2010 Second Cybercrime and Trustworthy Computing Workshop

Protecting Web 2.0 Services from Botnet Exploitations

Nguyen H Vo, Josef Pieprzyk

{*nguyenvo, josef* } @ *science.mq.edu.au*

Department of Computing, Macquarie University, Australia

Abstract—**Recently, botnet, a network of compromised computers, has been recognized as the biggest threat to the Internet. The bots in a botnet communicate with the botnet owner via a communication channel called Command and Control (C&C) channel. There are three main C&C channels: Internet Relay Chat (IRC), Peer-to-Peer (P2P) and web-based protocols. By exploiting the flexibility of the Web 2.0 technology, the web-based botnet has reached a new level of sophistication. In August 2009, such botnet was found on Twitter, one of the most popular Web 2.0 services. In this paper, we will describe a new type of botnet that uses Web 2.0 service as a C&C channel and a temporary storage for their stolen information. We will then propose a novel approach to thwart this type of attack. Our method applies a unique identifier of the computer, an encryption algorithm with session keys and a CAPTCHA verification.**

Index Terms—**botnet, Web 2.0, Trojan 2.0, API, MAC address, communication channel, CAPTCHA**

I. Introduction

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 [14]. Botnet has been recognized as the biggest threat to the Internet because of its criminal usages. These include Distributed Denial of Service (DDoS) attacks, spamming, traffic sniffing, key logging, identity theft and malware spreading [12]. As traditional IRC C&C methods are becoming quite easy to detect, the current trend is to use web-based and P2P models for C&Cs. In particular, web-based bots are predicted to grow due to Web 2.0 security weaknesses [12].

The new breed of botnets exploits Web 2.0 technologies such as RSS feeds, API (Application Programming Interface), blogs and mashup and have reached a new level of sophistication. For example, Trojan 2.0 uses a popular public web service such as Google or Yahoo as a C&C channel for communicating with botnets [6]. In August 2009, a botnet that used Twitter as a C&C channel was found [26]. The "Twitter" botnet uses the status messages to send out new links to contact, then these contain new commands or executables to download and run. The bots in the "Twitter" botnet use the RSS feed to get the status updates. 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 tools.

In this paper, we will present a method to prevent the botnet from using Web 2.0 service as a C&C channel for communication. Since a bot must use API to post information, we can verify if the user is a human being or a bot before executing the API calls. Our tool, API Verifier, will challenge the user with CAPTCHA if it detects that the API call is from a new computer. More precisely, when API is called, API Verifier will send the permanent media access control (MAC) address of ethernet/wireless card of the computer, which is globally unique, to the web service for verification. Should the verification fail, i.e, this MAC address does not match with the one associated with the username, the user will be requested to enter characters on a CAPTCHA image. Since the bot cannot solve a CAPTCHA challenge, the access attempt will not be verified. To avoid the MAC address being spoofed while being sent to web server, our tool also employs an encryption algorithm.

The paper is structured as follows. The next section will explain in detail botnets and their communication methods. In Section 3, we will describe an attack scenario, where a botnet uses a legitimate Web 2.0 service as a C&C channel to by-pass the current botnet detection tools. A summary of related works is presented in Section 4. We then propose an approach to detect the botnet 2.0 communication channel and introduce our tool, API Verifier, in Section 5. Section 6 is dedicated to a discussion about possible attacks and countermeasures. Conclusion and open problems are detailed in the final section.

II. Botnets and their Communication Topology

A botnet is a network of compromised computers, which are called bots (also called zombies or drones). The hacker controlling a botnet is called botherder or bot-master. It is also important to distinguish a computer that has become a bot (or botnet client) from a computer that has been broken in by a hacker. Schiller et al. [14] define a botnet as a collection of computers that meets the following criteria:

- the botherder is able to take action on the bots without having to log into the bot operating system.
- bots must be able to act in a coordinated fashion to accomplish a common goal (of the botnet) with little or no intervention from the botherder.

A typical botnet will consist 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 botherder issues a command/message to the bot server that futher communicates with the bots. The bots then execute the commands and report back the results to the IRC server.

