in that they distinguish humans from computers, but they differ in that the judge is now a computer. A Captcha is public Turing test in the sense that its code and the data used by it should be publicly available. This requirement is crucial from a security point of view. It should be difficult to write a computer program that solves Captcha challenges even if the details about the design of the Captcha software are publicly known [129].

A typical Captcha challenge can be an image containing distorted characters on a web registration form. The user can complete the registration only if he has entered the correct text displayed in the Captcha. Assuming that it is difficult to design a computer program that can read distorted text, the Captcha can be applied to tell apart human from program.

By definition, the security of a specific Captcha solution fluctuates and very much depends on the balance between new features of Captcha that are not easily identified by the known pattern recognition algorithms. One can see Captcha design and analysis as the fight between proverbial sword and shield. Because of this nature, Captcha is not applied for high security applications. In spite of its limitations, Captcha is the main and practical tool for detection of malware. Interestingly, Captcha is applied in the following applications.

- **Free Website Registration**. Nowadays, Captcha has been extensively used in almost every registration form on websites such as free email, forum, blog and social network registrations. These free services previously suffered from auto-registration attacks from bots. Without being able to distinguish from human users and bots, these free email services were abused and became a tool for spamming. In particular, bots signed up for thousands of email accounts every minute and used these accounts to spread spam (ads and phishing). The abuse has been reduced considerably by requiring users to solve Captcha challenges. This example clearly illustrates the main point of using Captcha. It is still possible to register a fake account but this needs to be done manually by a human and the registration cannot be done on a massive scale.

- **Stopping Spam**. Captcha may also offer a partial solution to spam. For example, the owner of a mailbox may choose to accept an email only if there is a human who sends it. In a such system, the sender contacts the recipient mailbox first and solves a Captcha challenge and then is allowed to forward the email. This kind of service is already on offer by some companies, see for instance, `www.spamarrest.com`.

- **Online Polls**. In most polls, IP addresses of votes are recorded in order to prevent a single user from voting more than once. This precaution is not sufficient when polling stations are attacked by bots. Each bot can subsume a different IP address and dramatically change the result of the poll. After introduction of Captcha the attack from bots is eliminated although a keen voter can still cast multiple votes from different machines. The change, however, to the result of the poll is going to be negligible.

- **Preventing Search Engine Bots and DDoS Attacks**. Some websites do not wish to be listed on search websites such as Google or Yahoo. Though such a request is usually respected by big companies like Google and Yahoo, a website owner can ensure that their websites are not indexed by a computer program. This can be implemented by asking the user who requests the access to a website to solve a Captcha. Similarly, Captcha is being used to control access to blogs and forums. Any guest who wishes to add a comment/entry to a web page has to pass a Captcha verification. Note that a distributed denial of service (DDoS) can be mitigated by Captcha.

- **Preventing Dictionary Attacks**. Dictionary attacks are a serious security problem especially if users tend to choose insecure passwords. Insecure, in this context, means that passwords are simple compositions or words. One of the ideas to prevent dictionary attacks (apart from choosing strong passwords) is to slow down the attacker. Every new guess of the password starts from solving a Captcha.

5.2 Captcha Evolution

As expected, designers of malware (spammers) quickly adapt to advances in anti-malware technology that includes Captcha verification. Over time, many variants of Captcha have been developed. The variants, however, exhibit the following six main schemes.

5.2.1 Text-based Scheme

This is a very early scheme employed by the Captcha designers but also the most popular, especially for low security applications. A text-based Captcha is an image that contains both characters and digits displayed in a such way that is easy to identify by a human but difficult by a character-recognition program. The text is displayed using variety of transformations such as rotation, deformation, distortion, division and shifts [131, 132]. Note, however, that an over-distorted text leads to mistakes and consequently, a human being can be classified as a computer program and denied the access.

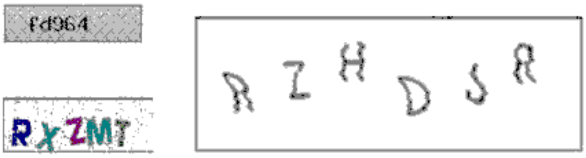


Figure 5.1: Typical EZ-Gimpy Captchas

Early Text-based Captcha

Around the year 2002, Yahoo used these early versions of Captcha to prevent computer programs from opening new accounts. Some further security enhancements included higher contrast between background and text to increase readability and changes in font-type display. Initially the EZ-Gimpy Captcha worked well but by the end of 2003, the next generation of bots and malware was able to read the text. Figure 5.1 show some samples of early Captcha.

Modern Text-based Captcha

To overcome the weakness in EZ-Gimpy Captcha, modern text Captchas focus less on the background noise but more on making segmentation difficult. In other words, a character-recognition program should not be able to identify individual characters or digits from the displayed text. This is accomplished by adding an curve line that crosses the text line or/and cramping symbols closer together. One of the most common versions of modern Captchas is Google's reCaptcha shown in Figuer 5.2. While this type of Captcha renders simple spam software useless, advanced pattern-recognition

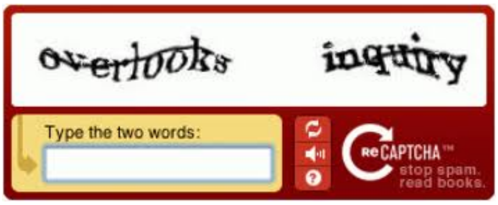


FIGURE 5.2: Modern Captcha: reCaptcha

programs are able to bypass this type of Captcha with a moderate rate of success [133].

5.2.2 Image-based Scheme

Image-based Captchas show a collection of images to users and ask them to identify those that hold some specific properties [134]. A typical image-based Captcha usually consists of 4 or more images for which users need to find the similarities of all images or to sort the images into categories or to pick a particular figure. A popular example of image-based Captcha is Asirra (Animal Species Image Recognition for Restricting Access), developed by Microsoft in 2007. An Asirra challenge consists of 12 images, each of which is of either a cat or a dog taken from `Petfinder.com` image database. To solve the Captcha, the user must select all the cat images, and none of the dog images (see Figure 5.3). At that time, it was believed that Asirra was considerably secure thanks to significant large image databases (over three million images of cats and dogs provided by `Petfinder.com`). However, in 2008, Golle [135] proposed a classifier that achieved a high rate of cat and dog image recognition which allows to break instances of Asirra. More precisely, the classifier is able to give correctly answer in 1 out of 10 cases.

Clearly, the image-base Captcha is an improvement in terms of not only friendliness and usability but also the security level may be much higher from the one achieved in the text-based variants. The designer of image-based Captcha should be aware that the interpretation of the question and the features asked to be identified may vary from person to person depending on the cultural differences between them.

To maintain a reasonable level of security, the database with pictures should be sufficiently large and the same picture should not be on display more than once. To implement the second requirement, it is enough to let Captcha randomly distort the picture on display. These two requirements must be satisfied to prevent the Captcha against the following simple attack. The attacker calls the Captcha system many times and manually classifies the pictures. She stores this information in her shadow database, which can be used by a program to correctly response to the Captcha.

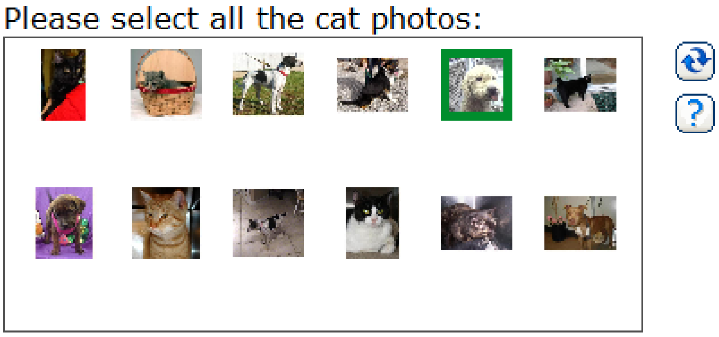

Please select all the cat photos:

Figure 5.3: Image-based Captcha: Asirra Captcha

5.2.3 Audio-based Scheme

Audio Captchas use a sound instead of a picture and are an alternative for users with eyesight disabilities. A typical audio Captcha session goes as follows. The program chooses randomly a word or a sentence of digits or a combination of the two together with an appropriate sound clip. The clip is next distorted by adding noise to it and is sent to the user as a Captcha challenge (see Figure 5.4). The user listens to the clip and enters the identified sentence. A drawback of audio Captchas is that non-native language speakers may have problems with understanding the clip. Bursztein et. al. [136] conducted an experiment in which 3 participants solved a number of audio Captchas. The results showed that 3 participants only gave the same answers for 31 out of 100 audio Captchas. This could be the reason why the audio Captcha is not very popular.

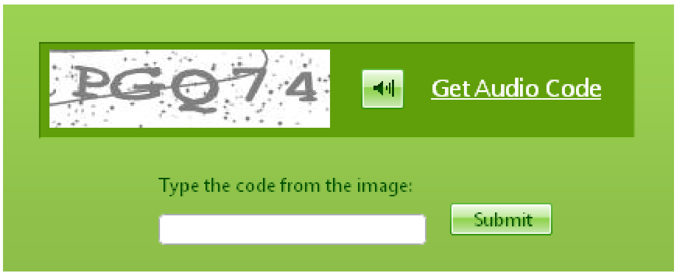


Figure 5.4: Audio Captcha as an alternative for text Captcha

5.2.4 Animated Captcha

This is a relatively new Captcha development, where the user is confronted with an animation or video and has to answer a question [125, 126, 137]. Animated Captcha is much harder to solve as the objects are moving around. A typical implementation uses an animated gif file, which contains a sequence of frames. A single frame is not enough to solve a Captcha but when the frames are combined, then the solution is easy to be found by a human. An example of such Captcha is given in Figure 5.5. The solution is “j46c” that is always on display while other characters and digits are moving in the background. To increase security, Captcha may use strings or pictures that are spread over many frames. Despite the additional complexity, there are pattern-recognition algorithms that are able to crack the Captcha. Some user machines do not support the animation and video display.

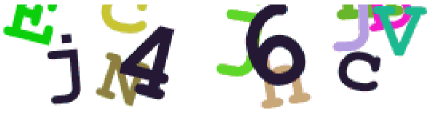
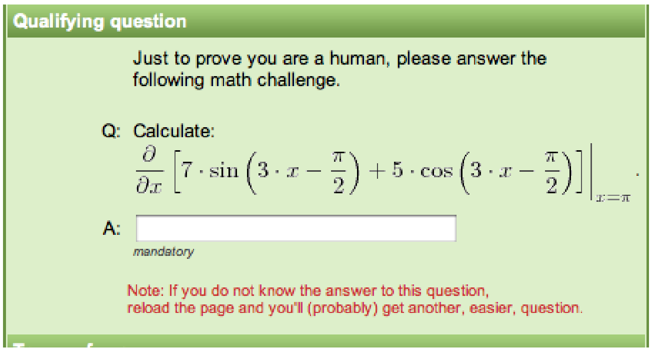


Figure 5.5: Animated Captcha

5.2.5 Problem-solving Captcha



Qualifying question

Just to prove you are a human, please answer the following math challenge.

Q: Calculate:

$$\frac{\partial}{\partial x}\left[7 \cdot \sin\left(3 \cdot x - \frac{\pi}{2}\right) + 5 \cdot \cos\left(3 \cdot x - \frac{\pi}{2}\right)\right]\bigg|_{x=\pi} .$$

A:

mandatory

Note: If you do not know the answer to this question, reload the page and you'll (probably) get another, easier, question.

Figure 5.6: Problem solving Captcha

In these Captchas, a user is required to work out a solution to a problem rather than to recognise an object or text. A problem solving tasks may take on any imaginable form, starting from a simple question like "How many days in a week?", solving algebraic equations, to complex cross-word puzzles. Problem-solving Captchas appear to be more secure as the tasks are more "human" oriented while designing a program for finding solutions seems to be very difficult or perhaps impossible.

A main drawback of problem-solving Captcha is that they are not user friendly. If a Captcha instance presented to a user is too complicated (such as the question shown in Figure 5.6), then most users are likely to decide not to try to solve it. Another practical difficulty is the creation and management of database with problem-solving questions. If the database is small, then the Captcha can be broken easily. For large databases, it takes a lot of time and resources to create questions and answers.

5.2.6 Drag and Drop Captcha

Drag and drop (DnD) Captcha asks a user to select an item, drag and drop it to a specific place [138, 139]. A successful execution of such action is considered to be a proof of human intervention assuming that a computer program is not able to mimic the action. Figure 5.7 presents a typical DnD Captcha. However, it is hard but not impossible for a program to move an object without using mouse events. For the Captcha shown in Figure 5.7, a program needs to randomly move one of the four objects to the answers area. The success rate of such attack is 25%. To prevent this kind of attacks, the number of objects should be increased. An increase to 20 would reduce the success rate of the attack to 5%. Alternatively, a solution to Captcha should involve a drag and drop of multiple objects (perhaps in a strictly determined order). However, this is going to increase the time necessary to answer the challenge.

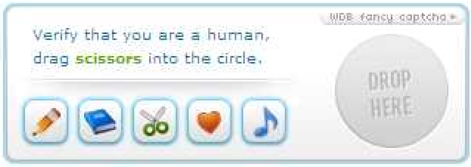

Verify that you are a human, drag scissors into the circle.

Figure 5.7: Drag and Drop Captcha

5.3 Motivation

Over the years, miscreants have devised techniques to break any newly designed Captcha, which leads to an endless competition similar to the one between proverbial sword and shield. Designers come up with new advanced Captchas while the attackers devise new ways of breaking them. In the response, Captchas are upgraded by new features and ideas and so forth. To keep a sufficiently wide security margin between designers and attackers, the Captcha systems need to be constantly improved. Note also that the attacker software needs both to learn instances of Captchas as well as to find correct solutions. This task may not be easy especially if Captchas instances are constantly modified (probabilistically or deterministically).

Recently, a new and very powerful attack has been reported in [140–142]. The attacking program outsources the task of solving Captcha instances to a group of people. One can dispute if this is a “real” attack as by the design principle, Captchas should be easy to solve by a human being. It is called the relay attack and is a type of man-in-the-middle attack. A workflow in the attack is illustrated in Figure 5.8.

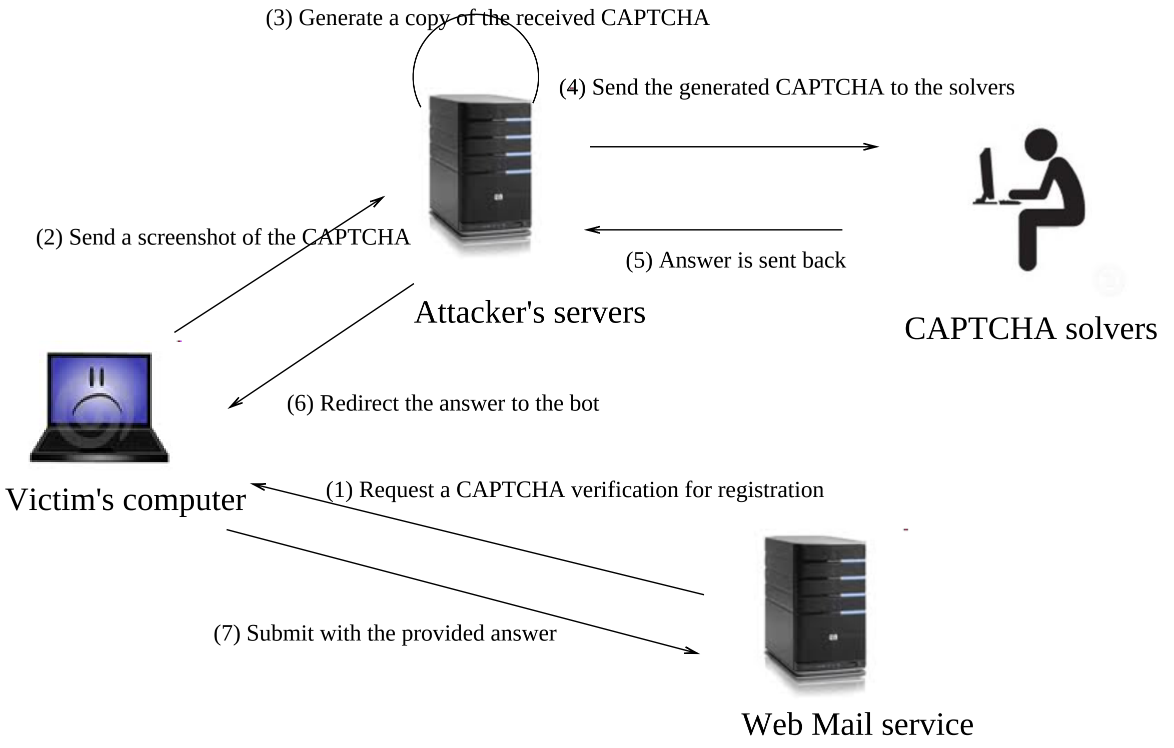

(3) Generate a copy of the received CAPTCHA
(4) Send the generated CAPTCHA to the solvers
(2) Send a screenshot of the CAPTCHA
(5) Answer is sent back
Attacker's servers
CAPTCHA solvers
(6) Redirect the answer to the bot
Victim's computer
(1) Request a CAPTCHA verification for registration
(7) Submit with the provided answer
Web Mail service

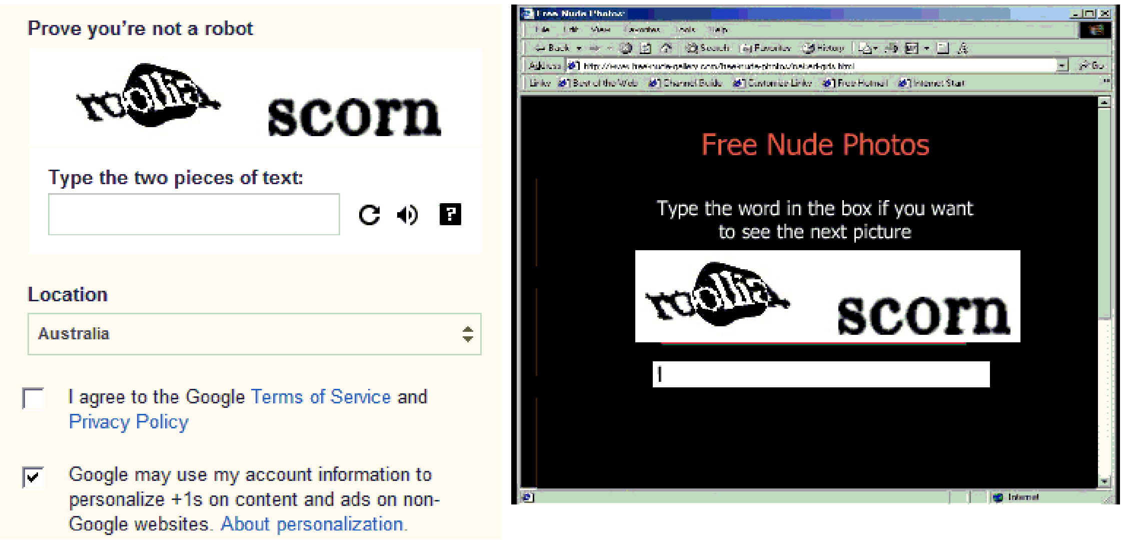
Figure 5.8: Relay Attack on Captcha

There are two ways of getting someone to solve the Captcha challenges:

1. **Laundry Attack**.

   In the laundry attack, unsuspecting users are lured to solve Captcha challenges on behalf of an attacking program. Here it is a typical scenario of the attack. The attacker makes some preparations. The attacker needs to build (or take over) a web site that is going to serve as the lure. Assuming that the attacker has already built such a website, which has gained enough popularity from users (this could be a porn website, for instance). Every time a user attempts to access a porn content, he/she is asked to solve a Captcha. However, the Captcha is not designed by the attacker but is fetched by the attacker from the victim website. The user solves the Captcha and the attacker presents the solution to the victim site.

This attack can break Captcha with a high rate of success. However, this attack may not gain much popularity as its business model is not competitive. Users are likely to prefer to use websites that are not asking Captchas to be solved.



Prove you're not a robot

Type the two pieces of text:

Location

Australia

□ I agree to the Google Terms of Service and Privacy Policy

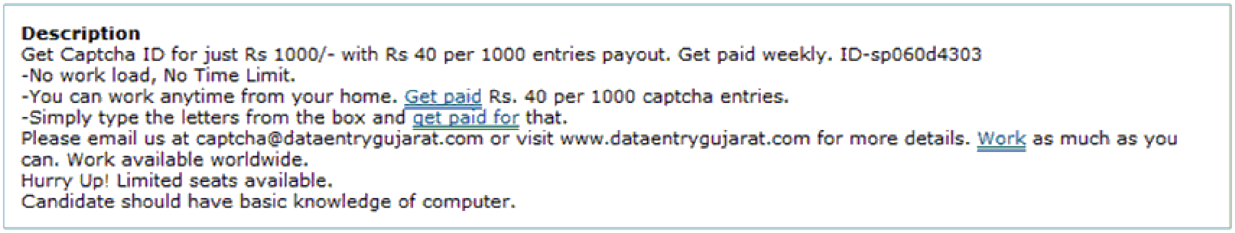
☑ Google may use my account information to personalize +1s on content and ads on non-Google websites. About personalization.

