Botnet with Browser Extensions

Lei Liu[1], Xinwen Zhang[2], and Songqing Chen[1]

[1] Dept. of Computer Science
George Mason University
Fairfax, VA 22030
{lliu3, sqchen}@cs.gmu.edu

[2] Computer Security Division
[2] Huawei Research Center
Santa Clara, CA 95050
xinwen.zhang@huawei.com

***Abstract*—Botnets are responsible for many large scale organized Internet attacks today. Along with the fight between botnet developers and defenders, the battle field has significantly evolved from traditional centralized IRC to various new approaches, aiming to make bots and command and control channel more and more stealthy. In this work, through prototype implementations, we demonstrate that browser extensions are a very effective botnet vehicle with very large installation base and the capability of accessing rich sensitive user data in the browser. The automatic update mechanism of browser extensions further offers a stealthy command and control channel between bots and a botmaster. Compared to many others, extension-based bots are more stealthy and harder to defeat since all mainstream browser architectures provide rich APIs for browser extensions to enrich users' browsing experience with insufficient consideration of malicious extensions. Via both an IE add-on and a Chrome extension, we show that attacks like email spamming, DDoS, and password sniffing are trivially feasible. Our study shows that an effective scheme is imperatively demanded to mitigate such threats.**

I. Introduction

Botnets are one of the biggest threats to Internet security today. They are responsible for a majority of large scale organized Internet attacks, such as DDoS and spamming, credit card number and password harvesting [1].

Typically, a botnet consists of three key elements: a botmaster or botmasters, hundreds to thousands of bots (compromised computers), and a command and control channel via which the botmaster controls the bots. Therefore, the battle between botnet operators and defenders focuses on these aspects: a botmaster tries its best to hide itself from being identified by using stealthy command and control channels, and a bot tries to hide from being detected by host-side anti-malware systems and obtain as many privileges as possible.

In early days, a botmaster often communicates with controlled bots through a centralized IRC server. This enables a botmaster to hide its communication with bots in normal users' legitimate traffic. Therefore, it is natural to monitor TCP port 6667 which is for IRC traffic [2] in order to detect command and control information. Other solutions include building IRC server scanners to detect potential botnets by identifying non-human behavior characteristics in traffic [3], [4].

Driven by profit, botnet developers are constantly exploring new mechanisms for command and control channels and developing more stealthy bots that can easily infect large scale hosts. For example, with the scrutiny of the IRC channels, bots have been found to bind to other commonly used applications, such as a Web browser [5], and to use other protocols, such as HTTP [6]. Random delay can be added to command propagation among bots in order to avoid detection [7]. Since the centralized control through a IRC server could be easily detected and shut down, P2P based botnets have also been developed [8], [9], [10], [11], [12]. For example, Overbot [12] has demonstrated that P2P bots can be built on Kademlia-based P2P networks.

In this work, we show another form of bots and command and control channels: browser extensions and their automatic update mechanisms implemented in mainstream browsers. Although the concept of developing or hiding bot in a browser extension has been mentioned recently [13], [14], to our best knowledge, we are the first to implement this botnet model and demonstrate its feasibility under the latest mainstream browser extension architectures. Via implementation, we show that potentially a malware developer can lure a user to install a malicious extension, which, under both Internet Explorer (IE) and Chrome's extension architectures, can easily steal user sensitive online data such as username/password, and send out to external servers. Furthermore, the extension can easily file cross-site HTTP requests to send spam emails and launch DDoS attacks. More critically, by leveraging the built-in automatic update mechanism implemented in IE, Mozilla, and Chrome, a bot in the form of a browser extension can easily obtain command and control messages from a botmaster without triggering any anti-virus software.

Several advances of browser development have motivated and enabled this new form of botnet. First of all, web browsers nowadays have become a major vehicle for common people to surf the Internet and consume web services, including e-commerce and online banking services. Therefore, plenty of sensitive information becomes the target for bots to harvest.

Second, there exist a large number of browser extensions and plugins to enrich users' browsing experience. For example, they can help browsers process different types of media contents, or automate user actions such as filling forms or remembering password. For this purpose, browsers provide lots of APIs for extension developers to access the core browser resources and web pages, including DOM, cookies, browsing history, bookmark, toolbar, etc. This opens a large attack window for malicious extensions and plugins.