Although IRC has been proven to be the best and dominant botnet communication protocol for many years [14], [13], it still has a weakness - it is centralized. With a simple structure as shown in Figure 1, every bot listens to and reports to a central location. If the IRC server that collects the information from the bots is detected, then by shutting down the server, the whole botnet is normally destroyed. In addition, sitting on an IRC channel, a message can be easily eavesdropped. That means commands from the botherder can be easily eavesdropped and even interfered as they transmitted via an IRC channel openly. The P2P botnet structure, on the other hand, is decentralized by creating a "buddy list" for each bot. In other words, a bot knows N other bots whose names are in its "buddy list". To issue a command to a P2P botnet, the botherder 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 buddy list and so on. Because of the lack of a centalized controller, P2P botnets are harder to track and shut down.

Recently, the third type of botnets has emerged. Unlike the previous botnets, the C&C channels are implemented using public web services. This type is called web-based botnets. Similar to the IRC botnet, a web-based botnet is centralized with a web server as a central command and control server (see Figure 1). Instead of sitting on an IRC/P2P channel and listening to the commands, the web-based botnet can exchange a single command with the server and close the connection. Because the connection established by web-based bots is not permanent, it is more difficult to detect and thwart this attack.

III. Botnet 2.0

Although Web 2.0 has improved our ability to freely communicate and share information, its great flexibility has also offered new C&C communication tools to potential attackers. One of the most significant threats is that attackers can use popular and publicly access Web 2.0 services such as Google or Yahoo blogs to communicate with her bots (see Figure 2) [6].

In particular, the attacker will post commands/updates to a public blog service and thanks to RSS, her bots will be alerted and get those commands/updates from the blog service. The bots then run the commands/updates and subsequently send responses or stolen information such as user's bank accounts or sensitive information to the blog service. 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 by-pass 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 of detection tools. This type of Trojan is named Trojan 2.0 by Finjan, a provider of secure web gateway solutions for the enterprise market [6]. In this paper, we call the botnets that use this communication method as Botnets 2.0.

IV. Related Works

Although botnets appeared almost 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 designing anti-spam software. New techniques apply a blend of various techniques such as honeypots and tracking techniques.

One of the most common methods for detecting bots is an analysis of network traffic flows. Some attempts have been made to examine traffic contents in order to find IRC botnet commands. Once found, the source of the traffic is identified and then shut down. Livadas et. al., proposed to use machine learning-based classification techniques to identify C&C traffic of IRC-based botnets [9]. Other approaches focused on an examination of traffic behavior such as bandwidth, duration of sessions and packing timing in order to identify botnets activity [9] [17]. However, these methods rely heavily on the historical data, which quickly gets obsolete if the data set is not updated. These approaches are also likely to produce a high rate of false positives [2].

Application of honeypots is a well known technique for discovering activities of attackers who own botnets. A honeypot is actually a bot that joins a botnet to spy and gather information about the botnet [11] [15]. Many researchers and security managers have been using honeynets, networks of more than one honeypots, to learn more about botnets and shut them down if possible. For example, the Honeynet Project [15] has done extensive work on capturing bots and characterizing botnet activities. This experience provided a valuable information to the research community. However, maintaining a honeynet system is expensive and is risky as the honeypot may harm other computers while trying to act like a real bot to avoid the bot-master's honeypot detection.

Another well known technique is network traceback, which attempts to determine the attacker's location by tracing the IP address of the bots. The IP tracing technique is trying to locate the source of the botnet packets by placing filters in a few key points in the network or probabilistically adding markings to IP packets to trace the source of the packets [5] [7]. In the connection chain traceback technique, the first step is to contact the previous host and ask the administrator to investigate her own systems. Then this process continues recursively across all hosts on the chain, which is an expensive investigation. There are only a few published works on connection chain tracing and none of the IP tracing techniques has been successfully implemented.

The above techniques, however, do not cover Botnet 2.0. Traffic analysis and honeypot techniques are unlikely to work for Botnet 2.0 as the bot activities are indistinguishable from typical http requests and responses generated by legitimate users and web sites. As long as the tracing techniques are concerned, they fail because they are going to find the address of an legitimate web site. Note that, in Botnet 2.0, the bot-master never communicates directly with their bots. To our best knowledge, there has not been published any work that addresses the protection against Botnets 2.0. In the next section, we will present a method to protect web server from Botnet 2.0 exploitation.

V. API Verifier

A. Motivation

To update information on Web 2.0 sites, a bot must use 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 to access 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 [1]). Therefore, we will challenge the user using a CAPTCHA image 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 for CAPTCHA verification when they use the same computer again.