Free Nude Photos

Type the word in the box if you want to see the next picture

Figure 5.9: Laundry Attack on Captcha

2. **Captcha Outsourcing**.



Description
Get Captcha ID for just Rs 1000/- with Rs 40 per 1000 entries payout. Get paid weekly. ID-sp060d4303
-No work load, No Time Limit.
-You can work anytime from your home. Get paid Rs. 40 per 1000 captcha entries.
-Simply type the letters from the box and get paid for that.
Please email us at captcha@dataentrygujarat.com or visit www.dataentrygujarat.com for more details. Work as much as you can. Work available worldwide.
Hurry Up! Limited seats available.
Candidate should have basic knowledge of computer.

Figure 5.10: Captcha Solver Job Advertisement

Instead of luring users to solve Captcha for free, the attackers may prefer to employ cheap labor especially in countries that have a reasonably fast Internet connection. Figure 5.10 [143] shows a recruitment advertisement for a position of Captcha solver. The Captcha solvers are paid \$2.5 per hour and able to solve 720 challenges during that time. Thus a single solution costs around 1/3 of a cent [142].

Design Principles

There has been a concerted and continuous effort to enhance Captcha so it is resistant against the optical character recognition (OCR) attacks and image and audio recognition algorithms (see [125–127, 129, 131, 134, 138, 139] for examples of the extensive literature coverage). Our attention, however, is focused on a very recent threat against Captcha, namely, the relay attacks. In particular, we are going to propose a new design of Captcha that is secure against the relay attack and is guided by the following design principles:

1. Non-transferability of solution of Captcha. The design of Captcha should rely on dynamic features. Note that using static features in Captchas helps attackers to transfer the answer to Captcha challenges between nodes of a malicious infrastructure. Being more specific, we are going to avoid challenges for which answers are *"what"* but rather prefer answers *"where"*.

2. Use of temporal characteristics in challenges. This principle say that a (correct) answer to a challenge changes over time. It also means that challenges may be based on animations and a drag-and-drop answers. The number of animated objects in challenges should be, on one hand, big enough to prevent a random guess attack from having a significant chance of success and on the other hand, small enough so users can generate answers quickly.

3. Reasonable delay time of animation. Setting a reasonable time interval (delay time) between two animations is crucial. If the delay time is too long, the bot and human Captcha solver will have enough time to communicate, hence the bot

can submit the correct answer. On the other hand, if the delay time is too short, the user may be annoyed because the images keep changing their position before they can drag and drop the answer. We should balance the security (against human solvers) and usability.

4. Non-transferability of Captcha challenges. This principle aims to prevent redirection (or hijacking) of Captcha challenges by an attacker. We can implement this principle in different ways. Our solution is going to use unique session IDs. In other words, each Captcha displayed has a unique session ID so that if there is a second request to the server hosting the Captcha with the same session ID, this session will be expired and a new Captcha with a new unique session ID will be generated and displayed in return. Therefore, the forwarded Captcha will be different from the one displaying on the victim's computer.

5.4 Related Works

Animated Captchas with drag and drop responses are not new. There are few studies [140–142] that employ various animation techniques. KeyCaptcha [141], developed by Mersane Ltd, is claimed to provide a powerful anti-spam website protection. KeyChaptcha challenges require users to assemble images from their pieces. An illustration of such challenge is given in Figure 5.11. The user needs to use a mouse to drag and drop 4 pieces of the puzzle into correct places to pass the challenge. It seems to be difficult for a program to perform such tasks. However, they are not easy for users either. Some people may find dragging a puzzle piece into the correct position difficult to do. Besides, these interactions are typically time consuming. The accuracy of alignment required in KeyCaptcha is too stringent resulting in multiple failures.

Captcha 2.0 developed by SiteBlackBox [140] uses animated undistorted text. It is user friendly and provides high usability. To thwart man-in-the-middle attacks, Captcha 2.0 never displays the whole text in one single frame. A user who attempts to solve

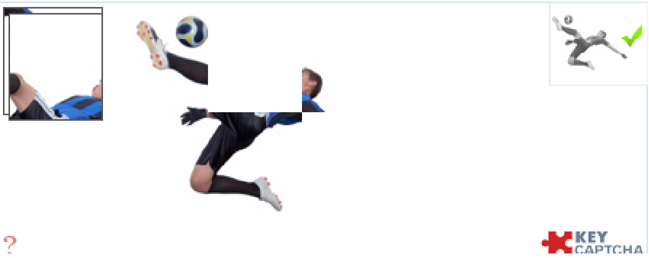


Figure 5.11: keyCaptcha

a Captcha 2.0 challenge must observe all the frames in order to reply correctly. To protect Captcha 2.0 against relay attacks, designers introduced an interesting security measure. The animated gif file is displayed as in the original form while any attempt to store it is going to modify it. However, if a program can somehow record the computer screen, then Captcha 2.0 can be broken by sending the video to a sweatshop.

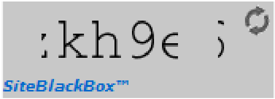


Figure 5.12: Captcha 2.0

Athanasopoulos et. al. [142] proposed a Captcha design that uses animated images and is quite similar to our scheme. However, the authors do not elaborate on the design details such as number of images used, image arrangement, image database and more importantly, delay time, which will be defined in our design. Those properties and components, in our opinion, are very important to realise the scheme in practice. Furthermore, we use drag and drop technique in our solution to enhance security.

5.5 Proposed Captcha Design

The Captcha system consists of two major components: client and server (see Figure 5.13). The server side is responsible for the generation of Captcha challenges and processing the responses of users. The client side accepts challenges, displays them and sends the users responses back to the server. Further in this section, we are going to describe main components of the Captcha system.

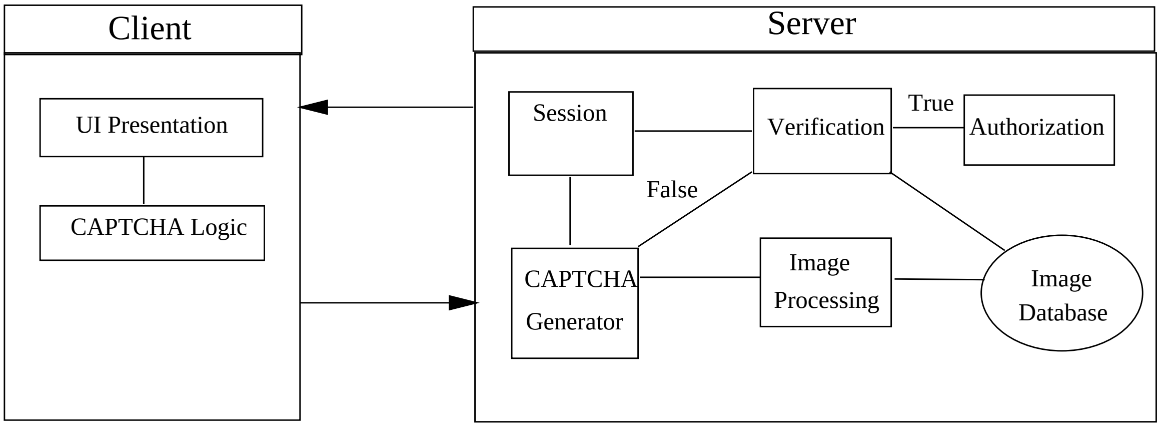


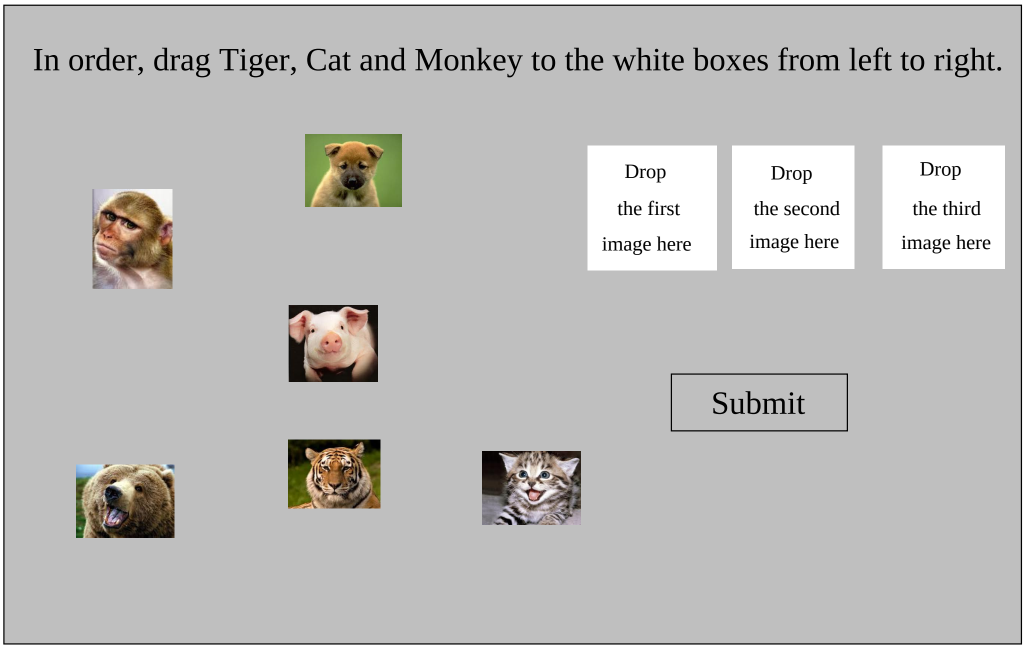
Figure 5.13: Enhanced Captcha: system architecture

5.5.1 Image Database

Images are collected from the Internet and resized to the standard size. The image database should be large and divided into categories such as dog, cat, pig, rabbit categories and so on. The more categories can be constructed, the better security can be achieved. As an improvement factor, the images of these animals or objects should be similar in shape in order to make the automatic image recognition harder. For example, only the head of the animal is shown on the image to reduce the risk of recognizing the animal via its body shape.

5.5.2 Captcha Generator

The Captcha generator is responsible for the creation of a Captcha challenge. The generator can be asked either by a client (when a user wants to access the server) or by a verification component (when a new request is issued by the verification component).



In order, drag Tiger, Cat and Monkey to the white boxes from left to right.

Drop the first image here
Drop the second image here
Drop the third image here
Submit

Figure 5.14: The enhanced Captcha GUI

The process of issuing a new Captcha challenge goes through the following steps:

1. Six images from six different categories are randomly selected from the image database. Three of them are chosen to be the correct answer (e.g Tiger, Cat and Monkey from the example in Figure 5.14). All images are assigned their unique labels (image id) at random. The labels are not stored in the database but generated "on-the-fly" when needed. A single image may be assigned different labels at different times. The number of images used to issue a challenge will be analysed closely later on in Section 5.6.

2. The chosen images are distorted by a noise and/or by adding some extra and random pixels and/or changing the image orientation. This is a crucial step to ensure that an attacker is not able to pre-process images ahead of challenges. The position of inserting the random shapes varies from the center of the images to its edges. Scale of colour adjustment is also randomly varied to prevent the bot classifiers from easily weeding out the noise. For implementation, one can use ImageMagick library [144] to insert the random noise to the image. Such random

noise introduction makes sure that each image appears with different noise levels. However, the crucial part of the image (ie figure of the animal) is distorted by “small” noises so that the figure is not over distorted and is recognizable.

3. The distorted images are embedded in dragable objects and displayed in a random placement (see Figure 5.14). Since our goal is to prevent human Captcha solvers to be able to tell the locations of the correct answers, a random arrangement is chosen to make it more difficult to describe the position of each object.

4. The question and answer boxes are formed. The user is asked to drag the required images in order as stated in the question, such as cat, dog and pig images must be dragged into the first, second and third boxes accordingly as shown in the Figure 5.14. Though this requirement is simple, it will make it significantly harder for brute force attack, which will be discussed in section 5.6.

5.5.3 Captcha Session

Once the challenge images are assembled, the session component selects the session identifier (ID) at random. The contents of the new Captcha challenge, such as the question, the correct answer and its session ID is stored in the session component memory. In particular, the server responds to the client request by sending a new Captcha with a unique associated session ID and waits for the response from the client. The client will send the response back to the server with the received session ID and an answer for the Captcha. The server will verify the answer using this session ID and the received answer. If it passes the verification, the user will be granted authorization, otherwise, the user will be asked to solve another Captcha if their number of attempts still does not reach the limit of attempts. In both cases, either Pass or Fail the verification, the current session, which contains the session ID and the Captcha content, will be discarded. Note, the use of session IDs prevents the redirection attacks as the first answer only (with the correct ID) is going to be considered by server. Next answers (even if correct) with a repeated session ID are rejected by the server.

The session parameters can be adjusted as required. For instance, the duration of sessions can be made as short as 1 minute and as long as a few minutes. If the server does not receive the answer on time the session expires and a new session is created with a new challenge.

5.5.4 Verification Unit

```
Verify() {
    If (sessionExpired=FALSE AND NoAttempt <=maxAttempts AND clientSessionID=sessionID
    AND correctAnswer=TRUE)
        return TRUE;
    return FALSE;
}
while (!Verify()) {
    generateQuestion();
    generateCaptcha();
    correctAnswer=CompareAnswer();
}
generateQuestion() {
    Get 6 random images;
    Label selected images;
    Select 3 labels as answer;
    Generate the question string;

}

generateCaptcha() {

    Randomly distort 6 selected images;

    Store answer (labels) to session;

    Generate session ID for the Captcha and save to session;

    Create drag-able objects;

    Form question and answer boxes;

    Return Captcha object;

}
```

Figure 5.15: A pseudo code of the verification and generate Captcha procedures

The verification unit returns true if and only if the following criteria are simultaneously met.

- The last challenge has been successfully solved. In other words, the user has put required images in the correct drop-off boxes.
- The session ID returned with the answer is the same as the one issued by the server.
- The duration of a session has been within the time limits set for the system.
- The number of attempted challenges has not exceeded the threshold fixed for the system.

Otherwise, it returns false.

A pseudo code of the verification and generate Captcha procedures are given in the Figure 5.15.

5.5.5 Client Side

The user interface (UI) of Captcha consists of two components: the presentation and logic units. The presentation unit gets Captcha challenges from the server and displays them on screen. The logic unit controls the animation (randomly changes the positions and labels of the dragable images) and conveys the user answers back to the server.

Animation Control

The logic unit randomly changes the positions of the images together with their labels every specific time interval. In our design, this interval is 10 seconds. The changes of positions do not have an impact of friendliness of Captcha as long as the changes are not too fast. Apparently, the label changes do not affect a user as the changes are transparent to the user and they still sees the same set of images. Before changing positions of images and assigning new labels to images, the logic unit sends an appropriate request to the server. In response, the server generates a new set of random labels and sends them back to the client (see Figure 5.16). The logic unit on the client side replaces the old labels of images and randomly changes the image locations. The periodical change of labels prevent the automated program from labelling the images so that the solver only needs to tell which labels are the correct answers then it is able to identify the images regardless their positions changes. Note that all messages communicating between the server and client are encrypted using private keys (registration usually requires SSL as well) to prevent bots from sniffing and analysing the message to learn the pair of old and new labels for each image.

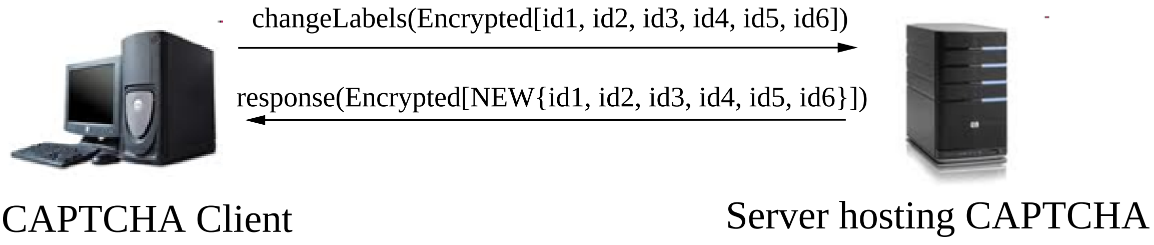

changeLabels(Encrypted[id1, id2, id3, id4, id5, id6])
response(Encrypted[NEW{id1, id2, id3, id4, id5, id6}])
CAPTCHA Client
Server hosting CAPTCHA

Figure 5.16: Client and Server exchanging new labels every 10 seconds

To illustrate the working of the system, consider the Captcha from Figure 5.14. The Captcha challenge contains 6 images of animals (cat, dog, monkey, pig, tiger and bear). The images are assigned the labels A, B, C, D, E and F respectively. After 10 seconds, the positions of images are changed and new labels are assigned. The new labels can be for instance, X, W, P, S, J and A accordingly.

Therefore, if the user solve the Captcha in more than 10 seconds (but less than 20 seconds, otherwise, the images change position for the second times), the correct answers sent to server will be J, X and P (provided the correct animals are tiger, cat and monkey in that order) rather than E, A and C as initial. Assume that an adversarial program/bot (residing on the user machine) applies the relay attack. The bot sends the challenge to a human solver and after some time receives back the answer. If the delay is bigger than 10 seconds, the solution is outdated because the labels have changed. The bot would need to correct the answer, which is probably as difficult as solving the original challenge.

5.5.6 Delay Time

As we indicated in our design principles, it is important to define the time interval after which the position of images changes. If the animation interval is too short, users will need more effort to look for the required images because they keep moving. On the other hand, if the interval is too long, the Captcha can be broken by the relay attack. To find out what is an acceptable animation time, we implemented the scheme and allow real users to test the Captcha.

The Experiment

Delay time

Figure 5.17 illustrates the relay attack and time delays between different phases of the attack. The delays are indexed from t1 to t6 and are defined as follows:

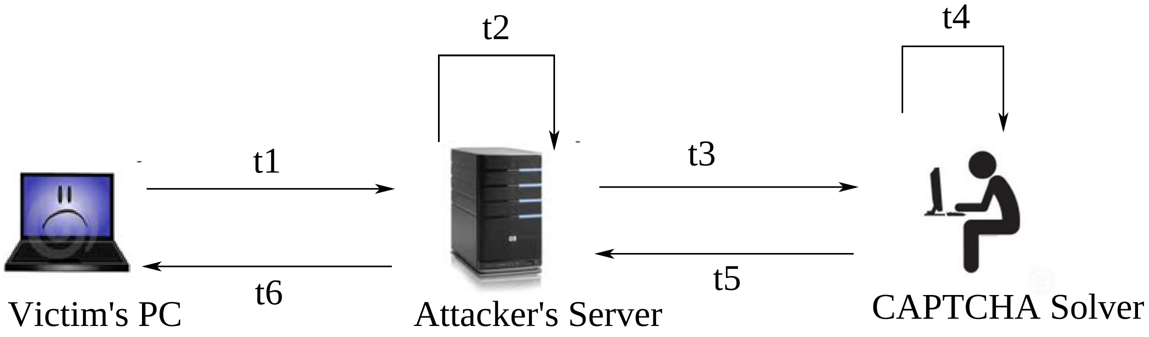


Figure 5.17: Time delays in relay attack

- **t1:** time needed by an adversarial program to capture the challenge and to send it to the attacker server.

- **t2:** time required by an attacker to edit the received challenge. This may be a simple filtering procedure to get only the Captcha image or label the objects on the Captcha.

- **t3:** time necessary to send the challenge to a human solver.

- **t4:** time needed by the solver to produce an answer to the challenge.

- **t5:** time for sending an answer to attacker's server

- **t6:** time for forwarding the answer to the bot at victim's PC

So the total time of communicating between the bot and the Captcha solver is $t = t1 + t2 + t3 + t4 + t5 + t6$. However, we were only interested in $t2, t3, t4$ and $t5$ because of the following reasons:

- The time $t1$ may be quite long but it varies a lot depending on the technique applied by the adversary program that capture the challenge details and process the image. Therefore, we ignored this time interval. This exclusion does not harm the result because it takes longer for a relay attack in reality.

- We also ruled out $t6$ because it is instant for forwarding the answer. It is also simpler to set up the simulation without the care of $t6$ because we do not need to keep track the time at the client side.

Experiment Setup: We try to setup the simulation as close as a real relay attack as possible. The experiment is set up as follows:

- Participants: 10 people.

- PCs: 10 PCs connected to high speed Internet.

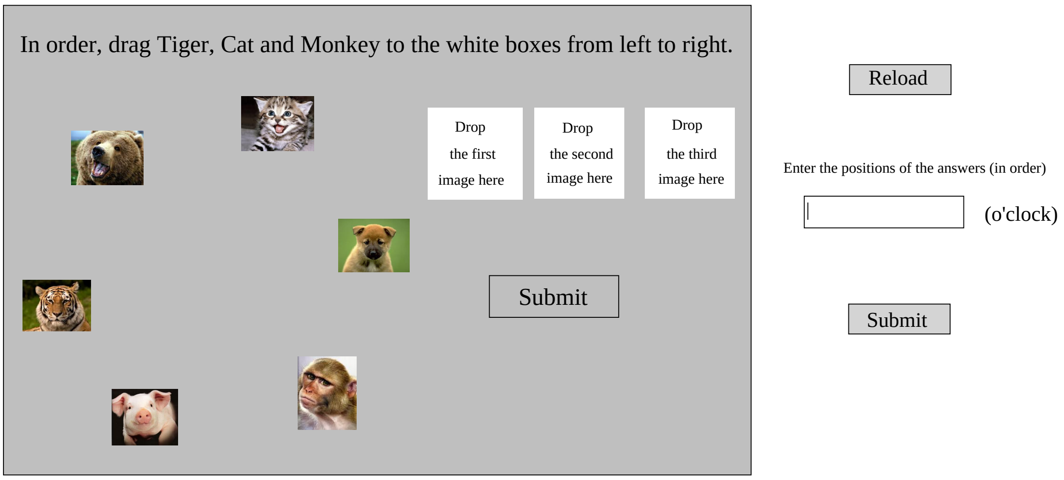
- Location: The PCs are located at the participants' home. The participants are online at the same time to conduct the experiment.