Third, through our implementation, we find that although mainstream browsers have some built-in security mechanisms for securing extensions, they are far from sufficient to confine the behavior of malicious extensions. In particular, in IE, browser extensions such as menu extensions, custom toolbars, explorer bars, and Browser Helper Objects (BHOs) share the same process space of the browser and thus can perform any action on the available windows and modules. A BHO even can modify the functionality of the browser by adding binary components. Therefore, it is very difficult to restrict the behavior of a malicious extension in IE without restricting the whole browser process's capability, e.g., running IE in protected mode [15]. Chrome adopts a multi-process architecture in which extensions run in separated processes from the main browser process. Chrome further defines individual permissions that can be assigned to extensions, such as the permissions to inject JavaScript into web pages, making cross-site access requests, accessing tab and window modules, cookies, local storage, etc. However, the design of Chrome extension security aims to prevent malicious web pages from leveraging extensions to obtain these permissions, and the coarse-grained permission management cannot prevent malicious extensions from accessing sensitive data in web pages and browser modules and making cross-site HTTP requests.

Last and most importantly, the extension update mechanism implemented in mainstream browsers provides a very stealthy command and control channel for extension-based botnets. Specifically, in Chrome and Mozilla, each browser extension includes an `update_url` in its metadata, with which the browser uses to check updates, e.g., each time when the browser starts [16], [17]. Therefore, a botmaster can easily propagate attack command and victim information to a large number of bots without being detected by anti-virus software. Such an approach could be used to distributed bot binary as well. In IE, a malicious add-on even can implement its own update mechanism and download arbitrary code and data. Compared to other approaches, a malware developer can simply develop popular extensions with bot functions hidden. In this case, all browsers with such extensions installed become bots, and a botmaster can communicate with them with ease.

With implemented malicious extensions on both IE and Chrome, we demonstrate three types of bot attacks, including email spamming, DDoS, and password sniffing. For email spamming, instead of sending spam emails in a burst fashion, we implement silent and sporadic email spamming that are harder to be detected. For DDoS, we launch DDoS attacks in the IE add-on and the Chrome extension with the same command file. For password sniffing, we demonstrate it is easy to sniff login password in e-transactions with different banks. Through these experiments, we show that such attacks could be practicably mounted with trivial efforts. Our study shows that we are in great need of some effective scheme to defeat such threats.

The rest of the paper is organized as follows. We discuss the security model for these attacks in section II. We present email spamming, DDoS, and password sniffing attacks in section III. We discuss research efforts to counter these attacks in section IV and make concluding remarks in section V.

II. Security Model

There are many different ways to distribute browser extension-based bots. As aforementioned, a botmaster can develop popular extensions with normal functions that can be discovered and downloaded by users. Although browser development communities encourage users to download extensions from trusted sources, e.g., in Chrome and Firefox extension galleries, it's usually very difficult to evaluate if an extension is benign [18], partially due to the large number of extensions from third-party developers and the very dynamic behavior of the programming languages used in extensions, typically JavaScript and HTML. On the other hand, there is no effective mechanism yet to prevent a user from installing extensions downloaded from other sources, e.g., embedded links from spam emails or phishing web pages. The recent Trojan posed as a fake Chrome extension suggests that such threats are not fictitious [19].

Alternatively, a bot extension does not need to have malicious code and functions when it is firstly installed, e.g., to escape from offline code analysis tools [18]. After the installation base reaches a certain level, the developer or the extension owner can leverage the stealthy update mechanism to include malicious code and data into the extension, hence tuning it to a bot whenever needed.

Therefore, in this study, we do not make a specific assumption for distributing and installing bot extensions. Instead, we evaluate how an extension can harvest sensitive information in a browser environment, and obtain network access capability to launch botnet attacks. We further explore how a botmaster leverages the automatic extension update mechanism for command and control channel. We implement bot extensions on both IE and Chrome, which represent two mainstream browser architectures with different extension security models.

III. Attack Cases

In this section, we present the details of three attacks, namely, emails spamming, DDoD, and password sniffing, which are implemented through a Microsoft Internet Explorer (IE) add-on and a Chrome extension.

A. IE Add-on and Chrome Extension

IE is the most popular browser. An IE add-on runs in the same process space and has the same privileges as the IE browser. As a result, it can access resources like all other native applications without any restriction. More specifically, an IE BHO has the privilege to 1) access Internet, 2) access disks, 3) access browser resources such as cookies and bookmarks, and 4) access web page objects such as all DOM elements. IE add-ons can also implement their own update mechanisms in an ad-hoc manner as they can access local filesystems and network resources freely.

To demonstrate the attack via IE add-ons, we have implemented a BHO named `HDV` for IE8 on Windows 7 with a hardcoded update URL. This BHO is claimed to help process video files and it periodically checks an update server via the URL. It downloads the update files whenever they are available. The hidden bot thus can receive commands distributed by the botmaster with the update server we have set.

Chrome is relatively new. It uses a multi-process architecture, where a single browser kernel process runs in the privileged mode to access platform and system resources, on behalf of multiple renderer processes. A renderer process runs in a sandboxed environment so it cannot directly access system resources such as the filesystem and the network. It can only send such requests to the browser kernel process.