How the above approach can prevent Botnet 2.0 from abusing web services? To set up a Botnet 2.0, the attacker needs to sign up for one or several accounts in a blog web service then he needs to write a program (bot) to automatically post blog entries to the blog web service from a victim computers. 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 attacker 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 “read” letters on the CAPTCHA image, it cannot pass the CAPTCHA verification. Hence, the API request is denied and as a result, the communication channel between bots and the attacker 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 paper, 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 6.

B. Concept of API Verifier

B.1 API Verifier Work Flow

The API Verifier consists of 2 components: an API Verifier Client and an API Verifier 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. 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 API Verifier Client sends a request to the web server and gets a time token in return. The API Verifier 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 and sends it to the API Verifier Server for verification. The API Verifier Server decrypts the message to get the MAC address and compares it with the existing MAC address associated with the user. If they match, it means that, the user uses the same computer as before and the verification is passed. Otherwise, the user works from this computer for the first time. Consequently, the API Verifier will present a CAPTCHA image to the user for verification. If the user is a human being, they will be able to enter the CAPTCHA string 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 be asked to enter CAPTCHA again. 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 server. See Figure 3 for details.

B.2 API Verifier Client

As mentioned above, the API Verifier is a program running at the client side. The API Verifier Client is also a library that a user needs to import into their code to be able to use the Web 2.0 service’s API. It has two main classes: a Server Interaction and a Client Information (see Figure 4)

1. *Client Information*: gets permanent MAC address from the Operating System, encrypts the MAC Address 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 strings entered by

users. The Sever Interaction also communicates with the Client Information to pass tokens to and receive encrypted MAC addresses from the Client Information.

B.3 API Verifier Server

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

B.4 Authentication

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

B.5 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.

B.6 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 [4]. By default, the current MAC address is the same as the permanent one. However, the current MAC address is easy to be changed (see [19] 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 tool, we only use the permanent MAC address. The reasons for that will be discussed in Section 6. 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 [20].

B.7 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 API Verifier Server will generate a random string, put it to a CAPTCHA image and send the image to the client side. The API Verifier Client will display the CAPTCHA image and get the CAPTCHA string entered by the user then send it back to the server for verification. The CAPTCHA image must be hard to solve by a computer program but readable by a human being. A user has three attempts of solving CAPTCHA within a given amount of time (which can be adjusted in the server).

B.8 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 many 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 is 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 API Verifier 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.

C. Implementation

A simple API Verifier has been developed for evaluation. The tool is implemented using Java and J2EE and is compatible with Windows platforms (XP and Vista were tested). The encryption algorithm we use is AES (Advanced Encryption Standard) and the length of the key is 128 bits. For simplicity, some properties files are used instead of a database for storing user's profiles. To get the permanent MAC address, it requires a very low level of code to interact with the device. We use a licensed library called Coroutine for getting the permanent MAC address

[23]. To ensure that a CAPTCHA image is hard to solve by a computer program, we distort the characters of the CAPTCHA string on the image and make segmentation difficult by introducing angle lines on the string.

VI. Security Analysis

In this section, we will discuss about possible attacks against the API Verifier, its limitations and possible ways to address these issues. There are three main attacks as shown in Figure 5.

1. Spoofing MAC Address: this attack can be done in two places: (a): changing the MAC address of the computer before the API Verifier Client gets the MAC address from the OS. (b): changing the encrypted MAC address when it is being sent from the API Verifier Client to the server.
2. API Verifier Client Fraud: the attacker may use his own program to imitate the API Verifier Client functionalities and make API calls to the server.
3. Distributed Denial of Service attack: the attacker may use the API Verifier Client to send requests to the server from a huge number of bots.

A. *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 by-passing a CAPTCHA challenge. As mentioned above, the MAC address can be changed in two places.

A.1 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 operating system (OS) registry. The permanent MAC address is the code burned in EEPROM of the device and cannot be changed. Therefore, the API Verifier 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 [24]. For bots, the less activities, the better. They silently reside on victim computers, steal data and wait for commands from the botherther. Therefore, bots rarely hijack the OS kernel because of the high risk of being detected.

Because the main scope of this paper is to address the communication of bots via web 2.0 service, another research paper 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 if the kernel is compromised by a bot and cannot return a trusted permanent MAC address. In this paper, we assume that OS kernel is not taken over by the bot and returns the correct permanent MAC address.