- Server: We set up a server which is responsible for generating an image of the Captcha challenge and posting it on a webpage with a Reload button (see Figure 5.18). Note that in the real scenario, a bot only sends the image of the Captcha challenge and not the challenge itself as our Captcha is resistant against redirection attack.

- Images arrangement on Captcha: To make it easier for the participants, we arrange the images circularly (see Figure 5.18). So the correct answer is 8, 1, 5 (o'clock). In a real scenario, it is harder for the solver to describe the positions as the images are displayed in random placement. In that case, the bot needs to label the object in the Captcha image. For example, cat is labelled 1, pig is labelled 2 and so on.

- Delay time measurement: the delay time is $t = t2 + t3 + t4 + t5$, which is the time from when the server generates a Captcha image to the time an answer reaches the server.

The Experiment



In order, drag Tiger, Cat and Monkey to the white boxes from left to right.

Enter the positions of the answers (in order)

Drop the first image here
Drop the second image here
Drop the third image here
Submit
Reload
(o'clock)
Submit

FIGURE 5.18: Captcha solving page

Following are steps in one round of the simulation:

1. The server generates a Captcha image and post it to the webpage. The server starts timing.

2. Each participant opens the webpage and frequently press the Reload button to retrieve new image.

3. Once an image is displayed, the participants describe the positions of the answers as quickly as possible. The participants enter some numbers to the textbox and hit submit.

4. The server measures the delays for each answer from the participant. Among 10 delay time of the participants, only the shortest time with correct answer is collected.

5. Go back to step 1 for another round of relay attack.

We run the simulation 100 times.

Results

The results after 100 runs are shown in the Graph 5.19. The time is measured in milliseconds. The average time is 12728.02 milliseconds and the minimum time is 11050 milliseconds.

Conclusion

From the results of the simulation, we decide to set the delay time between two animations being 10 seconds. It is considered based on the following assertions:

- 10 second is shorter than the minimum time recorded in the simulation. Therefore, it is likely that a bot fails relay attack on the Captcha as it is not able to get the answer in the relay attack before the objects on the Captcha change their positions and labels.

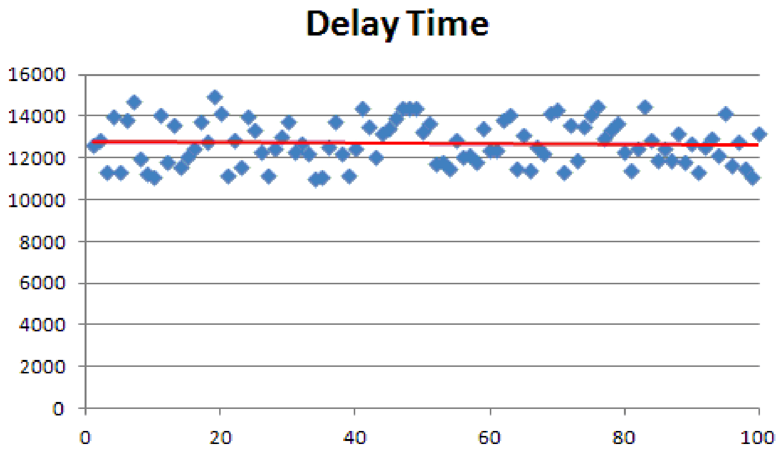


Figure 5.19: Communication time in the simulated relay attack

- An average person is able to recognize and drag and drop one image within around 3 seconds. So it takes about 9-10 seconds to solve the Captcha. Therefore, it is likely that an average person can solve the Captcha challenge before the objects move.

Note that, though this is not popular, it may happen that a bot may try to lure the victim on the same computer to solve the Captcha for it. In this case, the delay time becomes $t = t2 + t4$. In the simulation, we measure $t = t2 + t3 + t4 + t5$ where $t3$ and $t5$ are the time transferring the image and answer between the participant (solver) and the server. Since each participant is connected to high speed Internet, $t3$ and $t5$ are negligible to the total time. In addition, 10 seconds is 1 second below the minimum time in the simulation. So it is acceptable in this special case as well.

5.6 Security Analysis

We defined three major attacks that bot can try to break our Captcha. They are brute force attack, relay attack and image recognition attack. Malicious users may try to exploit insecure implementation such as allowing the re-use of session ID or small fixed pool of Captcha images to bypass the Catpcha. However, we have addressed these issues in the design principles (correct use of session, use of encryption and sufficient large database).

5.6.1 Brute force Attack

In the brute force attack, the adversarial program (bot) tries to guess the classification of images. Then it drags and drops the images to appropriate boxes. A Captcha scheme is considered secure if the brute force attack succeeds with the probability smaller than 5% [145].

Consider our Captcha scheme that uses 6 images and three boxes. Assume that P(a) is the probability of getting a correct answer, i.e, the three required images are correctly dragged into the corresponding answer boxes. Let $P(i)$ is the possibility of getting the ith image to the correct answer box, we have:

$P(a) = P(1) * P(2) * P(3).$

$P(1) = 1/6$
$P(2) = 1/5$
$P(3) = 1/4$

Therefore, $P(a) = 1/6 * 1/5 * 1/4 = 1/120.$
That means less than 1% is the chance that an attacker can get a correct answer using brute force attack on our enhanced Captcha.

It is easy to improve the resistance of our Captcha scheme against the brute force attack by increasing the number of images. However, the more images are used, the lesser usability may be achieved because users may need to spend more time to solve the challenge. Therefore, when choosing the number of images used in a Captcha design, a good balance between security and usability is essential and needs to be carefully considered. In our design, we believe using six images is a good choice. This will be discussed further in section 5.6.4

5.6.2 Relay Attack

The enhanced Captcha is designed in order to bypass the relay attack.

- The delay time is carefully measured and decided to ensure that a bot does not have enough time to get an answer from a solver. In addition, the delay time chosen is smaller than the lower bound of the simulation results.
- The images are randomly arranged so it is harder to describe the positions of the images. To be able to correctly tell the locations of the objects the bot either needs to label them such as 1 to 6 or re-arrange the objects (eg on one line), which requires time and effort to modify the captured image of the Captcha before sending to the solver.

Comparing with the result in the simulation, the delay time in real relay attack may take longer as there is geographical distance between Captcha solver and the bot server. We also set up an experiment to test the enhanced Captcha ability to be immune against a real relay attack from Koobface. The results (see Section 6.2.4) show that it takes at least 14 seconds for the bot to receive the answer from a human solver. Therefore, it is quite safe to conclude that our enhanced Captcha is secure against the relay attack.

In addition, it is easy to improve the resistance by reducing the delay time between animations. The delay time parameter is configurable for this purpose. However, again, there is a trade-off between security and usability. A good balance between security and usability needs to be considered.

5.6.3 Image Recognition Attack

Image recognition is a hard problem. Unlike text with fixed shapes of letters and numbers, there is no boundary on the numbers of shapes of the objects on the images. Therefore, it is very hard for an automatic program to learn and distinguish all the possible figures from a large and diverse image database. Although there have been a number of image recognition proof of concepts published [134, 135, 143, 146–150], there is no practical evidence that these exploits are actually efficiently working in the real world. The image recognition algorithms are usually tested against some image-based Captcha schemes that have flaws in their design. For example, Asirra [135] uses only two classes of animals dog and cat. Another example is Bongos [146] that asks users to classify a visual block into the right group (eg lines and circles). Not only a random guess results in a success rate of 50%, the figures used are also easy to recognize using segmentation.

There are images that are hard to auto-recognise. According to [117, 125–127, 146, 151], automatic image recognition program can only achieve a small rate of success (less than 5%) on those image Captcha schemes that have the following characteristics:

- significant large image database
- large number of image classes (categories)
- dynamic database. That means the database is not fixed but updated with new images.
- randomly distorted images. It means each time an image is displayed, it is processed with a random distort. This can prevent an auto-program to store the image to its database and recognize it in the next times. This also makes it hard for segmentation.
- good challenge question. The challenge should not be too easy to guess (as in the case of Bongos).

The above characteristics are our design principles. Though the emphasis on our scheme is to prevent the relay attack, we also carefully select and process the images so that it is hard to auto-recognize. Therefore, we can conclude that our Captcha is resistant against image recognition attack.

5.6.4 Usability

We claim that our enhanced Captcha is user friendly because the following requirements are achieved:

- It is easy to identify the correct answer: Since the images are only randomly distorted such as adding noise to the background without distorting the shape of the animals, it is easy to recognize the animal on the images.
- It takes a reasonable amount of time to solve the challenge: selecting 3 images from 6 images does not require much time for a human to perform. It takes average 3 seconds to drag an image to the corresponding box so it takes around 10 second for solving a Captcha.

5.7 Conclusion

Captchas are designed to discriminate between computer scripts from spammers and real human beings. By exploiting the human-machine gap on recognizing visual objects (such as text and images), Captcha is still the most popular Turing test to differentiate human and bots. Captcha is used on virtually every major website on the Internet. Captcha can be used in online polls, stopping spam and dictionary attack (eg guessing password). Certainly, there will be advances in automatic Captcha recognition. But the trick is to maintain a gap between the Captcha design and Captcha solver. The popularity of Captcha in practice shows that it is still good to use Captcha against bots.

Recently, the relay attack poses a significant threat to Captcha security. As human solve the Captcha for bots in the relay attack, it gains high success rate. In this chapter, we have presented a design of an enhanced Captcha to bypass the relay attack. Our design has the following advantages:

- It is hard for auto-recognition. We use an image-based scheme, which is much harder to auto-recognize comparing to text-based. The image database is also large, dynamic and has a large number of categories. The images are also randomly distorted before embedded to the Captcha challenge.

- It requires human intervention. We use drag and drop scheme that requires a user to use a mouse to drag and drop the answers to appropriate places.
- It is resistant against the dictionary attack. A random guess gains less than 1% success rate.
- It is resistant against the relay attack.
- It is user-friendly as a user can easily recognize the images.
- It can prevent bots from redirecting the Captcha challenge to another website for solving.

However, our enhanced Captcha still have some disadvantages.

- More efforts for creating and maintaining image database. To create a good and large image database as described in section 5.5.1, it may require significant efforts for collecting and updating qualified images.
- Short delay time. Though an average person can usually solve our Captcha within 10 seconds (ie the delay time between animations), some may spend more time, even minutes, to solve the Captcha. In that case, the user may get annoyed as the images in the Captcha challenge keep moving after every 10 seconds. The delay time in our design is adjustable; however, this comes with the cost of security against the relay attack.
- Not compatible to some browsers. As there are animation and drag and drop in our Captcha, it is likely that RIA (rich internet application) technology is chosen to implement the Captcha. Though RIA is applied widely, there may be cases that users' browsers do not display the Captcha correctly. A solution in this case is to ask the users to turn on appropriate options on their browser.

We have another concern is that our scheme security against image recognition is uncertain. As we already use many tricks to improve the resistance against this type of attack, it is acceptable that our Captcha is secure. In practice, this security measurement is usually unknown until there is a proposed algorithm that can conceptually break the Captcha (by image recognition). There are several ways to improve the Captcha security.

- Use 3D objects. We can use 3D objects instead of images in our enhanced Captcha. According to Cui et. al [126], it is harder for an automatic program to recognize 3D objects than images. However, this may affect performance of the system. For example, it takes more time to generate and load the Captcha.

- Use code obfuscation. Bots can try to reverse-engineer the Captcha challenge to look for information that it can use to recognize the images. Therefore, we can use code obfuscation to avoid fast and automated reverse-engineering.

- Use RIA (Rich Internet Application) technology. RIA not only improves the user interaction of Web application but also offers better security against reverse engineering and automate recognition attacks on the client rich application [125]. Main RIA technologies can be used to implement Captcha are Ajax based on JavaScript, Adobe's Flex based on Flash, and Microsoft's Silverlight. More details about these technologies can be found in [152]. Each technology has its own advantages and drawbacks. Therefore, when choosing one, a good balance among security, usability (such as popularly supported by many browsers) and performance should be carefully considered.

6

Implementation and Experiments

In this chapter, we are going to describe the implementation and testing of API Verifier and our new enhanced Captcha. Though these are only prototypes, the API Verifier and the enhanced Captcha are still implemented with all functionalities as described in Chapter 4 and 5. We use JavaScript, Ajax and J2EE to implement these systems. In testing, to conduct attacks against the API Verifier and the new Captcha, we use a lab-test bot which is modified from Koobface, a current most popular existing Web 2.0 bot. All relevant types of attacks are covered in our experiments to ensure that the API Verifier and the enhanced Captcha function as expected. From the results of the experiments, we are able to conclude that our proposed systems can provide a plausible solution to protect Web services from being exploited by Web 2.0 bots.

6.1 Implementation

6.1.1 Test of Security Goals of the API Verifier and the enhanced Captcha

We have implemented our system to check its efficiency and friendliness on the one side and its resistance against the well-defined attacks as described in Section 4.4 and Section 5.6 on the other. In particular, the API Verifier is tested against spoofing MAC address and bypassing Captcha attacks while the enhanced Captcha is tested against relay attack. To achieve the goals, we perform a collection of tests to ensure the API Verifier and the enhanced Captcha meets the following security requirements.

1. The API Verifier is able to distinguish between a human user and a computer program

This is also the main feature of the API Verifier. As mentioned in Chapter 4, to be able to filter out requests from bots, the API Verifier must be able to specify which API calls are likely from bots based on the encrypted MAC address and Captcha verification. We examine the following hypotheses.

1. **Hypothesis 1a**: The API Verifier is able to detect if a user uses a new computer to make an API call to the Web service, consequently requests a Captcha verification from the user.

2. **Hypothesis 1b:** The API Verifier is able to verify if an answer for the Captcha challenge from a user is correct or not, hence grant authentication accordingly.

3. **Hypothesis 1c:** If a user already passes a Captcha verification and uses the same computer again, the API Verifier is not going to challenge the user with another Captcha verification.

2. The API Verifier is immune against spoofing MAC address attacks

As stated in Section 4.4, there are several attempts an attacker can make to replace the genuine MAC address with a fraud one. We claim that our API Verifier is designed in such a way that it is possible to prevent such attacks. In the experiments, we test the following hypotheses.

1. **Hypothesis 2a:** If a MAC address is spoofed by replacing the encrypted message while it is sent to the server for verification, the API Verifier is able to detect and deny the communication.

2. **Hypothesis 2b:** If the current MAC address of a computer is modified, the API Verifier is still able to get the permanent MAC address.
   Recall, the current MAC address is the one registered with the OS. By default, the current MAC address is the same as the permanent MAC address.

3. The API Verifier is able to prevent automated program from bypassing its Captcha verification phase

Apart from spoofing the MAC address, an attacker can also try to solve the Captcha challenge of the API Verifier. Therefore, we need to conduct some experiments to ensure that the API Verifier is able to prevent this type of attack. We deploy two forms of Captcha, one is the popular text-based Captcha and one is our enhanced Captcha as described in Chapter 5. We study the following hypotheses.

1. **Hypothesis 3a:** If there is a redirection of a Captcha challenge, the API Verifier is able to detect and generate a new Captcha.

2. **Hypothesis 3b:** The Captcha used by the API Verifier is not broken under relay attacks.

6.1.2 Koobface

As far as we know, there is no existing botnet that behaves like Botnet 2.0 described in Chapter 3. In other words, there is still no botnet that truly uses Web service as a means of C&C as described in the proposed C&C model of Botnet 2.0. Thus for our tests, we choose Koobface. This is a popular Web 2.0 botnet that we modify so it serves our purposes.

Recall, Koobface is a Web 2.0 botnet that targets users of social networking websites such as Facebook, Myspace, Twitter and so on [77]. By luring social network users to visit their website under the disguise of a genuine website such as Youtube, Koobface has been using social networking websites as the means of spreading itself. Once a user has visited a malicious website, they usually get a message asking to download an update (for instance a new version of Flash) so they are able to view the contents. When users decide to download the update, they unwillingly download and install the bot. Once installed in the victim's computer, the bot starts reporting back to a Koobface C&C server and waits for commands. After stealing a victim's private information (such as username and password), the bot continues to search for new opportunities to propagate itself. For this purpose, it uses the API provided by the social network service and logs onto the victim account. Next, it posts Koobface spam site links to the victim account. Figure 6.1 illustrates how Koobface undergoes social network propagation.

The Koobface bot is equipped with a Captcha breaker component to bypass Captcha verification if there is any. Once a Koobface bot has encountered a Captcha, it sends a copy of a Captcha challenge to the C&C server. Each Koobface bot is programmed to keep querying the C&C server for new Captcha challenges. Once a new challenge is fetched by one of the bots, the user of the infected computer is prompted to solve the challenge. Figure 6.2 [38] illustrates the communication exchanged when the Captcha breaker tries to bypass the verification.

After downloading the image, the Captcha breaker component relegates other open program windows to the background and prompts the victim to identify the characters

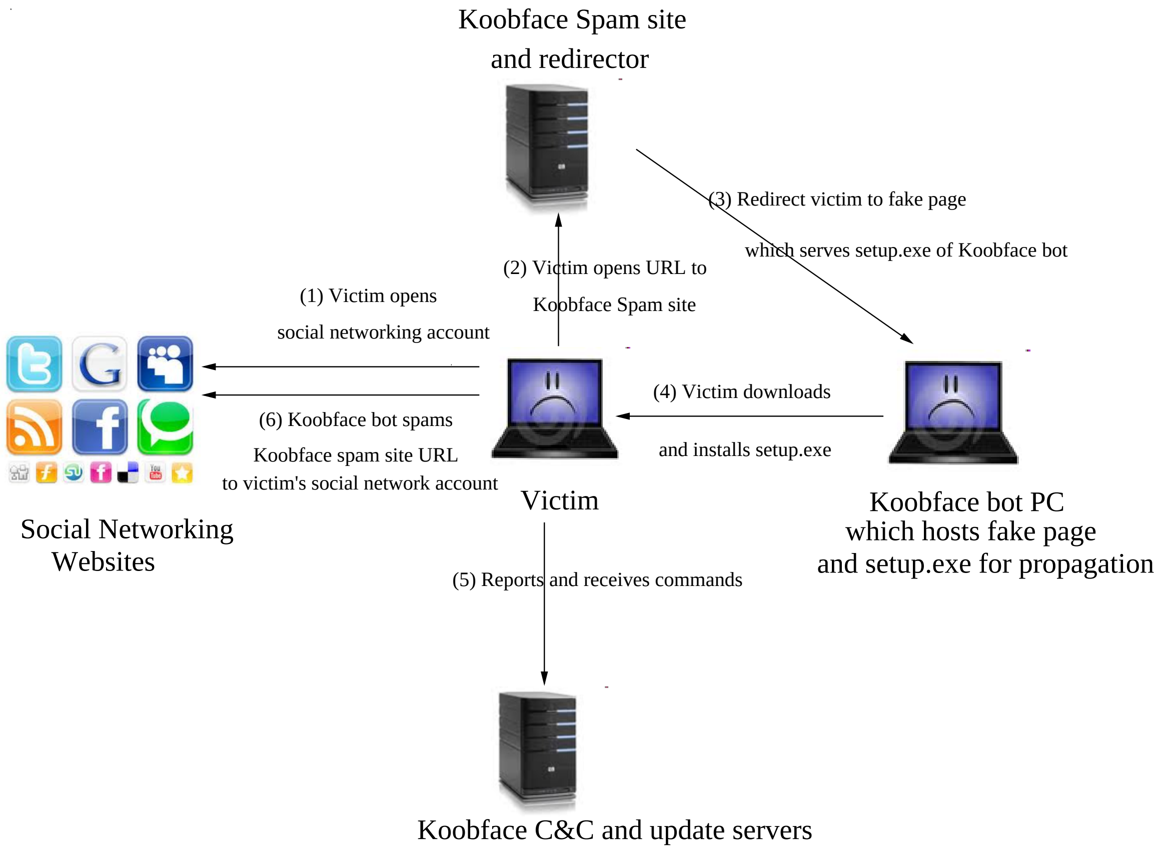


Figure 6.1: Koobface Social Networking Propagation

on the Captcha image. One typical Captcha image generated by this component to allure the user to solve the challenge for them is shown in Figure 6.3 [38].