A Chrome extension usually includes an extension core and one or more content scripts. A content script is written in JavaScript that can be injected into a web page when the page is loaded. It then runs in the renderer process space to access the DOM tree. The extension core includes one or more background web pages written in HTML and JavaScript, and runs in a separate renderer process. A content script has the least privileges so it cannot access any object out of its renderer process space and has to communicate with the extension core via Chrome's inter-process communication (IPC). While the extension core contains the bulk of the extension privileges, it runs in a sandboxed environment. Therefore, it cannot access resources of the host platform and the network directly, and can only communicate with external web resources via `XMLHttpRequest`. Although binary code can be included in a Chrome extension, it can be run in a sandboxed plugin process; therefore we focus on JavaScript and HTML based extensions only in our study.

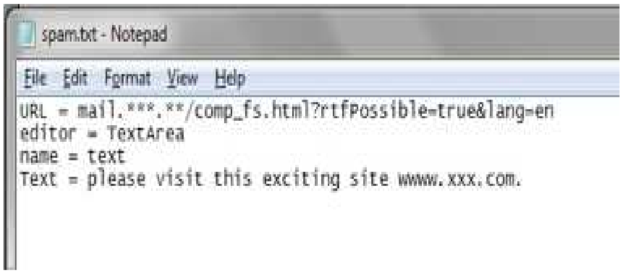
Different from IE, Chrome defines a set of of permissions for extensions, such as the permission to inject JavaScript into web pages, make cross-site access requests, and access tab and window modules, cookies, local storage. The desired permissions of an extension are specified in a `manifest.json` file by its developer, and prompted to the user when it is installed. The design of Chrome extension architecture is based on the assumption that extensions are benign-but-buggy; that is, the goal of the security architecture is to protect extensions from being exploited by malicious web pages and control the potential damage done to the browser kernel process if an extension is exploited.

For Chrome, we again develop an extension, which is claimed for video processing but includes bot functions. Next we demonstrate that even with very normal permission specifications, a malicious Chrome extension can conduct typical botnet attacks.

B. Email Spamming

Today botnets are notoriously responsible for most of spam emails on the Internet [1]. A spammer controls or rents a botnet and sends spamming commands to bots. After receiving spamming commands, bots send spam emails to victims. To defeat various malware detection mechanisms, a bot, instead of keeping sending spam emails at a high rate, can be instructed to send spam emails only sporadically, which makes detection more difficult. Our implementation below works in such a way.

First, we present our implementation with an Microsoft Internet Explorer (IE) add-on. Because an IE add-on has full privileges to access the DOM elements of a web page, we leverage this feature for spamming attacks. When a user is accessing a web email system, the user often composes email content in an edit area and sends out when editing is done. In this case, the email content is saved in the DOM element. Thus, for spamming attacks, right before the email is actually sent out, `HDV` can modify the DOM element by injecting some spam content to the email content. When the victim opens the tampered email, she will read the embedded spam content. In this implementation, we experiment with the popular iPlanet email system [20].

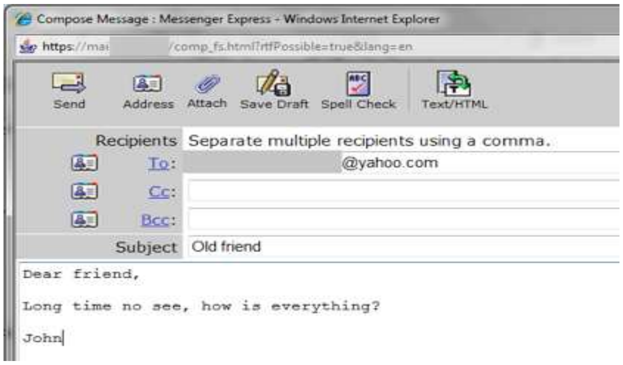


```
URL = mail.***.**/comp_fs.html?rtfPossible=true&lang=en
editor = TextArea
name = text
Text = please visit this exciting site www.xxx.com.
```

Fig. 1. IE BHO for Email Spam: Step 1 – Prepare the spam content

Figure 1 shows the update file we prepare for the update for `HDV`. To inject the spam content, the BHO only needs to tamper with the DOM element that stores the email content at a proper time. In our implementation, the email content is saved in a `TextArea` element with name `text`. Thus, the attack steps are as follows.

- First, the BHO retrieves the update file and reads spam content and a specific web email URL.
- Second, the BHO waits for the given URL to be accessed and listens to various browser events to monitor user behaviors.
- Third, when the user sends out the email, which is usually triggered by a click event, the BHO accesses the DOM element with name `text` and appends the spam content at the end of email.