A.2 Changing the encrypted MAC address

In this scenario, the bot will replace the MAC address encrypted by the API Verifier 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 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 it 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, her bots will pass the MAC address verification too. As a result, her bots are able to post information to the web service.

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. Consequently, the attacker will not be able to spoof the MAC address using this method.

B. *Protection of API Verifier Client and the secret key*

The second possible attack is to replace the API Verifier 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 send 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 API Verifier 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 API Verifier Client
2. it is hard or impossible to disassemble the API Verifier 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 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 API Verifier Client. Obfuscation is a technique of making a content very hard to read and understand. Although obfuscation is not provable secure, in practice, it makes the reverse engineering task very costly in terms of time and effort [10]. 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 [8]. Therefore, a good choice of coding language and an obfuscation tool can improve the protection of API Verifier.

C. By-passing the CAPTCHA Verification

Another attack is to by-pass the CAPTCHA verification by using one of the following techniques:

(a) targeting victims that have already passed CAPTCHA verification, then using the victim accounts rather than the attacker account. Because the user have already passed the CAPTCHA verification, when the bot calls API using the same account to post stolen information, it will not be asked for solving a 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 in their blog.

(b) solving the CAPTCHA by analysing the picture and extracting characters on the image. According to Mori and Malik [25], their approach of solving CAPTCHA image can identify characters on a CAPTCHA image with high rate of accuracy. However, we can always make a CAPTCHA harder to be analysed such as distorting the characters of the CAPTCHA string on the image and making segmentation difficult by introducing angle lines on the string as used in our implementation. We can even make it 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 looking to gain access.

(c) sending the CAPTCHA image to someone (human-being) to solve the CAPTCHA challenge for the bots. If the bot send the CATPCHA to the attacker, 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 bot-master; otherwise, her identity can be traced back. In addition, a user only has a given amount of 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.

Another way is that the bot could trick the user into solving the CAPTCHA by posing it when the user web surfs to any site or sending the CATPCHA image to other users in the botnet with hope that one of them is surfing a website and solves the challenge for them. To avoid this attack, we can display a prompt to the user on the CAPTCHA image. For example, the image will not only contain characters to enter but also a clear message like "Please enter below characters to gain access to automatic updating information to XYZ web service. If you are not using this service, you should not enter any character" or something like that. We also give the bot a hard time by using random size for the image and making the prompt hard to be extracted from the image. Because the time for solving CATCHA is short, this type of attack is unlikely to be launched successfully. In addition, we can detect if the bot sends the image to many users in the botnet by monitoring the network traffic at that time. If after the CAPTCHA image is displayed, a high load of sending package is detected, the API Verifier will generate and display another CAPTCHA image. After three attempts, the API Verifier will terminate and only allow another attempt in a certain amount of time. Please 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.

D. Distributed Denial of Service attack

Since the MAC address and CAPTCHA verifications would require adequate resources for generating time token/CAPTCHA, decrypting and verifying, an attacker may request the MAC address and CAPTCHA verifications repeatedly from a large number of bots to overload the web service (DDoS attack). To prevent this, an API Verifier will support an option, which allows us to set limit on the number of verification attempts. For example, we may restrict a user to solve the CAPTCHA challenges for no more than three times. If they all fail, the user will need to wait for a certain amount of time or the account will be blocked. For the MAC address verification, the API Verifier will only allow one API call per user at a time. Because many bots usually share the same account, this will prevent the attacker from using thousands of bots to make simultaneous requests to the web server.

E. Multiple MAC addresses

Sometimes, users may wish to access several computers to automatically post entries to the web service. 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 will not increase the chance for the attacker to succeed and maintain a Botnet 2.0. Although one account can be shared by a number of bots, as a botnet usually consists of thousands of bots, it still requires thousands of accounts and CAPTCHA verifications that the attacker cannot afford.

F. Virtual Machine

One shortcoming of the API Verifier is that it cannot get the permanent MAC address of a computer when the API is called from a virtual machine running on the computer. Therefore, we currently do not allow a user to use virtual machine to automatically post entries to Web 2.0 services. If the API Verifier cannot get the permanent MAC address because the user is running a virtual machine, an error message will be returned to inform the user and ask them not to use virtual machine to make API calls to the web server. Although this is a non-trivial limitation, it will not have a significant effect on the users as a majority of them do not use virtual machine for posting blogs or information to websites.