Koobface is a sophisticated bot that consists of many components but apart from the above functionalities, others are out of context for testing the API Verifier. Therefore, we implemented a simple version of the Koobface bot that featured only the propagation and Captcha breaker component. There are several reasons to consider the Koobface botnet for our experiements:

- It is targeting social network websites who are also targeted by Botnet 2.0.
- It has similar properties to the ones exhibited by Botnet 2.0 (for instance, it automatically posts information to victim's blog by using API)

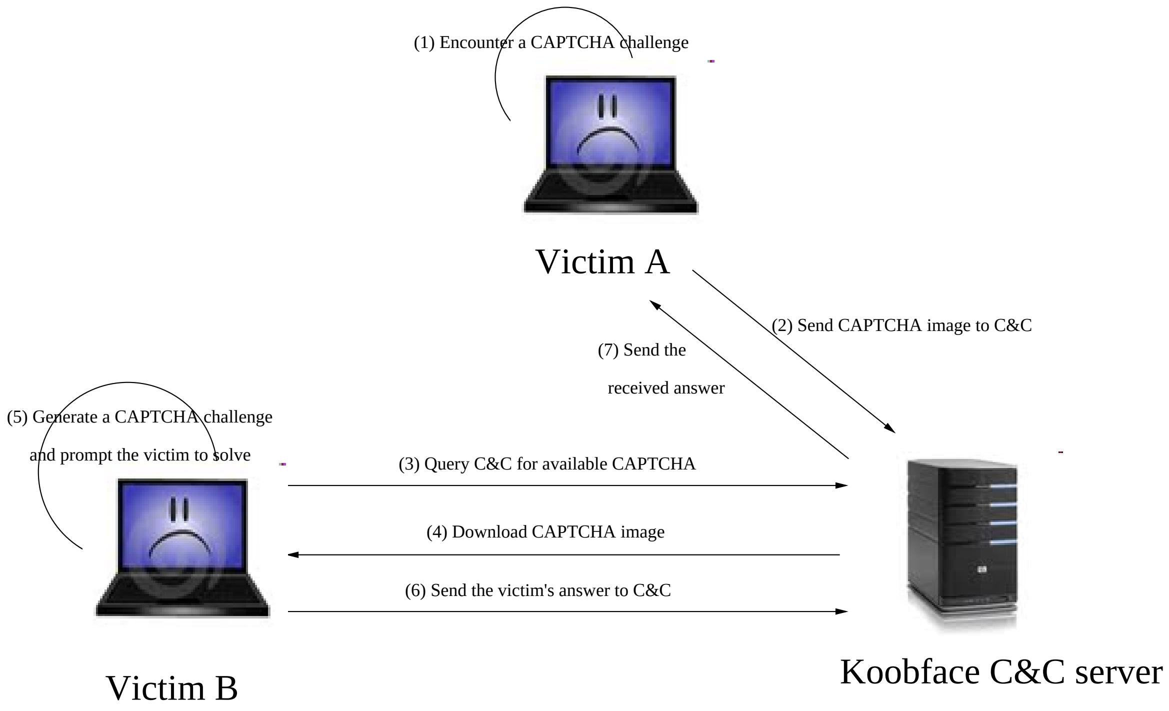
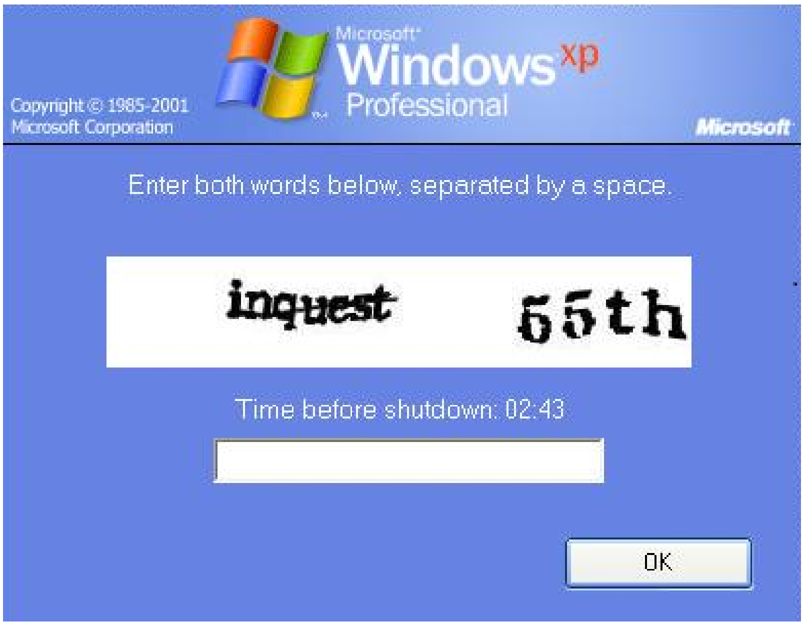

(1) Encounter a CAPTCHA challenge
Victim A
(2) Send CAPTCHA image to C&C
(7) Send the
received answer
(5) Generate a CAPTCHA challenge
and prompt the victim to solve
(3) Query C&C for available CAPTCHA
(4) Download CAPTCHA image
(6) Send the victim's answer to C&C
Victim B
Koobface C&C server

Figure 6.2: Koobface Captcha Breaker Component



Enter both words below, separated by a space.

Time before shutdown: 02:43

Figure 6.3: Sample of Koobface fake Captcha

- It contains a Captcha breaking component that is useful for testing our API Verifier.

6.1.3 Implementation of the API Verifier and the new Captcha

Since it is for testing purpose, we only implement a demo version of the API Verifier. We do not implement a full set of API methods (e.g get, update entries and so on) but a simple post method that allows to fetch user data to the server and store it. The reasons for not implementing the full functionality of the API Verifier are as follows:

- The main task of the API Verifier is to distinguish a bot from human activities rather than testing the API calls.

- Methods and functionalities vary from one Web service to another and to test the security of our API Verifier we need to implement only those components that relate to the identification of users.

The implemented components of the API Verifier are simplified. For instance, we use properties files for storing users' profiles instead of a database or set up a normal server instead of using a Web 2.0 service. The components work in a similar way but we save time and effort.

The API Verifier has two parts, namely Client (APIV client) and Server (APIV server). For the implementation of the API Verifier, we use the Java language and the J2EE programming tools. The implementation is working under the Window operating system. The encryption algorithm we use is the Advanced Encryption Standard (AES) with 128-bit keys. One aspect of our design that requires special attention is the access to the permanent MAC address. We need to use a lower-level code to interact with the ethernet/wireless device. The library called Corutine [153] helps us to get the permanent MAC address. We implement two versions of Captcha. The first version is a modern text-based Captcha with angle lines on the string to make its segmentation difficult. The second version is our enhanced Captcha (see Chapter 5). More details about the functions implemented as well as a sequence diagram of the API Verifier can be found in Table 8.1 and Figure 8.1 in the Appendix.

The enhanced Captcha is implemented using JavaScript and AJAX. The implementation is scaled down and served as a proof of concept. Consequently, instead of a large number of images, the Captcha database contains a small number of images. The images are categorised into 10 classes (animals). A Captcha challenge contains 6 images which are randomly selected by the generator. In our experiment, the images are arranged circularly for the ease of describing their positions. Each image is assigned with a random label. After every 10 seconds, the images on the Captcha are randomly changed their labels and positions. Pseudo code of the Captcha system can be found in Section 5.5.4 and the function list as well as the sequence diagram can be found in Table 8.2 and Figure 8.2 in the Appendix.

6.1.4 Implementation of simple Koobface botnet

A simplified version of Koobface botnet is designed with a simple C&C server and bots. The version is implemented using Java and J2EE scripting. Unlike the real one, the C&C server is only for forwarding the Captcha images and Captcha answers to the bots as only these functionalities are relevant to our testing purpose. The simplified Koobface bot contains the Captcha breaker as the only component taken from the original bot. Other necessary functionalities (such as the ability to interfere messages exchanged between APIV client and server and ability to make post request to APIV server) are added to the design so the attacks against the API Verifier could be conducted. The implemented bot does not have the ability to steal victim's information. All the credentials such as usernames and passwords are assumed known to the bot once it invades the victim PC. The function list and the sequence diagram of the simplified Koobface can be found in Table 8.3 and Figure 8.3 in the Appendix.

6.2 Experiments and Results

6.2.1 Experiment Set Up

In the experiment, the networks of botnet, Web service and users' computers are set up with precautions so that bots are not accidentally spread to the outside world via the Internet. The networks are set up as follows:

- The networks are local area networks (LAN) and isolated from the Internet.

- Two servers are set up, namely, the Web server that provides public Internet service and the C&C server that controls the bots. Note that the API Verifier is installed on the Web server.

- The simplified Koobface does not interact with any other servers except the set-up servers. The IP addresses of the set-up servers are hard-coded and there are no other IP addresses provided so the bot could not access any other sites.

- APIV client and the simplified (lab-test) Koobface bots are installed on three computers A, B and C. The bots on those computers are also named bot A, bot B and bot C.

- The database storing animal images for the enhanced Captcha is located on the API Verifier server. When generating a Captcha challenge, the APIV server randomly picks the animal images or randomly generates a string depending on the Captcha form used. For tests, where we are checking our design against the Captcha bypassing attacks, we use both forms of Captcha. For other tests, we apply the text-based Captcha only.

6.2.2 Security Requirement 1: the API Verifier is able to distinguish between human and a computer program using Captcha verification

1. Assumption:

There are some assumptions when conducting the experiments for this security requirement.

1. Bot is not able to solve a Captcha challenge. Testing with a bot trying to break Captcha is conducted in different scenarios in Section 6.2.4.
2. User Alice did not use computer A or B before to make an API call to the Web service.

2. Experiment 1a:

Description: This experiment is to test if the hypothesis 1a is true.

Hypothesis: the API Verifier is able to detect if a user uses a new computer to make an API call to the Web service, consequently requests a Captcha verification from the user.

In the experiment, Alice who uses computer A for the first time will make an API call to the Web service. We expect that the system will ask Alice to solve a Captcha because it detects an API call from a new computer. We also expect that the response and Captcha display are reasonable fast (within 2 seconds).

Experiment procedure:

1. User Alice on computer A tries to make an API call `post(url of server, username, password, text message)` to the Web service.
2. The APIV server returns a message window asking the user to solve a Captcha challenge as showed in Figure 6.4.

3. Alice correctly solves the Captcha and is able to post the message.

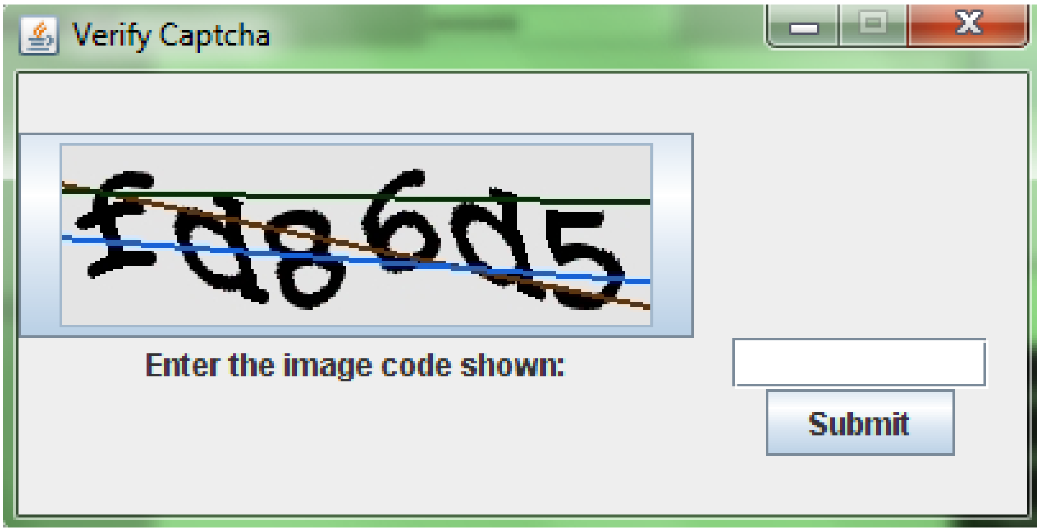


Figure 6.4: Example of Captcha verification from the API Verifier

Results: A Captcha verification is returned as expected. The response time is 1.07 seconds, which is an acceptable performance.

3. Experiment 1b:

Description: To verify the hypothesis 1b, we will test if the system is able to handle the Captcha verification properly.

Hypothesis: the API Verifier is able to verify if an answer for the Captcha challenge from a user is correct or not, hence grant authentication accordingly.

Alice will intentionally type in a wrong answer for the Captcha when she receives a Captcha verification from the API Verifier. We expect that the system will generate and show a new Captcha challenge because it should be able to detect that the answer is not correct.

Experiment procedure:

1. This time Alice uses a different computer, say B, to post a message to the website using API post method as in the previous experiment.

2. The APIV server returns a message window asking the user to solve a Captcha challenge as it detects a new computer used.

3. Alice intentionally types in a wrong answer.

4. The server returns an error message and a new Captcha challenge for another verification.

5. Alice types in the correct answer and the posted message is through and displayed on the website.

Results: The system asks Alice to solve a Captcha challenge as expected because Alice uses computer B for the first time. When Alice provides a wrong answer, the system is able to detect it and generate a new Captcha challenge per required. When Alice types in a correct answer, the API call is accepted as the designed purpose. The response times from server are 0.94 seconds and 1.21 seconds, which are reasonable.

4. Experiment 1c:

Description: This experiment is to confirm if the hypothesis 1c is true.

Hypothesis: If the user already passes a Captcha verification and uses the same computer again, the API Verifier is not going to challenge the user with another Captcha verification.

Experiment procedure:

1. Alice uses computer B again to post another message to the website using API `post` method.

2. APIV server this time does not ask Alice for verification and the message from Alice is displayed.

Results: The system operates as the design purpose. Because computer B is already used to successfully make API calls to the Web service in experiment 1b, it is recognized as a genuine user. Therefore, Alice does not encounter another Captcha verification.

5. Discussion:

The results show that all three hypotheses are confirmed. Therefore, we can conclude that the API Verifier satisfies the first security requirement. In particular, the API Verifier is able to distinguish an API call is from a genuine user or a bot. If it is from a bot, the API call is denied. Otherwise, the API call is accepted and the user will not encounter another Captcha verification when making following API calls.

The experiments also show that this security requirement is achievable in practice. The response time of the system is also within the accepted range. The performance can also be improved further by improving the code. For example, using Ajax to update certain components of the GUI rather than loading the whole page can reduce the responding time.

6.2.3 Security Requirement 2: the API Verifier is able to prevent spoofing MAC address attacks

1. Assumptions:

We make the following assumptions when conducting the experiments.

1. The kernel of operating system (OS) is not accessible to bots. This assumption is satisfied in the two following cases. In the first case, when OS is properly protected against any unauthorised access (including via bots). In the second case, when bots do not try to access OS kernel as any attempt may reveal their presence.

2. Alice never auto-updates her blog before, in other words, Alice does not need to use any API and just manually updates her blog via a browser. Alice's PC (computer A) is infected by a lab-test Koobface bot A.

3. The bot installs APIV client on computer A to be able to post content to the Web service using API.

2. Experiment 2a:

Description: This experiment is to test if the hypothesis 2a is true.

Hypothesis: If a MAC address is spoofed by replacing the encrypted message while it is sent to the server for verification, the API Verifier is able to detect and deny the communication.

In this experiment, the bot wants to use Alice's account to store URLs of malicious websites. Since Alice has never used the web API before, the bot will be given a Captcha challenge. To bypass the Captcha verification, the bot may try to spoof the permanent MAC address of computer A on the fly. In particular, the bot will try to replace the encrypted MAC address message with a fraud one. The fraud message is extracted from a genuine communication between a human user (i.e, an attacker) and the Web server. To do this, the attacker could attempt the following steps:

1. makes an API call using Alice's account on computer C. The API Verifier sends a Captcha challenge.

2. solves the Captcha challenge and therefore, he is identified as a human.

3. observes the Internet traffic and intercepts the encrypted message (say E(M)) that contains the permanent address of computer C.

Then the attacker instructs the bot to replace the genuine encrypted MAC address with the above E(M) with hope that it will not encounter a Captcha verification.

We expect the system is able to detect that E(M) is not a correct encrypted MAC address in the attack. Therefore, the system will present a Captcha challenge which the bot cannot solve.

Experiment procedure: The attack is performed according to the following steps:

1. Bot A uses Alice's credentials and calls a `POST` command (API) from computer A.
2. The API client sends the user information (including username, password and the permanent MAC address of computer A) to its API verifier server. The information is encrypted using the private key and session key (as described in section 4.2).
3. The bot observes the traffic between computer A and the network and replaces the encrypted message containing the permanent MAC address of A by the encrypted message containing the permanent address of C, (E(M)).
4. The API verifier server checks the information received (with the replaced cryptograms).

Results: An error message is returned and a Captcha challenge is displayed. Because the bot is not able to solve the Captcha, it fails to make any API call to the Web service. The results confirm the experiment goal.

3. Experiment 2b:

Description: The experiment is to confirm the hypothesis 2b.

Hypothesis: If the current MAC address of a computer is modified, the API Verifier is still able to get the permanent MAC address.

In this scenario, bot A will attempt to alter the value of the current MAC address of the computer. The system still should be able to get the correct value of the permanent

MAC address and deny the communication from the bot.

Experiment procedure:

1. Bot A replaces the MAC address of computer A stored in the computer register (known as current MAC address) by the MAC address of computer C.

2. The bot makes a post request using API with the hope that with the masked current MAC address, it will be able to bypass the Captcha verification phase.

3. However, the API Verifier Server still returns a Captcha for verification.

Results: A Captcha verification is thrown even the bot changed the current MAC address to the attacker's desired one. When examining the log file of the system, we can see that the value of the MAC address sent to the APIV Server is the same with the permanent MAC address of computer A. This shows that the system functions as expected.

4. Discussion:

The experiments show that the API Verifier is able to get the permanent MAC address even the current MAC address is changed and only uses the permanent MAC address for identifying the computer. In addition, the API Verifier is able to detect spoofed encrypted message of the MAC address. By investigating the log file, we can see a different value of the MAC address (after decrypted at the server side), not the attacker's desired one (computer C). In other words, the APIV Server is able to detect that E(M) this time (in the attack) does not give a correct MAC address of computer C. This is the achievement of using session keys. Note that E(M) was encrypted with a different session key. So when the APIV server decrypts E(M) with the current session key, it gives a different value rather than the permanent MAC address of computer C. As hypotheses 2a and 2b are confirmed, we can conclude that the security requirement 2 is realisable in practice.

6.2.4 Security Requirement 3: the API Verifier is able to prevent automated program from bypassing its Captcha verification phase

1. Assumptions:

In these experiments, we are going to test the new enhanced Captcha security. We use a sample of Koobface botnet to attack this new scheme. Therefore, we need to modify the API Verifier as well as the existing Koobface bot to suit our purposes as described in Section 6.1.4. These changes are included as the assumptions as follows:

1. The capability of our Bot 2.0 has been upgraded. The bot is able to redirect the link of a Captcha challenge to other computers asking people to solve challenges on its behalf.
2. Alice's and Bob's computers (computers A and B, respectively) are infected by our lab-test bots and their copies residing on computers A and B are called bots A and B, respectively.
3. Alice has never used the web API service before.
4. Carol owns computer C. Carol can be the attacker or a Captcha solver who works for the attacker.
5. The current automated image recognition algorithms cannot distinguish two images that represent two different classes (two images of two different animals, for instance).
6. The lab-test bot is able to mimic the "drag-and-drop" action performed in reality by a human. Note that assumption is difficult to be satisfied.

2. Experiment 3a:

Description: We are going to test the hypothesis 3a in this experiment.

Hypothesis: If there is a redirection of a Captcha challenge, the API Verifier is able to detect and generate a new Captcha.

Bot A will try to impersonate Alice and use a human working on a foreign machine (computer C) to solve a Captcha challenge. Because the bot cannot solve the Captcha challenge, it will attempt to re-direct the Captcha challenge to a human user to solve for them. We assume that there is a human user using computer C that is willing to solve the Captcha for the bot. We expect that API Verifier should be able to detect the re-direction of the Captcha challenge, discard the current Captcha and generate a new one.

Experiment procedure: The attack goes as follows:

1. Bot A intends to impersonate Alice. It uses Alice's credentials and makes an API call to the Web service.

2. In response, the APIV server returns a Captcha challenge to bot A.

3. Bot A receives the challenge and redirects the Captcha link to the C&C server.

4. The C&C server forwards the link to the human working on the computer C (Carol). Note that each Captcha link is assigned with a session ID (say X in this case).

5. Carol opens the link, however, she only receives an error message as the APIV server discards the session X.

6. At the mean time, the APIV server generates a new Captcha challenge (Y) and displays to bot A.

Results: Carol only receives an error message and bot A encounters a new Captcha challenge (Y) as expected. When Carol opens the link, there is a request to the APIV server to re-display the Captcha challenge having session ID X. This violates the rule that each session ID is used once. Therefore the APIV server discards the session X, returns an error message to the client who opens the re-direct link. As Captcha X is expired on computer A as well, the APIV server generates a new Captcha challenge Y and returns it to bot A. As a result, bot A is not able to solve the Captcha Y and is not able to make any further API call. In other words, the hypothesis 3a is confirmed.

3. Experiment 3b:

Description: This experiment is to test the hypothesis 3b.

Hypothesis: Captcha used by the API Verifier is not broken under relay attacks.