VII. Conclusion and Future Works

In this paper, we have described a new type of botnet, Botnet 2.0, which attempts to exploit the flexibility of Web 2.0 to maintain and expand the botnet. We have proposed and implemented 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. Our implementation shows that the API Verifier can successfully detect if a user is a bot before allowing them accessing to API. If the user is a bot, the API call is denied and as the result, the bot cannot post any information to the web services. Consequently, the communication between the bots and the bot-masters cannot be established. The API Verifier is also carefully designed so that it is immune against all possible attacks including spoofing MAC address and DDoS attacks. The API Verifier is also protected against reverse engineering. However, there is one weakness that the current version of API Verifier cannot get the permanent MAC address of a computer if it is running a virtual machine.

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. The key of the approach is to base on a unique and permanent identifier to identify a computer and protect this identifier from being spoofed. In this paper, we use the permanent MAC address as the identifier of the computer. In future research, we will look at other alternatives such as Universally Unique Identifier (UUID) [16] or Trusted Platform Module [22] which may offer different ways to identify the computer as well as address the virtual machine limitation. TPM may also offer a more secure way to protect the unique identifier from being spoofed.

Because of research purpose, we have only implemented API Verifier compatible with Windows XP and Vista platforms which are run by majority of personal computers. Other platforms such as Linux, MAC or mobile phone OS will require different techniques to get permanent MAC address as well as to display CAPTCHA properly. We have not either implemented the monitoring network traffic function to against by-passing CAPTCHA verification attack as mentioned in section 6.3. If API Verifier goes to production, the above challenges must be taken in to account.

References

[1] Addison, D., 2006. *Web Site Cookbook: Solutions & Examples for Building and Administering Your Web Site.* O'Reilly, 2006.

[2] Akiyama, M., Kawamoto, T., Shimamura, M., Yokoyama, T., Kadobayashi, Y., Yamaguchi, S., 2007. A Proposal of Metrics for Botnet Detection Based on Its Cooperative Behavior. Proceedings of the 2007 International Symposium on Applications and the Internet Workshops. Jan. 2007 p 82 - 85.

[3] Bacher, P., Holz, T., Kotter, M., Wicherski, G., 2005. Know your Enemy: Tracking Botnets. The Honeynet Project and Research Alliance. Available at: http://www.honeynet.org/papers/bots/ last access 30th September, 2007.

[4] Bradford E., Mauget L., 2002. *Linux and Windows: A Guide to Interoperability.* Prentice Hall PTR, 2002.

[5] Carrier, B., Shields, C., 2004. The Session Token Protocol for Forensics and Traceback. ACM Transactions on Information and System Security (TISSEC), volume 7 , Issue 3 (August 2004), p. 333 - 362.

[6] Finjan Web Security Trends Report Q4 2007.

[7] Jin, Y., Zhang, Z., Xu, K., 2007. Identifying and Tracking Suspicious Activities through IP Gray Space Analysis. Proceedings of the 3rd annual ACM workshop on Mining network data MineNet '07. California, USA 2007.

[8] Linn, C., Debray, S., 2003. Obfuscation of executable code to improve resistance to static disassembly. CCS03, October 2731, 2003, Washington, DC, USA.

[9] Livadas, C., Walsh, R., Lapsley, D., Strayer, W. T., 2006. Using Machine Learning Techniques to Identify Botnet Traffic. Proceeding 31st IEEE conference on Local Computer Networks, p. 967-974. Florida, USA, 2006.

[10] Low, D., 1998. Protecting Java Code via Code Obfuscation. ACM.

[11] McCarty, B., 2003. Automated identity theft. IEEE Security & Privacy Magazine, volume 1, issue 5, Sept.-Oct. 2003 p.89 - 92.

[12] *Online eWEEK Magazine* `http://www.eweek.com/`

[13] Symantec Internet Security Threat Report: Trends for January-June 07. Volume XII, Published September 2007. Available at `http://eval.symantec.com/mktginfo/enterprise/white_papers/ent-whitepaper_internet_security_threat_report_xii_09_2007.en-us.pdf`. Last access on 1st October, 2007.

[14] Schiller, C.A., Binkley, J., Harley, D., Evron, G., Bradley, T., Willems, C., Cross, M., 2007. *Botnets: The Killer Web App.* Syngress.

[15] Vrable, M., Ma, J., Chen, J., Moore, D., Vandekieft, E., Snoeren, A.C., Voelker, G.M., Savage, S., 2005. Scalability, fidelity, and containment in the potemkin virtual honeyfarm. Proceedings of the twentieth ACM symposium on Operating systems principles, p. 148-162.

[16] Schill, A., 1996. Distributed Platforms: Proceedings of the IFIP/IEEE International Conference on Distributed Platforms: Client/server and Beyond: DCE, CORBA, ODP and Advanced

Distributed Applications. International Federation for Information Processing. Chapman & Hall, 1996

[17] Strayer, W.T., Walsh, R., Livadas, C., Lapsley, D., 2006. Detecting Botnets with Tight Command and Control. Proceedings 2006 31st IEEE Conference on Local Computer Networks, p. 195 - 202. Tampa, Florida, 2006.

[18] BotHunter `http://www.bothunter.net/`. Last access on 5th April, 2009.

[19] `http://www.nthelp.com/NT6/change_mac_w2k.htm`. Last access on 10th December, 2008.

[20] `http://msdn.microsoft.com/en-us/library/bb314143.aspx`. Last access on 10th December, 2008.

[21] `http://code.google.com/apis/blogger/docs/2.0/developers_guide_protocol.html`. Last access on 10th December, 2008.

[22] `https://www.trustedcomputinggroup.org/groups/tpm/`. Last access on 10th December, 2008.

[23] `http://www.nevaobject.com/`. Last access on 10th December, 2008.

[24] `http://www.cisco.com/web/about/security/intelligence/virus-worm-diffs.html`. Last access on 5th April, 2009.

[25] `http://www.cs.sfu.ca/ mori/research/gimpy/`. Last access on 5th June, 2009

[26] `http://asert.arbornetworks.com/2009/08/twitter-based-botnet-command-channel`. Last access on 4th Septemper, 2009