In this scenario, bot A will try to impersonate Alice and use another bot B that lures a user B to solve the Captcha challenge. When bot A encounters a Captcha challenge, instead of redirecting the Captcha link to another foreign computer as in the previous experiment, bot A will send the Captcha challenge information to bot B via the C&C channel. This information can be an image of the Captcha challenge or elements of the Captcha challenge so that bot B can re-construct the challenge on computer B that it resides. Bot B will lure the user of computer B to solve the challenge for it. When the answer is received, bot A will submit the answer to the APIV server to pass the Captcha verification. We expect that our new enhanced Captcha will be able to prevent the bots from indentifying the locations of the answers. That means the answers sent from bot B are useless. As a result, bot A is not able to make any further request to the Web service. In addition, the responsding time for displaying the enhanced Captcha should not exceed 2 seconds.

We will use two Captcha schemes in this experiment. One is a typical text-based Captcha and the other is our enhanced Captcha. Bot A and B are also modified so that it can perform relay attacks against our new Captcha.

Experiment procedure: The attack proceeds according to the following steps.

1. Bot A steals Alice's credentials and uses them to make an API call to the web

service.

2. In response, the APIV server returns a Captcha challenge to bot A.
3. In turn, bot A captures the Captcha challenge and sends the image of the Captcha challenge to its C&C server.
4. Bot B residing on computer B queries the C&C server and downloads the image of the Captcha challenge.
5. Bot B generates an appropriate challenge from the information received from the C&C server such as the Captcha image and a limit time so that it should get the answer before the session on computer A expires.
6. Bot B lures user B to solve the Captcha challenge. Consider the following two cases:

   (a) **Case 1: Captchas are text-based**. A Captcha challenge used by the APIV server looks like the one presented in Figure 6.5. Then:

      i. If Bob decides to solve the Captcha challenge, he enters the string.

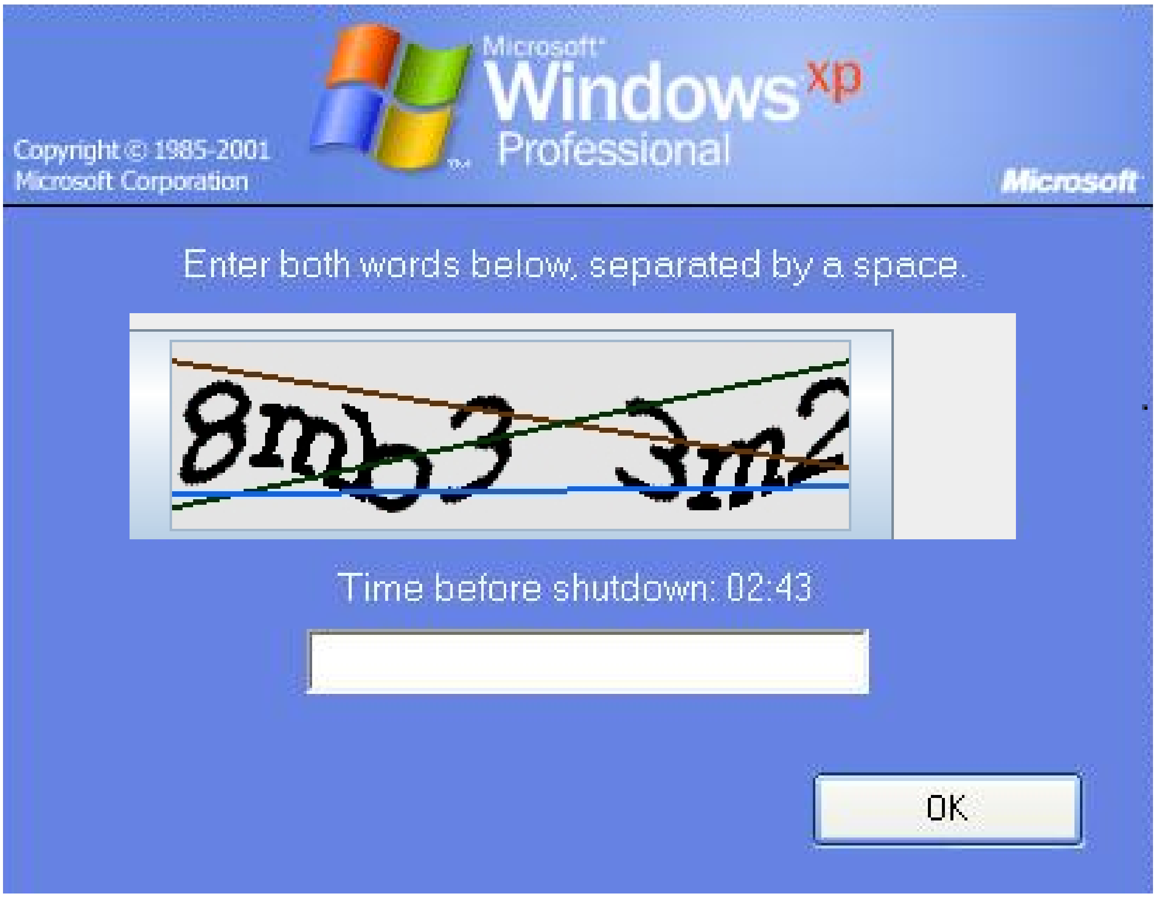
      ii. Bot B takes the response and sends it to the C&C server.

      iii. Bot A prompts the C&C server and collects the solution to the challenge.

      iv. Bot A submits the answer and passes the Captcha verification.

   (b) **Case 2: Enhanced Captchas are used**. Note that the original Captcha breaking component of Koobface fails to collect all necessary information about the challenge. Thus, the challenge cannot be reconstructed by bot B and consequently, Bob is not able to solve it (even if bot B has been successful in convincing Bob to do this).

   To continue with our experiments, we have modified the setup so the bots are able to store all the information about the Captcha challenge. We have tested two different ways of the Captcha challenge reconstruction.



Copyright © 1985-2001 Microsoft Corporation

Enter both words below, separated by a space.

Time before shutdown: 02:43

Figure 6.5: Example of Koobface fake Captcha with text Captcha

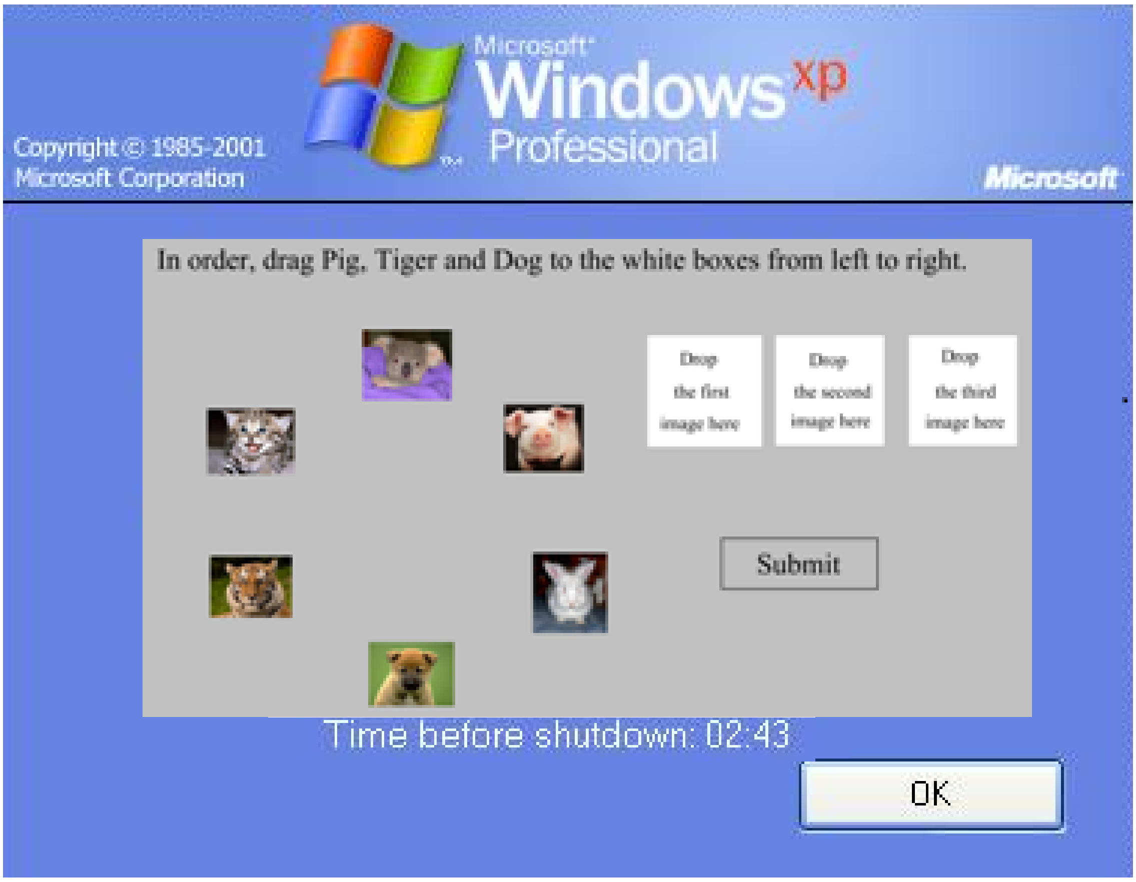
- The bots are implemented in such a way so that they are able to capture all the information about challenges so they can reconstruct them. In this case, the attack continues:

  i. Bot A collects information about the Captcha challenge and sends it to the C&C server.

  ii. Bot B downloads the Captcha challenge information, reconstructs the challenge and displays it to Bob (see Figure 6.6).

  iii. Bob solves the challenge by dropping the images to the answer boxes.

  iv. Bot B identifies which images are dropped and sends the dropped


Microsoft Windows xp Professional
Copyright © 1985-2001
Microsoft Corporation
Microsoft
Drop the first image here
Drop the second image here
Drop the third image here
Submit
OK

In order, drag Pig, Tiger and Dog to the white boxes from left to right.

Time before shutdown: 02:43

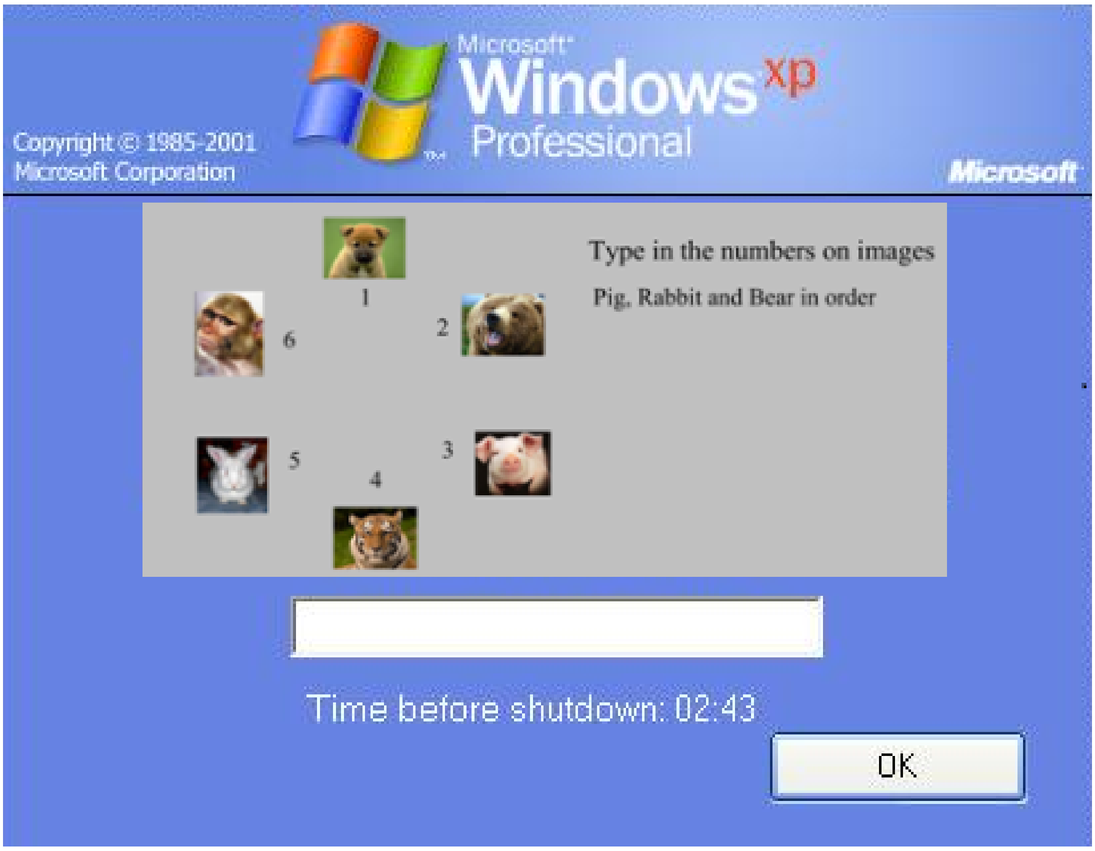
FIGURE 6.6: Example of Koobface fake Captcha with our enhanced Captcha

image identifiers (labels) to the C&C server.

v. Bot A fetches the labels from the server and tries to drop the appropriate images into the answer boxes.

vi. However, because it takes more than 10 seconds for bot A to get the answer, the labels as well as positions of the animal images have been changed. Therefore, the labels got from the C&C server become obsolete. Bot A fails to indentify which images are the answers and cannot submit an answer. Note that the average time needed for bot A to get a solution from bot B is 17 seconds (recorded after running the test 10 times).

- Instead of reconstructing challanges, bots take screenshots of Captcha challenges and creat a customised Captcha from screenshots. So additional features, such as taking screenshot of Captcha, cropping images and creating new Captcha with suitable questions, are implemented to the C&C server. In this case, the attack continues as follows.



Type in the numbers on images

Pig, Rabbit and Bear in order

Figure 6.7: Another Koobface fake Captcha with our enhanced Captcha

i. Bot A takes a screenshot of the Captcha challenge then sends the image to the C&C server

ii. The C&C server crops the image to get only the animal ring and the string containing names of animals required to drop to the answer box (eg Pig, Rabbit and Bear). We do not use OCR (optical character recognition) to extract the string as OCR do not always give the correct string.

iii. The C&C server labels the animals on the image from 1 to 6 and store the new image.

iv. Bot B downloads the new image and constructs a new Captcha challenge as shown in Figure 6.7. Notice that there is a new question that asks "Type in the numbers on images of Pig, Rabbit and Bear in order".

v. Bot B tricks Bob to solve the challenge. Bob enters the correct answer into the answer box. In this case, the answer is (3,5,2).

vi. Bot B gets the correct answer and sends it back to the C&C server.

vii. Bot A prompts the C&C server and if the solution has arrived, it downloads it and drags and drops appropriate images into the answer boxes.

viii. The APIV server returns an error message and a new Captcha challenge for another verification. The answer submitted by bot A is wrong because the images on the Captcha already changes their positions before bot A gets the answer from the C&C server. Note that the average time needed for the bot A to get solution from the bot B is 15 seconds (recorded after running the test 10 times).

Results: In the case that a text-based Captcha is used, bot A is able to solve the Captcha challenge by submitting the answer sent from bot B. As the answer is a string of text, it is transferable between bots. As a result, this type of Captcha scheme is subjected to relay attacks. On the other hand, relay attacks fail in the case of our enhanced Captcha is used. Either bot A transfers the captured image or components of the Captcha challenge to bot B, the communication times are longer than the delay time between the two animations of the Captcha. Therefore, when the answer reaches bot A, it is already obsolete and cannot be used to solve the challenge. In other words, bot A cannot submit an answer to the challenge or the answer is wrong. In both cases, the request from bot A to the Web service is denied as the design purpose of our system. This also confirms the hypothesis 3b. In terms of performance, the log file indicates that the average time of creation and displaying the enhanced Captcha is 1.87 seconds. This is an acceptable responding time.

4. Discussion:

The above experiments show that the hypotheses 3a and 3b are true. In other words, our system satisfies the security requirement 3 - bots cannot bypass our Captcha verification. In the experiments, we choose Koobface as it has ability to conduct a relay attack on the Captcha challenge. Comparing with the majority of existing Captcha schemes, which are subjected to the relay attack, our enhanced Captcha is more secure. The minimum delay time recorded in case 1 is 16.2 second while it is 14.3 seconds in case 2. The gap between the minimum delay times in the experiments and the set delay time of the Captcha (10 seconds) is significant enough to conclude that Koobface fails to break our enhanced Captcha. In the experiments, the environment is quite ideal for the bots to communicate with the C&C channel. Bots A and B can keep prompting the C&C server to minimize the delay time. In practice, they have a harder time to communicate to avoid being detected. So the real communication time is expected to be much longer. Therefore, it is safe enough to conclude that our Captcha is secure against relay attacks.

In addition, the bot cannot ask Bob to solve the Captcha within less than 10 seconds. The reasons is, in reality, no application is shut down in such a short notification. This may lead to suspicion to the victim; hence he may find out his computer is infected by a Koobface bot and remove it. If limit time is not provided or is about 1 minute for example, the victim may take time to learn the error message and not in hurry to solve the Captcha. Hence it may take much more than 10 seconds to solve the challenge.

A better way is that the Koobface bot sends the Captcha image to the employed Captcha solvers. But again, 10 seconds is still too short for the bot to get the answer (as explained in Section 5.5.6). In short, it is quite safe to conclude that the API Verifier is able to prevent bots from bypassing Captcha verification.

6.3 Conclusion

6.3.1 Summary and Achievements

We implement a simplified system to test our solution for the API Verifier and the new enhanced Captcha. Though it is a demonstration version, the API Verifier and the enhanced Captcha are implemented with full functionalities as specified in the design. In particular, the APIV client and APIV server can function as expected (such as communication between the 2 components, session handling, getting the correct permanent MAC address and handling Captcha verification). Apart from a small database, our enhanced Captcha is also implemented with full features (such as image-based, animation, drag and drop ability and session handling). This shows that the design of the API Verifier and the enhanced Captcha are feasible and efficient to implement to real systems.

We choose Java as the programming language and use J2EE programming tools to implement the systems as well as the simplified Koobface. The purpose is twofold. First, we are familiar with the language and J2EE technology. Second, it is easier for us to use Ajax for implementing animation and drag and drop features. There is one possible disadvantage is that the source script based on interpreted JavaScript can be deciphered by bots. For future implementation, we need to address this issue by using obfuscation or encryption to improve the resistance against reverse-engineering attack.

An alternative for Ajax can be Flex technology based on Flash. Flex is a framework for creating RIA (Rich Internet Application) based on Flash Player. Because Flex based on compiled language, it may offer a better resistance against reverse-engineering. In this case, however, because the Captcha challenge is a flash object, it may affect performance and usability of the Captcha. Therefore, when implementing the real Captcha system, the technology choice should be carefully considered.

The implementation is used to test the main functionalities of the system and evaluate its security. From the experiment results, we can conclude that the API Verifier is most likely able to prevent bots from spoofing MAC addresses and bypassing Captcha verifications. In other words, the API Verifier is able to stop Web 2.0 bots (such as Koobface) in general and Bot 2.0 in particular from abusing Web services as a means for propagation and/or as communication channels. Our enhanced Captcha is also resistant against the relay attack.

Besides, our systems have the following advantages:

- As the APIV client is implemented as a dll file, it is light and easy to download and install.
- The APIV client is a library. Users only need to import its header and make API call as normal.
- The APIV client and APIV sever are communicating via SSL. Users' information is all encrypted.
- Our enhanced Captcha is user friendly as the images in our enhanced Captcha are clear and easy to recognize by human.
- The enhanced Captcha is portable. It is not only for use in the API Verifier but also can be plugged into other applications such as registration and online vote.

6.3.2 Drawbacks and Possible Improvements

There is one case in which the API Verifier is not able to prevent Koobface to post malicious contents to social networking websites. It happens when:

- victim already (and usually) used his computer to automatically update contents to his profile and
- now his computer is infected with Koobface bot.

That means the victim already passed a Captcha verification when he first time made an API call to the Web service. Therefore, the user would never encounter another

Captcha verification if he used the same computer to auto-update his profile. Hence, when Koobface bot invades this victim's computer, it does not need to solve a Captcha while using API. However, this scenario is not a concern because there is a very small number of genuine users need and are able to use API to auto-update their blogs/profiles. To be able to use API, a user needs to have basic knowledge of coding. Furthermore, learning API commands of each Web service is not trivial. Users with such skills, in addition, are usually cautious with malicious links and virus. That means they are not easily trapped by Koobface bot (or other Web 2.0 bots) from the beginning. Those users also tend to have their computer well guarded, screen their computer regularly and easily realize abnormal behaviours such as malicious links on their blogs. As a result, they can detect the Koobface bot and remove it before it is able to make any further damage. Thus, we believe that this small failure rate of the API Verifier is accepted in practice.

We can also improve security against this attack by adding an optional feature to notify users of API calls. In particular, the API Verifier will send an email of confirmation to the user whenever an API call is made. Therefore, the user is alerted if their computer is infected once the bot starts posting malicious URLs to their account. Using this feature, a user may receive a lot of confirmation email if they frequently auto-update their profile. However, the user can turn this option off anytime.

Note that the above attack is unlikely to take place in Botnet 2.0. The main difference between the Koobface botnet and Botnet 2.0 is that while Koobface only uses Web services as a means of propagation, Botnet 2.0 utilizes Web services as a C&C channel as well as temporary storage of stolen information. Koobface bots still report to and receive commands from a botmaster's C&C server meanwhile Bots 2.0 do it via Web services as well. Therefore, Bots 2.0 do not use victims' accounts but their own for storing and exchanging information and commands.

Another drawback is that, we do not cover API Verifier fraud attacks which involves decryption and reverse-engineering. We believe that the encryption techniques, such as AES and obfuscation, employed to the tool are among the best existing algorithms. In the future, we may study further about obfuscation techniques to improve this security aspect.

Finally, we do not conduct an image recognition attack on our enhanced Captcha. Image recognition is a hard problem and we are not able to set up such an attack. We already introduce extra features (such as distorting the images with small noise, random labels, large number of image classes and significant large database) to improve its resistance against auto-recognition. However, further study about image recognition techniques is essential to ensure security of our enhanced Captcha.

7 Conclusion

7.1 Conclusion

Botnets have long been cybercriminals' technology of choice in their quest for financial profit. They provide a complete way of controlling a PC and utilizing it for criminal activities. Spammers are renting botnets on a daily basis, and identity thieves are leveraging their keylogging capabilities. More sophisticated attackers are using botnets for direct financial fraud and DDoS. Traditionally, one of the main communication channels used to control bots is IRC protocol, which is followed by P2P and HTTP protocols to circumvent firewalling and improve the botnet resilience. Recently, bots communicate mainly over HTTP with arbitrary servers on the Internet. Moving forward, Bots 2.0 have adapted the C&C channels to utilize the openness of Web 2.0 technology. Botmasters have begun to exploit social network websites as their C&C

infrastructures, which turns out to be quite stealthy because it is hard to distinguish C&C activities from the normal social networking traffic. Our motivation is to investigate Bots 2.0 and propose a suitable solution for this kind of threat.

In this thesis, we have proposed a plausible solution to protect Web services from bots abuses. API Verifier consists of 2 components APIV client and APIV server. It is able to detect bots activities by monitoring API calls made to Web services and using Captcha challenge to verify human users. Once the system detects an API call made from a new computer, it will ask the user to solve a Captcha challenge. There are two cases:

1. If the user is able to solve the Captcha, he/she will be verified as a legitimate user and the API call is through. After that, the user will not encounter another Captcha when using the same computer to make API calls to the Web service.

2. On the other hand, if the user fails to solve the Captcha after 3 attempts, the user is considered as a bot and the API call is denied.

As part of the solution, we have also presented an enhanced Captcha design to thwart relay attack. Our design principal is using animation to change positions of the answers before a bot can get the correct answer from a human solver involved in the relay attack.

7.1.1 Evaluation of Research Goals

To assess if our solution meets the determined research goals, we are going to evaluate our work on the following tasks which are specified in Section 1.2.

1. Overview of the current bot and botnet technology together with the description of main security countermeasures.

   We have presented an in-depth literature review on bots and botnets and existing detection techniques in Chapters 2 and 3 . In particular, we have showed the evolution of bots, their characteristics, their life cycle and network architectures. We have identified the propagation methods used by bots and the survival techniques that allow bots to remain stealth. We have also studied different approaches to detect bots and botnets, which can be related to our solution. This work is important to be able to analyse bots and botnets and propose plausible solutions for detection and protection.

As Web 2.0 bots and Bots 2.0 are our targets, we have studied this class of bots in Chapter 3. We have also provided the model of Bot 2.0 attacks to demonstrate their specific interactions with Web services. Based on these interactions, in particular using API to update content to Web services, we have proposed the API Verifier to filter API calls made from bots to the Web services.

2. Propose a system that is going to be able to distinguish a human request from a bot request to Web services.
   We believe that the API Verifier is able to efficiently differentiate human activities from bots ones. The reasons are:

   (a) Our system is able to indentify if an API call is from a new computer thanks to the permanent MAC address which is globally unique for each computer.

   (b) Bots cannot solve our enhanced Captcha. The reasons will be discussed in Task 4.

   (c) Our system can gain a high successful rate of filtering API calls from bots. Due to their nature (as a computer program and low presence to hide away from detection tools), bots need to use API to auto-update contents to a Web service. In contrast, majority of legitimate users manually update their profile via a web browser and may never use an API call once. Therefore, it has a high chance that when a bot make an API call from a new victim computer, it is the first API call made from that particular computer. As a result, a bot will most likely encounter a Captcha for which it cannot solve. In the case of Bot 2.0, the bot will update content to their own profile (eg botprof) created by the botmaster. The chance of getting a Captcha verification in this case is 100% as it is always the first time the user botprof makes an API call from a new computer, which is the newly victim's.

(d) Our system works on any Web 2.0 bots. We do not base on any particular bot or botnet to design our system. As all Web 2.0 bots need to use API to interact with Web services, their API calls will be monitored and filtered by the API Verifier regardless their genre. Therefore, our solution is general.

In addition, our system is customizable. As described in the designs of the API Verifier and the enhanced Captcha, we have made many fields configurable. Following is the un-exhaustive list of configurable paramaters:

- Session time of the API Verifier. The session time varies depend on the provider's policy so it should be flexible. This can be configured on the server side.
- The number of Captcha attempts. Some service providers want to improve usability by increasing number of Captcha attempts; however, this comes at the expense of the security level.
- The number of MAC addresses allowed for each user. Some users may use more than one computers to auto-update their profiles using APIs. Therefore, to avoid annoying users with Captcha verification every time they switch between their computers, the number of MAC addresses a user can hold for his/her profile is adjustable.
- The number of images in Captcha challenge. By increasing the number of images, its security is improved but it is less user-friendly as a user needs to spend more time to solve a Captcha.
- The delay time of animation Captcha. The default delay time is set to 10 seconds, which is the most suitable value per our simulation results. However, this value is adjustable to suit client's needs. Again, it is important to balance usability and security when setting the delay time.

3. Define features of our system that cannot be changed or spoofed by bots. In our system, we have used two of such features: permanent MAC address and enhanced Captcha.

   (a) permanent MAC address: the permanent MAC address used in our system cannot be spoofed by bots as of the following reasons:

   - Bots cannot change the physical permanent MAC address as it is burned in the ethenet/wireless device.
   - Bots cannot change the permanent MAC address value when the APIV client retrieves it from OS kernel unless the kernel is compromised. Since bot prefers a low presence, it will not hack the OS kernel as it has a high risk of being detected by firewall and other detection tools.
   - Bots cannot change the permanent MAC address value while it is being transferred to server as the permanent MAC address is encrypted with a session key that is random and unique at a time and unknown to the bot.

   (b) enhanced Captcha: bots cannot solve our enhanced Captcha as explained in Task 4.

   As these features cannot be spoofed by bots, it is very hard for Web 2.0 bots to bypass our system detection. Unlike many detection systems that base on historical data, our system do not need to update these features for detecting variants of future Web 2.0 bots.

4. Define a collection of security measures further on called attacks to check the security of our proposed system.

   We have defined a collection of attacks for both the API Verifier and the enhanced Captcha. We have also tested our system against these attacks to evaluate the systems security. The results can be summarized as follows:

   For the API Verifier

   There are three ways that a bot can attempt to bypass our detection system:

(a) Spoof permanent MAC address: as our system bases on the permanent MAC address to identify if an API call is from a new computer, a bot can try to replace the permanent MAC address value with the one of a computer on which the bot already successfully made API calls before. These attacks can take place in three places as explained in task 3. Also from the experiment results in Chapter 6, we can conclude that our system is secure against these attacks.

(b) APIV client fraud: bot can try to replace our APIV client with its own to bypass the MAC address check. However, as we use a secret key and AES to encrypt communication between APIV client and APIV server, it is hard for a bot to replicate APIV client if it cannot recover the secret key. Apart from using AES, we also use obfuscation to prevent bots from retrieving the key by using reverse-engineering. Since it is very hard to break AES and obfuscation, we are not able to conduct testing for this security attack. It may require another study to fully address this matter.

(c) Solve Captcha: bot can try to solve the Captcha challenge to be verified. Therefore, we have designed a new Captcha to ensure that bot cannot solve it even using relay attack. The analysis and experiment results are summarised in the next section.

For the enhanced Captcha

We have defined three major attacks on our enhanced Captcha: dictionary attack, image recognition attack and relay attack. The emphasis is on relay attack as it is recently popular and has high success rate. We believe that our Captcha is secure against the above attacks based on the following analysis and experiment results:

(a) Dictionary attack: Since our Captcha uses 6 images and the answers must be arranged in order, the chance of guessing a correct answer is less than 1%, which is considered secure against this type of attack.

(b) Relay attack: The experiment results in Chapter 6 shows that Koobface bot fails relay attack on our Captcha design. The shortest time for Koobface bot to receive an answer is 14 seconds. On the other hand, the objects in the Captcha already change their positions and labels after the delay time (10 seconds) collapses. The delay time is determined in a simulation mimicking the relay attack in real time. Therefore, we believe that it is quite safe to conclude that a bot cannot conduct a relay attack in less than 10 seconds.

(c) Image recognition attack: Though we do not conduct a security test against image recognition attack, it is believed that it is much harder for an automatic program to recognise image than text [125–127]. We have also improved its security by introducing random and small noise to the image, using random labels, careful choice of images and using significant large database. Instead of manually distorting the images and storing them in a database, the noise insertion is carried out automatically and randomly at the time that the image is processed and embedded to a Captcha challenge. Therefore, the noise level for each image at different time is different and harder to auto-recognised by a bot. In general, it is hard to evaluate this security measurement until there is successful image recognition attack on the real system.

In conclusion, our solution meets the defined research goals. Our system is able to filter API calls from bots Web services; hence, is able to protect Web services from being abused by bots to propagate or establish communication. We also define a collection of attacks and conduct some experiment attacks to test our system security. The results show that our system is secure against all defined attacks.

In addition, our system can be extended to notify users and/or network administrators about suspicious API calls made from their computers. As mentioned in Section 6.3, the API Verifier can feature an option to send notification email to the user for every API call made. By using this feature, a user who never uses API will be alerted that their computer is infected if there are API call attempts from their computer. For those who usually use API to auto-update their profiles, they will be also alerted if there are strange updates. Users can turn off this option if they wish to. The API Verifier can also provide a log file which can help administrators to monitor all API calls.

However, the API Verifier has the following drawbacks:

1. The API Verifier fails to detect API calls from bots if

   - the bot uses victim's credentials to make API call to Web services and
   - the victim used to make API call before using the same credentials and computer

   The API Verifier fails in this case because the victim already passed the verification the first time they use API on their computer. Therefore, all API calls afterward from that computer will be accepted. However, as the number of users using API is relative small, the failure rate of API Verifier is acceptable. Moreover, with the notification option as described above, a victim will be notified and/or notice suspicious updates on his/her profile and remove the bot.

2. The API Verifier may annoy legitimate users as they need to solve a Captcha for their first API call on a new computer. Due to logical consequence of our design principles, a user always needs to solve a Captcha when they use a new computer to auto-update their profile. However, after bypassing the verification, they will not encounter another Captcha if still using the same computer. To improve the usability, the number of MAC addresses associated with a user is configurable so that a user who wants to use multiple computers to auto-update their profile do not need to solve Captcha every time they switch between their computers.

7.1.2 Critical Assessments on our Solution

The results we have obtained are promising and meet our defined goals, but there are some aspects that should be considered for further improvements:

- APIV client is not compatible with multiple platforms. In our study, we only work on Windows platform and our prototype of APIV client is only compatible with Windows XP and Vista while in practice, users can use other OS for their computers (eg Mac, Linux) and other devices such as mobile phones and tablets. Each OS may need different method to get the permanent MAC address. Therefore, further study is required to be able to implement other versions of APIV client that are compatible with many platforms.

  In addition, our solution works under assumption that the OS kernel is hard to compromise. This assumption is acceptable for PCs as PCs kernel are usually well guarded with firewall and anti-virus software. However, this assumption is not necessary true for mobile phone and tablet OS as anti-virus software for these devices does not receive much attention and users tend to not use any anti-virus software for their mobile phones and/or tablets. Fortunately, cybercriminals are still not yet much interested in mobile devices either. However, to ensure our solution work effectively on mobile devices too, further study about mobile phone botnet as well as how to protect mobile phones from malware are needed.

- Lack of testing on some security measurement. We do not test our system against some defined attacks such as reverse-engineering attack on APIV client and image recognition attack on the enhanced Captcha. As it is difficult to set up such experiments, we cannot cover these attacks in this study. Therefore, to improve our system security, further study on obfuscation and image recognition attack should be considered in future studies.

- Small Sample. Due to limited resources, data collected from our simulation and experiments are relative small. It would be more realistic if we can have more attendees in the simulation to define a suitable delay time for the animation Captcha. However, the delay time is customizable so if we want to improve security, we can assign it with a smaller value. It is also better if we can run the security test against more Web 2.0 bots than only Koobface.

7.2 Future Work

For future research, the above critical points should be taken into account to improve the API Verifier. Besides studying techniques to get permanent MAC address from different platforms, there are some directions that can be considered for our future study.

7.2.1 Mobile phone botnet

As mentioned in Chapter 2, mobile phones can be the main target of future botnet. Threats to mobile devices have been a hot topic within the security community for several years, yet there is a little body of publication about mobile botnet structures as well as possible detection techniques. Though mobile phone botnets are not a significant threat right now, we expect attacks to explode at any time. There are some interesting topics that we can work on:

1. Mobile phone challenges: there are a number of challenges of mobile phones so that it takes quite a long time for mobile phone botnet to be a security problem. Some of the challenges are: lack of IP address, constant change of connectivity, platform diversity and communication cost (SMS cost). It is interesting to analyse these challenges and predict some techniques that bot writers would use to address the challenges.

2. C&C models: It is always important to learn C&C model of a botnet. As smart phones now can be as powerful as a PC, mobile phone bots can be developed based on existing PC bots. Therefore, possible C&C models can be P2P, HTTP, SMS or hybrid of these models. P2P and HTTP models can be similar to those used by PC botnets, yet SMS model introduces a new dimension for controlling the bots. SMS seems to be a promising model as it is hard to investigate. The main reason is that SMS service providers may not allow to monitor it. In addition, SMS is always available even when the device is turned off. The message will be delivered once the device is turned on again. Hybrid model such as HTTP

and SMS is also promising. Bots can propagate through SMS with phishing links contained in the message and update themselves via HTTP protocol.

3. Possible detections of mobile phone botnets: some possible detections can be:

    - Text analysis: to look for suspicious links and spam and filter these messages. This technique can be employed at the provider side to filter spam message before delivering it. It can also be utilised in a simple detection tool to filter spam in user SMS inbox.
    - Application to prevent malicious installation: this application can be a simple anti-virus software for mobile phones with signature-based techniques to alert users with installing malware to their devices.
    - Application to detect suspicious behaviours such as sending SMS to multiple destinations or in period or suspicious processes that occupy CPU and memory of the device.

7.2.2 Obfuscation

We can study further about obfuscation to improve our system security against reverse-engineering attack. It is interesting that obfuscation can be more efficient on some programming languages than others. Therefore, it is important to find out:

1. available obfuscation techniques and how they work,
2. attacks on obfuscation and how to cope and improve obfuscation techniques,
3. which programming language and obfuscation method are most suitable for our implementation,
4. possible improvement to existing obfuscation techniques or propose new obfuscation techniques.

7.2.3 TPM

We can also work on Trusted Platform Module (TPM), a protected and encapsulated microcontroller security chip. As mentioned in Section 4.6.1, we can use the endorsement public key of TPM as an alternative for a permanent MAC address. It is claimed that TPM is highly secure against external software attacks, therefore it may be more secure if we use the endorsement key of TPM and secret keys generated from TPM. However, we currently cannot utilize TPM as it is still not used by majority of PCs and it is hard to communicate with. If working on this direction, there are some topics that we can consider:

1. develop an intermediate user interface to interact with TPM. Currently, majority of users cannot activate and interact with TPM as it requires programming skills to understand the TPM specification and its APIs. If we can develop a UI for users to click and point to make request to TPM such as generating keys or storing sensitive information to the chip, it will be a significant facility to make TPM more accessible and practical.

2. security analysis on TPM. There is little work on security analysis on TPM. This may due to the fact that it is hard to interact and set up experiment with TPM.

8

An Appendix

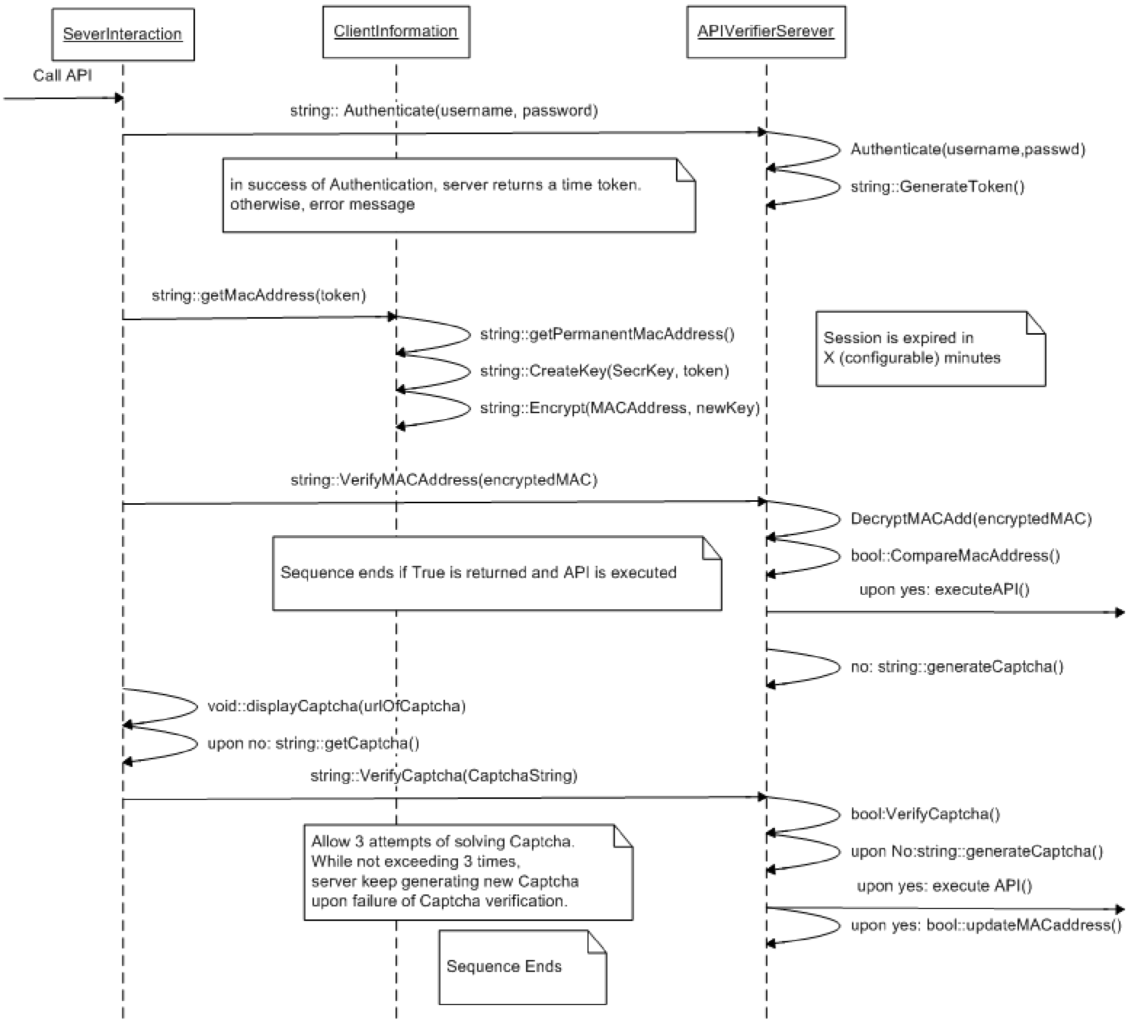

SeverInteraction
ClientInformation
APIVerifierSerever
Call API
string:: Authenticate(username, password)
Authenticate(username,passwd)
string::GenerateToken()
in success of Authentication, server returns a time token.
otherwise, error message
string::getMacAddress(token)
string::getPermanentMacAddress()
string::CreateKey(SecrKey, token)
string::Encrypt(MACAddress, newKey)
Session is expired in
X (configurable) minutes
string::VerifyMACAddress(encryptedMAC)
DecryptMACAdd(encryptedMAC)
bool::CompareMacAddress()
Sequence ends if True is returned and API is executed
upon yes: executeAPI()
no: string::generateCaptcha()
void::displayCaptcha(urlOfCaptcha)
upon no: string::getCaptcha()
string::VerifyCaptcha(CaptchaString)
bool:VerifyCaptcha()
Allow 3 attempts of solving Captcha.
While not exceeding 3 times,
server keep generating new Captcha
upon failure of Captcha verification.
upon No:string::generateCaptcha()
upon yes: execute API()
upon yes: bool::updateMACaddress()
Sequence Ends