VIII. Figures

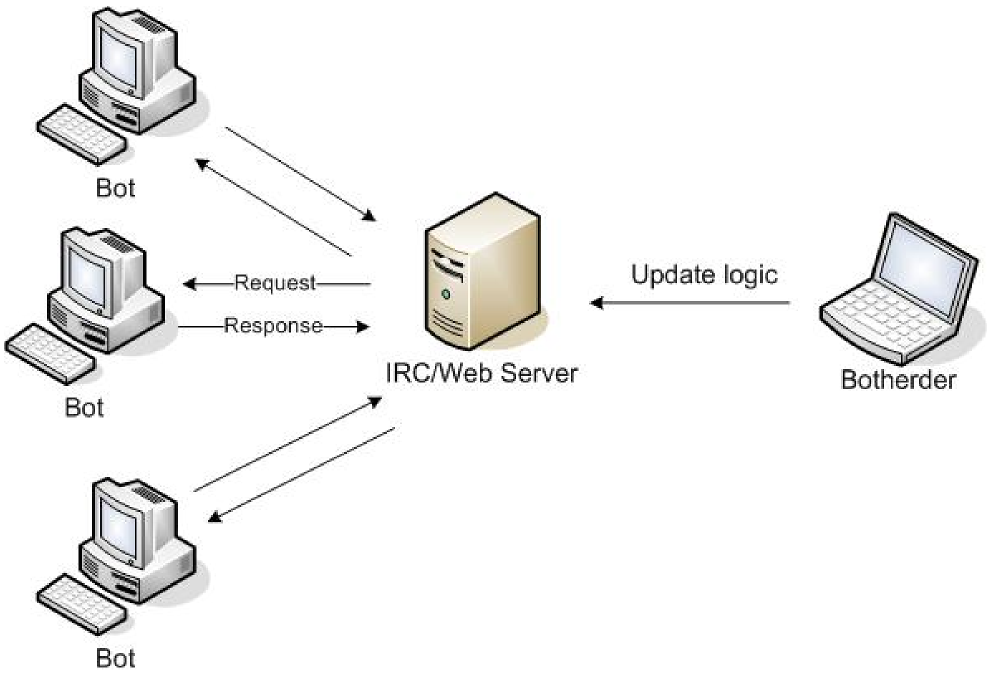

Fig. 1. Topology of a typical botnet

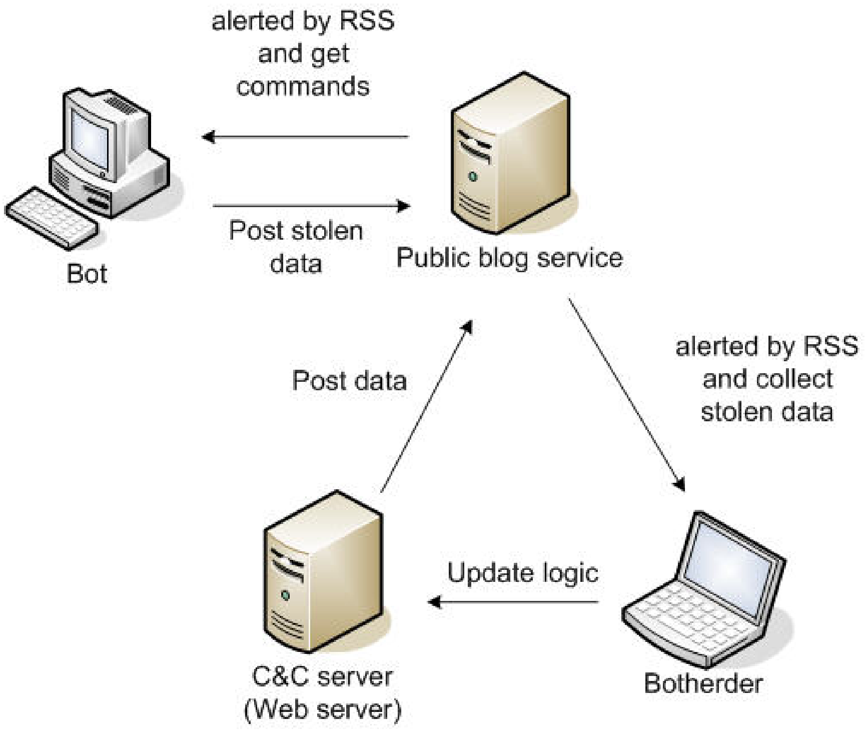

Fig. 2. Trojan 2.0 Command and Control

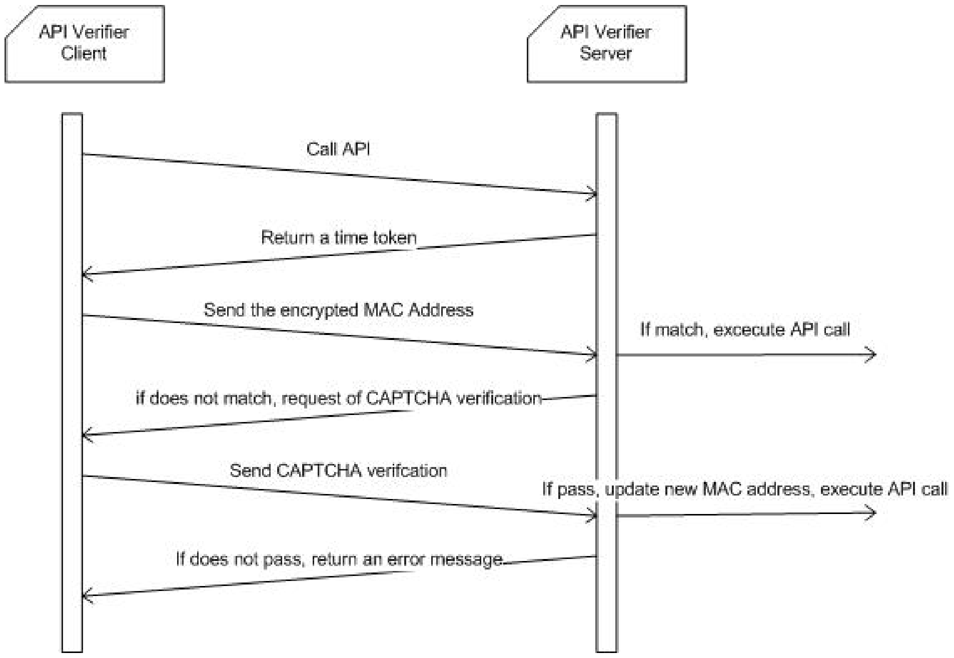
API Verifier Client
API Verifier Server
Call API
Return a time token
Send the encrypted MAC Address
If match, excecute API call
if does not match, request of CAPTCHA verification
Send CAPTCHA verifcation
If pass, update new MAC address, execute API call
If does not pass, return an error message

Fig. 3. The API Verifier Workflow

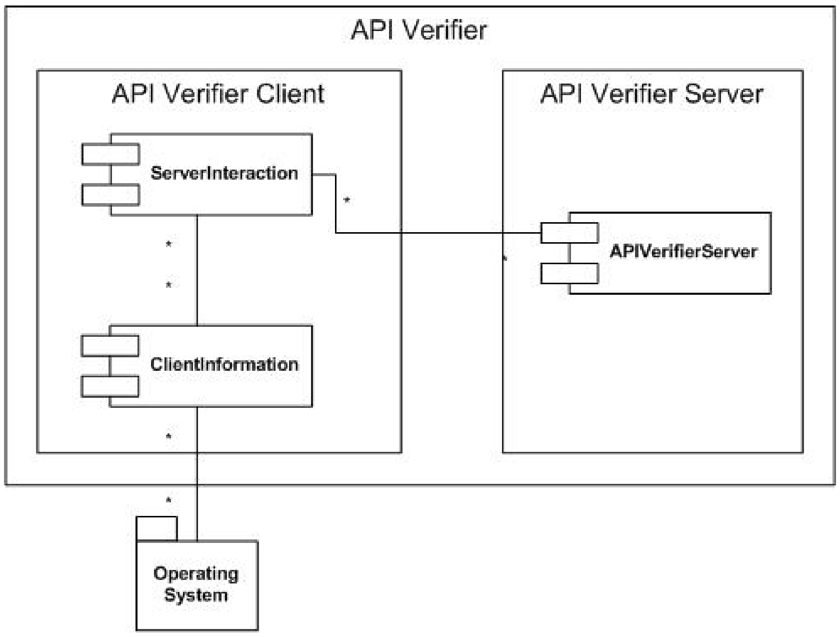

Fig. 4. Components of API Verifier

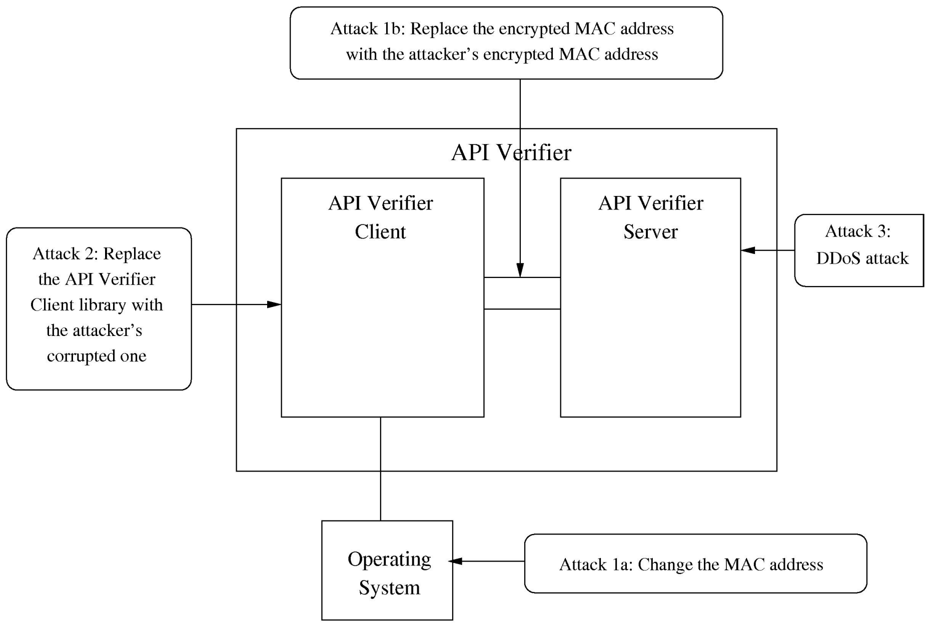
Attack 1b: Replace the encrypted MAC address with the attacker's encrypted MAC address
API Verifier
API Verifier Client
API Verifier Server
Attack 2: Replace the API Verifier Client library with the attacker's corrupted one
Attack 3: DDoS attack
Operating System
Attack 1a: Change the MAC address

Fig. 5. Model of Possible Attacks