Figure 8.1: The API Verifier Sequence Diagram

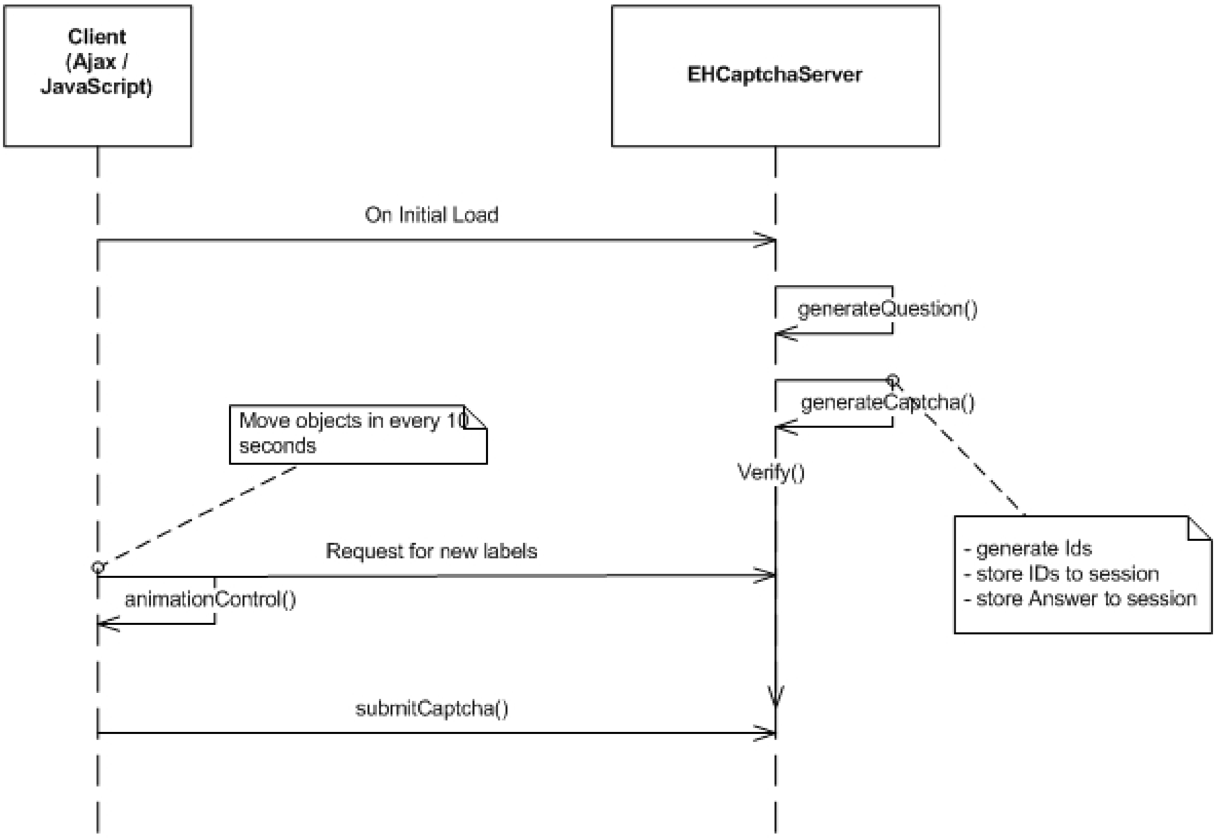


FIGURE 8.2: The Enhanced Captcha Sequence Diagram

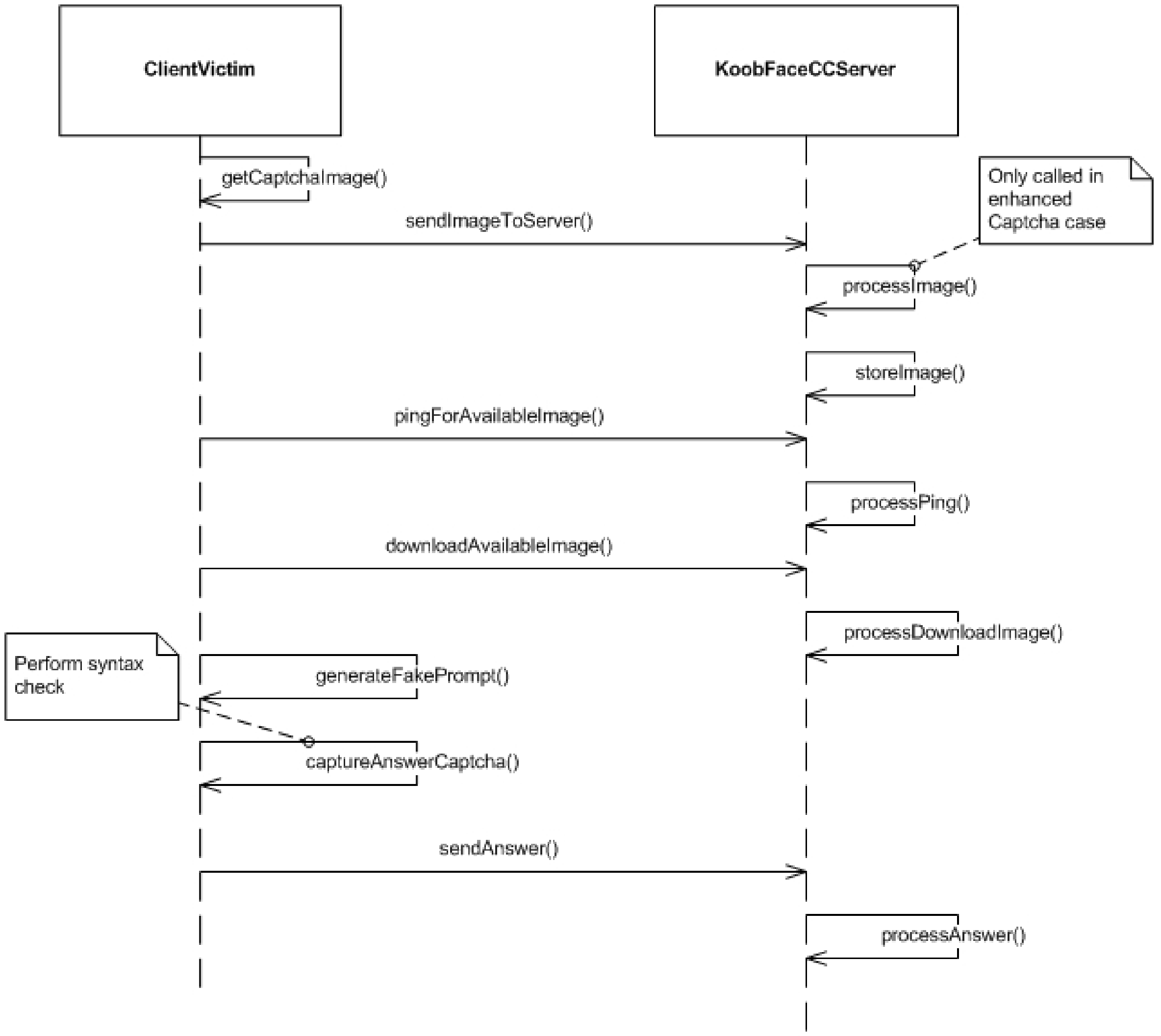


FIGURE 8.3: The Simplified Koobface Sequence Diagram

Function name	**Description**	**Class**
`private string GetPermanentMACAddress()`	Argument: none Return: This function returns a string of permanent MAC address of Ethernet/Wireless card. If cannot get MAC Address, return empty string	ClientInformation
`string CreateKey(string, string)`	Arguments: string: secret key, string: time token Return: a new key. This function is to combine the secret key and the time token to create a new key.	ClientInformation
`string Encrypt(string, string)`	Arguments: string: message, string: key Return: encrypted message This function is to encrypt message using provided key and return an encrypted message. Encryption algorithm can be RSA or ECC	ClientInformation
`public string GetMACAdress(string)`	Argument: string: time token Return: encrypted permanent MAC Address This function will call the above functions and return an encrypted permanent MAC Address.	ClientInformation This function is a public function to be called by ServerInteraction.

Function name	**Description**	**Class**
`string Authenticate(string, string, string)`	Argument: string: url of server, string: encrypted username, string: encrypted password Return: token or message code This function makes http connection to server and post username, password to server. Return: - time token (upon successful connection and authentication) - or error message: “connection fail” or “authentication fail”	ServerInteraction
`string VerifyMACAddress(string, string, string)`	Argument: string: url of server, string: encrypted MAC Address, string: API data Return: successful message or url of Captcha. This function makes connection and sends encrypted MAC address to server for verification and API data. Server will compare this MAC Address with the one associated with user stored in the database. If they are matched, return successful message. Otherwise, return url of Captcha.	SeverInteraction

Function name	**Description**	**Class**
`string GetCaptcha()`	Argument: None Return: Captcha string This function displays Captcha using returned url from VerifyMACAddress and a message box requesting user to enter Captcha string displayed on browser. The function returns Captcha string entered by the user.	ServerInteraction
`string GetEnhancedCaptcha()`	Argument: None Return: string contain IDs of answered images This function generate a Captcha challenge (the enhanced Captcha) and displays Captcha using returned url from VerifyMACAddress. When user drags and drops answers to answer box and submit, the function return IDs of these images.	ServerInteraction
`string VerifyCaptcha(string, string)`	Argument: string: url of server, string: Captcha string Return: successful message or url of (new) Captcha or error message. This function makes connection and sends Captcha string to server for verification. In return: • successful message string if Captcha string is correct	SereverInteraction

Function name	**Description**	**Class**
	- url of a new Captcha if it is not correct - error message string if user has exceeded 3 attempts of solving Captcha.	
`bool authenticate(string, string)`	Argument: string: username, string: password Return: true or false This function validates username and password	APIVerifierServer
`string GenerateToken()`	Argument: none Return: string time token This function generates a random time token.	APIVerifierServer
`string CreateKey(string, string)`	Arguments: string: secret key, string: time token Return: a new key This function is to combine the secret key and the time token to create a new key.	APIVerifierServer
`string Decrypt(string, string)`	Arguments: string: message, string: key Return: decrypted message. This function is to decrypt message using provided key and return a message. Encryption algorithm can be RSA or ECC.	APIVerifierServer

Function name	**Description**	**Class**
`bool CompareMACaddress(string, string)`	Argument: string: MAC address, string: username Return: true or false. This function queries existing MAC address associated with username from server's database and compare with passed in MAC Address. If they are matched return true, otherwise, return false.	APIVerifierServer
`string GenerateCaptcha()`	Argument: None Return: string: url of the Captcha. This function generates a Captcha string and image, stores the Captcha file in server and returns an url of the Captcha.	APIVerifierServer
`bool VerifyCaptcha(string)`	Argument: Captcha string Return: true or false This function verifies if the received Captcha string is matched with the originally generated one. Return true or false.	APIVerifierServer
`bool updateMACAddress(string, string)`	Argument: string: username, string: MACAddress Return: true or false This function store new MAC Address of the provided user. The function returns true if successfully update the information, otherwise return false.	APIVerifierServer

Table 8.1: Function List of API Verifier

Function name	**Description**	**Class**
`void generateQuestion()`	Argument: none Return: none This function: • gets 6 random images • generates random image labels • selects three of the six labels of the images to be the answers • generates a question	string EHCaptchaServer
`string generateCaptcha()`	Argument: none Return: String of HTML DOM This function • distort 6 images chosen in generateQuestion • store answer (labels) • form Captcha challenge • stores Ids to session	EHCaptchaServer

Function name	**Description**	**Class**
`bool Verify()`	Argument: none Return: True of False if false call the generateQuestion() and generateCaptcha () to recreate Captcha.	EHCaptchaServer
`submitCaptcha(string)`	Argument: string Return: none This function submits the answer string submitted by the user to server for verification.	Client (Ajax/JavaScript)
`animationControl()`	`Argument`: none `Return`: none In every 10 seconds, sends request to server to get new labels and move objects in the Captcha.	Client (Ajax / JavaScript)

Table 8.2: Function List of the Enhanced Captcha

Function name	**Description**	**Class**
`ImageStream getCaptchaImage()`	Argument: none Return: This function returns a ImageStream of the Captcha image.	ClientVictim
`bool sendImageToServer (ImageStream)`	Argument: ImageStream Return: True or False (successful or fail to send the image)	ClientVictim
`bool pingForAvailableImage()`	Argument: none Return: True or False (new available image or not)	ClientVictim
`ImageStream DownloadAvailableImage()`	Argument: none Return: This function returns a ImageStream of the Captcha image	ClientVictim
`generateFakePrompt()`	Argument: none Return: none A fake form will be generated with presenting the Captcha image and text input for answering Captcha	ClientVictim
`string captureAnsweredCaptcha()`	Argument: none Return: answer string that user enter from the fake form. This function also does the simple predefined syntax checking for the entered Captcha.	ClientVictim
`bool sendAnwer(String)`	Argument: answer string Return: True or False (Success or fail to send an answer to server)	ClientVictim
`string storeImage(ImageStream, HttpRequest)`	Argument: ImageStream, HttpRequest (client request information) Return: String location of the image to be stored in the server.	KoobFaceCCServer

Function name	**Description**	**Class**
`list processPing(HttpRequest)`	Argument: HttpRequest Return: List of available image.	KoobFaceCCServer
`ImageStream processDownload (HTTPRequest)`	Argument: HTTPRequesr Return: ImageStream to client	KoobFaceCCServer
`void processAnswer(HTTPRequest)`	Argument: HTTPRequesr Return: none	KoobFaceCCServer
`ImageStream ProcessImage(ImageStream)`	Argument: ImageStream Return: ImageStream after processed This function is to crop the input image, get relevant information and form a new image for the fake Captcha.	KoobFaceCCServer (this function is only called in the case of enhanced Captcha)

Table 8.3: Function List of the Simplified Koobface

Glossary

The following list is neither exhaustive nor exclusive, but may be helpful.

Symbol	**Definitions**
API	Application Programming Interface. Set of routines and and tools which help an application to request services from the operating system.
C&C	Command and Control. A structure used to send commands by the attacker and receive response from the bots.
DDoS	Distributed Denial of Service attack. An attack carried out when multiple hosts infected by trojan viruses consume the bandwidth of a network or a system by generating large number of packets causing a denial of service.
DLL	Dynamic Link Library. A library for shared executable functions between Windows applications.
DNS	Domain Name Server. A service which responsible for translating domain names into numeric IP addresses.
HTTP	Hyper Text Transfer Protocol used by World Wide Web (WWW) to communicate with webservers and browsers.

Symbol	**Definitions**
IP	Internet Protocol. Network layer protocol which specifies the format of packets.
ISP	Internet Service Provider.
IRC	Internet Relay Chat. A special protocol for real-time chat. The botmasters use this protocol to control their bots.
MAC address	Media Access Control address of the ethernet/wireless device.
OS	Operating System.
PC	Personal Computer.
P2P	Peer to peer. A computer network in which each computer in the network can act as a client or server for the other computers in the network, allowing shared access to files and peripherals without the need for a central server.
SSL	Socket Secure Layer. The standard security technology for establishing an encrypted link between a web server and a browser.
EV-SSL	Extended Validation SSL.
TCP	Transmission Control Protocol. The suite of communications protocols used to connect hosts on the Internet.
TTL	Time-To-Live. A field in IP which specify how many hops a packet can visits before being discarded.

TABLE 8.4: Glossary

List of Publications

- Hanh Nguyen Vo and Josef Pieprzyk *Protecting Web 2.0 services from botnet exploitations.* In Cybercrime and Trustworthy Computing Workshop (CTC), p 18 - 28 (2010)

References

[1] C. A. Schiller, J. Binkley, D. Harley, G. Evron, T. Bradley, C. Willems, and M. Cross. *Botnets: The Killer Web App* (Syngress, 2007). 1, 7, 18, 23, 25, 41, 48, 50, 82

[2] eWeek. *Online eweek magazine* (2009). URL `http://www.eweek.com`. 1, 4

[3] S. Walsh. *Spam still profitable to the tune of $4000 a day* (2009). URL `http://www.allspammedup.com/2009/09/spam-still-profitable-to-the-tune-of-4000-a-day/`. 2

[4] K. Charles. *Phishing overview in the united states* (2010). URL `http://securityorb.com/2010/10/phishing-overview-in-the-united-states/`. 2

[5] Y. Jin, Z. Zhang, and K. Xu. *Identifying and tracking suspicious activities through ip gray space analysis*. In *The 3rd annual ACM workshop on Mining network data MineNet* (2007). 3, 7, 83

[6] M. Walfish, M. Vutukuru, H. Balakrishnan, D. Karger, and S. Shenker. *Ddos defense by offense*. In *the 2006 conference on Applications, Technologies, Architectures, and Protocols for computer communications*, vol. 36 of *ACM SIGCOMM Computer Communication Review*, pp. 303–314 (2006). 3, 8, 33

[7] Y. Xiang and W. Zhou. *An intrusion surveillance system to detect irc-based ddos attacks*. In *The International Multi-Conference on Computing in the Global Information Technology*, pp. 65–70 (2007). 3, 8

[8] McAfee. *How i become a zombie* (2011). URL http://images.mcafee.com/infographic/zombie.html?cid=93629. 3

[9] McAfee. *Mcafee threats report: Second quarter 2009.* Tech. rep., McAfee (2009). URL http://mcafee.com/us/resources/reports/rp-quarterly-threat-q2-2009.pdf. 3

[10] Symantec. *Symantec internet security threat report: Trends for january-june 07* (2007). URL http://eval.symantec.com/mktginfo/enterprise/white_papers/ent-whitepaper_internet_security_threat_report_xii_09_2007.en-us.pdf. 3

[11] Wikipedia.org. *Botnet* (2010). URL http://en.wikipedia.org/wiki/Botnet. 3

[12] Spamhaus (2010). URL http://www.spamhaus.org./. 4, 33

[13] BBCNews. *Microsoft aids shutdown of rustock spam net* (2011). URL http://www.bbc.co.uk/news/technology-12772319. 4, 5, 34

[14] Symantec. *Symantec's march 2011 messagelabs intelligence report.* Tech. rep., Symantec (2011). URL http://www.symanteccloud.com/mlireport/MLI_2011_03_March_Final-EN.pdf. 4, 27, 34

[15] Mailshine.com. *What is the global cost of spam?* (2011). URL http://www.mailshine.com/2011/06/whats-the-global-cost-of-spam/. 4

[16] Ponemone. *Second annual cost of cyber crime study. benchmark study of u.s. companies.* Tech. rep., Ponemone (2011). URL http://www.arcsight.com/collateral/whitepapers/2011_Cost_of_Cyber_Crime_Study_August.pdf. 4

[17] G. Goth. *Fast-moving zomies: Botnets stay a step ahead of the fixes.* IEEE Internet Computing Magazine **11**, 7 (2007). 4

[18] B. McCarty. *Botnets: Big and bigger.* IEEE Security and Privacy Magazine **1**, 87 (2003). 81

[19] G. P. Schaffer. *Worms and viruses and botnets, oh my! rational responses to emerging internet threats*. IEEE Security & Privacy Magazine **4**, 52 (2006). 4

[20] P. Bacher, T. Holz, M. Kotter, and G. Wicherski. *Know your enemy: Tracking botnets* (2005). URL http://www.honeynet.org/papers/bots. 4, 24, 33, 34, 81

[21] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. *Bothunter: Detecting malware infection through ids-driven dialog correlation*. In *16th USENIX Security Symposium* (2007). 4, 7, 80

[22] T. Brewster. *Microsoft takes credit for rustock shutdown* (2012). URL http://www.itnews.com.au/News/290922,how-enterprises-can-help-stamp-out-spambots.aspx. 5

[23] R. Westervelt. *Botnet masters turn to google, social networks to avoid detection* (2009). URL http://searchsecurity.techtarget.com/news/1373974/Botnet-masters-turn-to-Google-social-networks-to-avoid-detection. 5, 63, 78

[24] J. Nazario. *Twitter based botnet command and control*. Tech. rep., ARBOR SERT (2009). URL http://ddos.arbornetworks.com/2009/08/twitter-based-botnet-command-channel/. 6, 78

[25] A. Lelli. *Trojan.whitewell: Whats your (bot) facebook status today?* (2009). URL http://www.symantec.com/connect/blogs/trojanwhitewell-what-s-your-bot-facebook-status-today. 6

[26] G. O. Gorman. *Google groups trojan* (2009). URL http://www.symantec.com/connect/blogs/google-groups-trojan. 6, 63, 78

[27] E. Stinson and J. C. Mitchell. *Characterizing bots remote control behavior*. In *DIMVA*, vol. 4579 of *LNCS*, p. 89108 (2007). 7

[28] Z. Zhu, V. Yegneswaran, and Y. Chen. *Using failure information analysis to detect enterprise zombies*. In *Securecomm* (2009).

[29] P. Barford and V. Yegneswaran. *An inside look at botnets*. In *Special Workshop on Malware Detection*, Advances in Information Security (2006).

[30] F. Y. W. Law, K. P. Chow, P. K. Y. Lai, and H. K. S. Tse. *A host-based approach to botnet investigation*. In *International Conference on Digital Forensics and Cyber Crime* (2009). 7

[31] C. Livadas, R. Walsh, D. Lapsley, and W. T. Strayer. *Using machine learning techniques to identify botnet traffic*. In *31st IEEE conference on Local Computer Networks*, pp. 967–974 (2006). 7, 79

[32] W. T. Strayer, R. Walsh, C. Livadas, and D. Lapsley. *Detecting botnets with tight command and control*. In *31st IEEE Conference on Local Computer Networks*, pp. 195 – 202 (2006). 79

[33] J. R. Binkley and S. Singh. *An algorithm for anomaly-based botnet detection*. In *Reducing Unwanted Traffic on the Internet* (2006).

[34] J. Goebel and Holz. *Rishi: identify bot contaminated hosts by irc nickname evaluation*. In *HotBots* (2007).

[35] G. Gu, J. Zhang, and W. Lee. *Botsniffer: Detecting botnet command and control channels in network traffic*. In *NDSS* (2008). 81

[36] G. Gu, R. Perdisci, J. Zhang, and W. Lee. *Botminer: Clustering analysis of network traffic for protocol- and structure-independent botnet detection*. In *Security* (2008).

[37] E. Cooke, F. Jahanian, and D. McPherson. *The zombie roundup: Understanding, detecting, and disrupting botnets*. In *Steps to Reducing Unwanted Traffic on the Internet Workshop* (2005). 7, 47

[38] J. Baltazar, J. Costoya, , and R. Flores. *The real face of koobface: The largest web 2.0 botnet explained*. A Trend Micro White Paper (2009). URL `http://us.trendmicro.com/imperia/md/content/us/trendwatch/`

researchandanalysis/the_real_face_of_koobface_jul2009.pdf. 8, 12, 87, 146, 147

[39] X. Geng and A. B. Whinston. *Defeating distributed denial of service attacks.* IEEE IT Professional Magazine (2000). 8, 33

[40] F. W. Security. *Finjan web security trends report* (2009). URL http://www.m86security.com/labs/web-security-trends-reports.asp. 11, 75, 76

[41] A. Rajab, J. Zarfoss, F. Monrose, and A. Terzis. *A multifaceted approach to understanding the botnet phenomenon.* In *the 6th ACM SIGCOMM Conference on Internet Measurement*, p. 52 (2006). 20

[42] Jones and Jim. *BotNets: Detection and Mitigation* (FEDCIRC, 2003). 20

[43] Trend-Micro. *Taxonomy of botnet threats.* A Trend Micro White Paper (2006). URL http://us.trendmicro.com/imperia/md/content/us/pdf/threats/securitylibrary/botnettaxonomywhitepapernovember2006.pdf. 20, 37, 41, 44, 47, 48, 50

[44] D. Fowler. *Netnews.* netWorker **3**, 7 (1999). 20

[45] mIRC. *mirc: Internet relay chat client* (2011). URL http://www.mirc.com/. 23

[46] M. Romano, S. Rosignoli, and E. Giannini. *Robot wars - how botnets work* (2005). URL http://www.windowsecurity.com/articles/Robot-Wars-How-Botnets-Work.html. 25

[47] J. Canavan. *The evolution of malicious irc bots. white paper: Symantec security response.* In *the VB2005 Conference*, pp. 104–114 (2005). 25, 33

[48] P. Wang, S. Sparks, and C. C. Zou. *An advanced hybrid peer-to-peer botnet* (2007). URL http://www.usenix.org/event/hotbots07/tech/full_papers/wang/wang.pdf. 25

[49] T. HOLZ, M. STEINER, F. DAHL, E. BIERSACK, and F. FREILING. *Measurements and mitigation of peer-to-peer-based botnets: A case study on storm worm*. In *the First USENIX Workshop on Large-Scale Exploits and Emergent Threats* (2008). 25, 26, 33, 59, 61

[50] G. Starnberger, C. Kruegel, and E. Kirda. *Overbot: a botnet protocol based on kademlia*. In *the 4th international conference on Security and privacy in communication netowrkse*, pp. 1–9 (2008).

[51] M. Bowden. *The enemy within* (2010). URL `http://www.theatlantic.com/magazine/archive/2010/06/the-enemy-within/8098/`. 25

[52] X. Li, H. Duan, W. Liu, and J. Wu. *Understanding the construction mechanism of botnets*. In *Ubiquitous, Autonomic and Trusted Computing*, pp. 508 – 512 (2009). 25

[53] J. Stewart. *Sinit p2p trojan analysis* (2003). URL `http://www.securiteam.com/securityreviews/6J0022A95E.html`. 26

[54] S. Golovanov and I. Soumenkov. *Tld4 - top bot* (2011). URL `http://www.securelist.com/en/analysis/204792180/TDL4_Top_Bot`. 26, 61, 62

[55] G. Hoglund and J. Butler. *Rootkits : Subverting the Windows Kernel* (Addison-Wesley Professional, 2005). 26, 61

[56] R. Sharp. *An Introduction to Malware* (Spring, 2008). 26, 29, 30, 31, 32, 61

[57] Z. Bu, P. Bueno, R. Kashyap, and A. Wosotowsky. *The new era of botnets*. McAfee Labs Technical White Papers (2010). URL `http://www.mcafee.com/us/resources/white-papers/wp-new-era-of-botnets.pdf`. 27

[58] Team-Cymru. *Developing botnet ... an analysis of recent activity* (2010). URL `http://www.team-cymru.com/ReadingRoom/Whitepapers/2010/developing-botnets.pdf`. 27

[59] McAfee. *2011 threats predictions*. Tech. rep., McAfee (2011). URL http://www.mcafee.com/us/resources/reports/rp-threat-predictions-2011.pdf. 27, 87

[60] J. Baltazar. *Web 2.0 botnet evolution koobface revisited*. Tech. rep., Trend Micro (2010). 27, 31, 74

[61] E. H. Spafford. *The internet worm program: An analysis*. Tech. rep., Purdue University, West Lafayatte (1988). 29, 31

[62] J. Nazario, J. Anderson, R. Wash, and C. Connelly. *The future of internet worms*. Tech. rep., Crimelabs (2001). 29, 30

[63] N. Provos, P. Mavrommatis, M. A. Rajab, and F. Monrose. *All your iframes point to us*. In *the 17th Conference on Security Symposium*, pp. 1–15 (2008). 30

[64] B. Krebs. *Storm worm dwarfs worlds top supercomputers* (2007). URL http://blog.washingtonpost.com/securityfix/2007/08/storm_worm_dwarfs_worlds_top_s_1.html. 31

[65] Ponemon. *2008 annual study: Cost of a data breach*. Tech. rep., Ponemon (2009). 33

[66] S. Baker, S. Waterman, and G. Ivanov. *In the crossfire*. Tech. rep., McAfee (2010). URL http://img.en25.com/Web/McAfee/NA_CIP_RPT_REG_2840.pdf. 33

[67] F. C. Freiling, T. Holz, and G. Wicherski. *Botnet tracking: Exploring a root cause methodology to prevent distributed denial-of-service attacks*. In *the European Symposium on Research in Computer Security*, pp. 319–335 (2005). 33, 34

[68] Fastcompany.com (2010). URL http://www.fastcompany.com/magazine/156/25th-anniversary-of-listserv. 33

[69] J. Stewart. *Bobax trojan analysis.* Tech. rep. (2004). URL http://www.secureworks.com/research/threats/bobax/. 33

[70] K. Chiang and L. Lloyd. *A case study of the rustock rootkit and spam bot.* In *the First USENIX Workshop on Hot Topics in Understanding Botnets* (2007). 34, 63

[71] brighthub.com (2011). URL http://www.brighthub.com/internet/security-privacy/articles/99607.aspx. 34

[72] N. Ianelli and A. Hackworth. *Botnets as a vehicle for online crime.* CERTCoordination Center (2005). 34

[73] R. Vogt, J. Aycock, and M. Jacobson. *Army of botnets.* In *the 2007 Network and Distributed System Security Symposium*, p. 111123 (2007). 34

[74] N. DASWANI and M. STOPPELMAN. *The anatomy of clickbot.a.* In *the First USENIX Workshop on Hot Topics in Understanding Botnets* (2007). 34

[75] B. Acohido. *Botnet-driven click fraud attacks pilfering millions from advertisers* (2010). URL http://lastwatchdog.com/botnet-driven-click-fraud-steals-millions-advertisers/. 34

[76] G. Ollmann. *Botnet communication topologies.* The Damballa white paper (2009). 37, 39

[77] J. Baltazar, J. Costoya, and R. Flores. *The heart of koobface: C&c and social network propagation.* A Trend Micro White Paper (2009). 42, 56, 74, 87, 146

[78] J. Nazario. *Bot and botnet taxonomy.* Tech. rep., Computer Security Institute (2008). 44

[79] J. B. Grizzard, V. Sharma, C. Nunnery, B. B. Kang, and D. Dagon. *Peer-to-peer botnets: Overview and case study.* In *the First USENIX Workshop on Hot Topics in Understanding Botnets*, pp. 1–1 (2007). 44

[80] G. Sinclair, C. Nunnery, and B. B. Kang. *The waledac protocol: The how and why*. In *4th International Conference on Malicious and Unwanted Software*, pp. 69 – 77 (2009). 44, 47, 62, 63

[81] Z. Zhang, B. Lu, P. Liao, C. Liu, and X. Cui. *A hierarchical hybrid structure for botnet control and command*. In *IEEE International Conference on Computer Science and Automation Engineering*, vol. 1, pp. 483–489 (2011). 44

[82] Dyndns. *Dynamic network services. dyndns: Free dns hosting*. URL http://www.dyndns.com/. 49

[83] IRChelp.org. *Irc networks and server lists* (2011). URL http://www.irchelp.org/irchelp/networks/. 50

[84] hakin9. *Robot wars how botnets work* (2005). URL http://www.windowsecurity.com/articles/robot-wars-how-botnets-work.html. 51, 52

[85] HoneynetProject. *Botnet commands - which commands the bots understand* (2008). URL http://www.honeynet.org/node/55. 52

[86] A. Rollin. *P2p protocol* (2011). URL http://p2pfoundation.net/P2P_Protocol. 57

[87] J. Zhang. *Stormworm & botnet analysis*. Tech. rep., Websense (2008). URL http://securitylabs.websense.com/content/Assets/Storm_Worm_Botnet_Analysis_-_June_2008.pdf. 58, 59

[88] Waste. *Waste project* (2011). URL http://waste.sourceforge.net/index.php?id=projects. 59

[89] LURHQ. *Phatbot trojan analysis* (2011). URL http://www.lurhq.com/phatbot.html. 59

[90] OpenSVN. *The overnet protocol* (2011). URL https://opensvn.csie.org/mlnet/trunk/docs/overnet.txt. 59

[91] D. Dittrich and S. Dietrich. *P2p as botnet command and control: a deeper insight.* In *Malware08* (2008). 59

[92] D. il Jang, K. yu Cho, M. Kim, H. chul Jung, and B.-N. Noh. *Evasion technique and detection of malicious botnet.* In *Internet Technology and Secured Transactions*, pp. 1–5 (2010). 62

[93] The-Honeynet-Project. *Know your enemy: Fast-flux service networks.* Tech. rep., The Honeynet Project and Research Alliance (2008). URL http://www.honeynet.org/papers/ff. 62

[94] D. il Jang, J.-S. Lee, J.-H. Park, M. Kim, and B.-N. Noh. *Analysis of http-based malicious botnet(the cases of kraken botnet).* In *30th Fall conference* (2008). 63

[95] S. Murugesan. *Understanding web 2.0.* IT Professional **9**, 34 (2007). 68

[96] Facebook. *Facebook developer resources - graph api* (2012). URL http://developers.facebook.com/docs/reference/api/. 69, 70

[97] Facebook. *Facebook developer resources - permissions* (2012). URL http://developers.facebook.com/docs/authentication/permissions/. 70

[98] K. Maraju. *Web 2.0 attacks explained.* ITtoolbox Research (2008). URL http://hosteddocs.ittoolbox.com/KM111408.pdf. 72

[99] S. Shah. *Top 10 web 2.0 attack vectors.* net-square.com whitepapers (2006). URL http://net-square.com/whitepapers/Top10_Web2.0_AV.pdf. 72

[100] J. Moeder. *Yahoo rss xss vulnerability* (2005). URL http://seclists.org/bugtraq/2005/Oct/205. 72

[101] E. J. Kartaltepe, J. A. Morales, S. Xu, and R. Sandhu. *Social network-based botnet command-and-control: Emerging threats and countermeasures.* In *ACNS 2010*, p. 511528 (2010). 73, 87

[102] N. VILLENEUVE. *Koobface: Inside a crimeware network*. The Information Warfare Monitor Magazine (2010). URL http://www.infowar-monitor.net/reports/iwm-koobface.pdf. 74

[103] Google. *Blogger apis*. URL http://code.google.com/apis/blogger/. 77

[104] A. Chen, Y. Jin, J. Cao, and L. E. Li. *Tracking long duration flows in network traffic*. In *INFOCOM*, pp. 1–5 (2010). 79

[105] M. Akiyama, T. Kawamoto, M. Shimamura, T. Yokoyama, Y. Kadobayashi, and Yamaguchi. *A proposal of metrics for botnet detection based on its cooperative behavior*. In *International Symposium on Applications and the Internet Workshops*, pp. 82–85 (2007). 80

[106] B. McCarty. *Automated identity theft*. IEEE Security and Privacy Magazine **1**, 89 (2003). 81

[107] M. Vrable, J. Ma, J. Chen, D. Moore, E. Vandekieft, A. C. Snoeren, G. M. Voelker, and S. Savage. *Scalability, fidelity, and containment in the potemkin virtual honeyfarm*. In *20th ACM symposium on Operating systems principles*, pp. 148–162 (2006). 81

[108] C. C. Zou and R. Cunningham. *Honeypot-aware advanced botnet construction and maintenance*. In *International Conference on Dependable Systems and Networks*, pp. 199–208 (2006). 82

[109] C. Raiciu, M. Handley, and D. S. Rosenblum. *Exploit hijacking: side effects of smart defenses*. In *SIGCOMM workshop on Large-scale attack defense. Applications, Technologies, Architectures, and Protocols for Computer Communication*, pp. 123–130 (2006). 82, 83

[110] B. Carrier and C. Shields. *The session token protocol for forensics and traceback*. ACM Transactions on Information and System Security (TISSEC) **7**, 333 (2004). 83, 84, 85

[111] A. Ramachandran and N. Feamster. *Understanding the network-level behavior of spammers*. In *Sigcomm Conference* (2006). 83, 85

[112] P. Ferguson and D. Senie. *Network ingress filtering: Defeating denial of service attacks which employ ip source address spoofing*. RFC **2267** (1998). 83

[113] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. *Practical network support for ip traceback*. In *SIGCOMM*, pp. 295–306 (2000). 84

[114] A. Yaar, A. Perrig, and D. Song. *Stackpi: New packet marking and filtering mechanisms for ddos and ip spoofing defense*. In *IEEE Symposium on Security and Privacy* (2003). 84

[115] S. Staniford-Chen and L. T. Heberlein. *Holding intruders accountable on the internet*. In *IEEE Symposium on Security and Privacy*, pp. 39 – 49 (1995). 85

[116] trendmicro.com. *A look back at 2011: Information is currency*. Tech. rep., Trend Micro (2011). URL http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/reports/rpt_a-look-back-at-2011_information-is-currency.pdf. 87

[117] D. Addison. *Web Site Cookbook: Solutions & Examples for Building and Administering Your Web Site* (O'Reilly, 2006). 90, 139

[118] E. Bradford and L. Mauget. *Linux and Windows: A Guide to Interoperability* (Prentice Hall PTR, 2002). 93

[119] K. Lai. *Changing the mac address in w2k and xp*. URL http://www.nthelp.com/NT6/change_mac_w2k.htm. 93

[120] MSDN. *Oid_802_3_permanent_address*. URL http://msdn.microsoft.com/en-us/library/ff569074.aspx. 93

[121] Cisco. *What is the difference: Viruses, worms, trojans, and bots?* URL http://www.cisco.com/web/about/security/intelligence/virus-worm-diffs.html. 99

[122] D. Low. *Protecting java code via code obfuscation.* In *ACM* (1998). 101

[123] C. Linn and S. Debray. *Obfuscation of executable code to improve resistance to static disassembly.* In *CCS* (2003). 101

[124] G. Mori and J. Malik. *Breaking a visual captcha.* URL http://www.cs.sfu.ca/~mori/research/gimpy/. 102

[125] K. A. Kluever and R. Zanibbi. *Balancing usability and security in a video captcha.* In *ACM International Conference*, vol. 2 (2009). 102, 118, 123, 139, 142, 179

[126] J. S. Cui, J. T. Mei, X. Wang, D. Zhang, and W. Z. Zhang. *A captcha implementation based on 3d animation.* In *International Conference on Multimedia Information Networking and Security*, vol. 2, pp. 179–182 (2009). 118, 142

[127] A. Markkola and J. Lindqvist. *Accessible voice captchas for internet telephony.* In *The Symposium on Accessible Privacy and Security* (2008). 102, 123, 139, 179

[128] Microsoft. *How digest authentication works* (2012). URL http://technet.microsoft.com/en-us/library/cc780170(v=ws.10).aspx. 106

[129] A. L. Von, M. Blum, and J. Langford. *Telling humans and computer apart (automatically) or how lazy cryptographers do ai.* In *47th Comm. of the ACM*, vol. 2, pp. 56–60 (2004). 112, 123

[130] A. M. Turing. *Computing machinery and intelligence.* Mind **59**, 433 (1950). 112

[131] A. B. Jeng, C.-C. Tseng, D.-F. Tseng, and J.-C. Wang. *A study of captcha and its application to user authentication.* Lecture Notes in Computer Science **6422** (2010). 115, 123

[132] M. Chew and H. S. Baird. *Baffletext: A human interactive proof.* In *SPIEIS&T Electronic Imaging, Document Recognition and Retrieval X*, pp. 305–316 (2003). 115

[133] D. Goodin. *Google's recaptcha busted by new attack* (2009). URL http://www.theregister.co.uk/2009/12/14/google_recaptcha_busted/. 116

[134] M. Chew and J. D. Tygar. *Image recognition captchas*. Lecture Notes in Computer Science **3225**, 268 (2004). 116, 123, 139

[135] P. Golle. *Machine learning attacks against the asirra captcha*. In *CCS'08* (2008). 116, 139

[136] E. Bursztein, S. Bethard, J. C. Mitchell, D. Jurafsky, and C. Fabry. *How good are humans at solving captchas? a large scale evaluation*. In *IEEE Symposium on Security and Privacy* (2010). 117

[137] I. F. Ince and Y. B. Salman. *Execution time prediction for 3d interactive captcha by keystroke level model*. In *4th International Conference on Computer Sciences and Convergence Information Technology*, vol. 2, pp. 1057–1061 (2009). 118

[138] A. Desai and P. Patadia. *Drag and drop: A better approach to captcha*. In *Annual IEEE Digital Object Identifier* (2009). 119, 123

[139] S. K. Chaudhari, A. R. Deshpande, S. B. Bendale, and R. V. Kotian. *3d drag-n-drop captcha enhanced security through captcha*. In *the International Conference and Workshop on Emerging Trends in Technology* (2011). 119, 123

[140] SiteBlackBox. *Captcha 2.0* (2012). URL http://www.siteblackbox.com/captchaService.php. 120, 124

[141] Mersane. *keycaptcha* (2012). URL https://www.keycaptcha.com/. 124

[142] E. Athanasopoulos and S. Antonatos. *Enhanced captchas: Using animation to tell humans and computers apart*. In *Communications and Multimedia Security LNCS*, vol. 4237, pp. 97–108 (2006). 120, 123, 124, 125

[143] M. Sopiars and S. Whitley. *Captcha is dead!* (2008). URL http://www.scribd.com/doc/45768733/CAPTCHA-is-Dead-Final. 123, 139

[144] ImageMagick. *Convert, edit, and compose images* (2012). URL http://www.imagemagick.org/script/index.php. 127

[145] P. Baecher, M. Fischlin, L. Gordon, R. Langenberg, M. Luetzow, and D. Schrder. *Captchas: The good, the bad, and the ugly*. In *LNI 2010*, pp. 352–365 (2010). 137

[146] B. B. Zhu, J. Yan, Q. Li, C. Yang, J. Liu, N. Xu, M. Yi, and K. Cai. *Attacks and design of image recognition captchas*. In *17th ACM conference on Computer and communications security* (2010). 139

[147] C. Fritsch, M. Netter, A. Reisser, and G. Pernul. *Attacking image recognition captchas. a naive but effective approach*. LNCS **6264**, 13 (2010). URL http://epub.uni-regensburg.de/16872/1/trustbus_1.pdf.

[148] Y. Ke, R. Sukthankar, , and L. Huston. *Efficient near-duplicate detection and subimage retrieval*. In *ACM Multimedia* (2004).

[149] J. Li and J. Z. Wang. *Real-time computerized annotation of pictures*. In *IEEE Pattern Analysis and Machine Intelligence*, p. 9851002 (2008).

[150] A. W. M. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain. *Content-based image retrieval at the end of the early years*. In *IEEE Pattern Analysis and Machine Intelligence*, pp. 1349–1380 (2000). 139

[151] R. Datta, J. Li, and J. Z. Wang. *Imagination: A robust image-based captcha generation system*. In *ACM Multimedia* (2005). 139

[152] W. Zhang. *Zhang's captcha architecture based on intelligent interaction via ria*. In *Computer Engineering and Technology (ICCET)*, vol. 6, pp. 57–62 (2010). 142

[153] NeOT. *Coroutine*. URL http://www.nevaobject.com/. 149