Compose Message : Messenger Express - Windows Internet Explorer
Send
Address
Attach
Save Draft
Spell Check
Text/HTML

https://ma[redacted]/comp_fs.html?rtfPossible=true&lang=en

Recipients: Separate multiple recipients using a comma.

To: [redacted]@yahoo.com

Cc:

Bcc:

Subject: Old friend

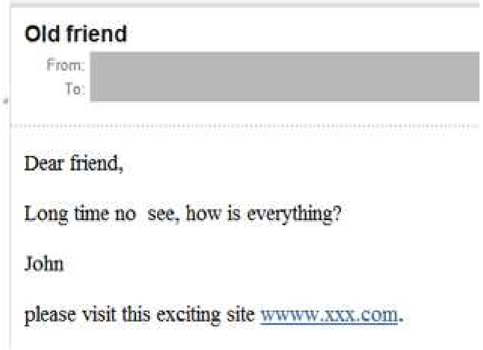
```
Dear friend,

Long time no see, how is everything?

John
```

Fig. 2. IE BHO for Email Spam: Step 2 – User sends out the email

Figure 2 shows the snapshot when the user finishes editing and is ready to send out the email. When a user clicks a button to send out an email, in IE, there is a click event with a button as the source and sometimes it can further lead to a form submission event. For the spamming purpose, HDV needs to capture these events in order to tamper with spam content. To capture the email sending action, HDV does the following in our implementation.

- It keeps listening to DISPID_DOCUMENTCOMPLETE event; when the document loading is complete, it walks through all DOM elements recursively and registers form events.
- When it captures a form event DISPID_HTMLFORMELEMENTEVENTS2_ONSUBMIT or a button event DISPID_HTMLELEMENTEVENTS2_ ONCLICK, it searches for the TextArea element with name text and appends the spam information to the email content.

In our implementation, in order to precisely capture email sending moment, we have instructed HDV to listen to both click and submission events and carefully analyze the event source. Different web email systems may have different implementation because they could use different components for editing and different approaches to send email. Thus other events may need to be monitored in other systems.



Old friend

From: [redacted]

To:

Dear friend,

Long time no see, how is everything?

John

please visit this exciting site wwww.xxx.com.

Fig. 3. IE BHO for Email Spam: Step 3 – The actual received email

Figure 3 shows the actual email received. In this attack, the spam content is appended to a legitimate email, and thus it is difficult to filter such an email even the embedded link is known to be malicious. On the other hand, it is not easy to automatically split the email content into two parts and remove the spam part eventually. Once this approach is widely taken, we believe that existing email spam filtering systems need enhancements to prevent it.

======

Having presented the implementation on the IE add-on, now we discuss how this is implemented in Chrome focusing on the implementation difference between IE and Chrome. A Chrome browser checks the update information of an extension from embedded update_url every a few hours. If an update is available, the browser downloads the update and updates the extension. Similar to IE, we utilize this mechanism to distribute bot commands.

To send out spam emails, the extension has the following permission specification:

```
"permissions": [
   "tabs", "http://*/*", "https://*/*"
]
```

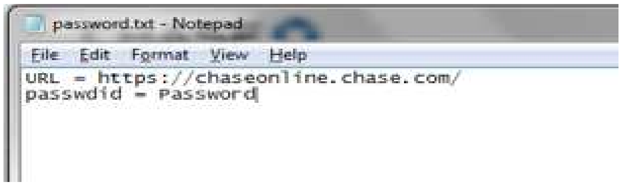
The http://*/* and https://*/* permissions are very common in popular extensions. These permissions enable the extension to send HTTP requests to all destinations.

With such permissions, we store the spam information in a file called spam.txt under the extension directory on our update server. Every time when the extension is activated, it will check this file and then obtain spam information including victims' email addresses and spam content. This approach can evade detection more effectively, because spam emails are sent out when the user logins into her web email system. As the extension is granted the privilege of "tabs", it listens to the update notification of the tabs with the method of chrome.tabs.onUpdated.addListener(). When the user logins into a web email system, the credential is represented by the session id (sid) and rewritten to the URL of the subsequent HTTP requests. As the bot extension has the tab permission, it can listen to the tab update notice. With this credential information, an HTTP request to the iPlanet email server is authorized to take any action on behalf of the user, instead of sending the user name and password in each transaction.

C. Password Sniffing

Nowadays, many Internet surfers use web browsers to do online shopping and access online bank accounts and financial services. Sensitive information such as bank account and password in these transactions is often saved by the web browser, temporarily or permanently, which makes web browsers a major target of spyware. Recent research has shown that many spyware are in the form of malicious browser plugins [21]. In our experiment, the example attack is against chaseonline.chase.com.

An IE add-on runs in the same memory space as the browser, and it can directly read all web page DOM elements. Therefore, one can leverage this for password sniffing trivially.

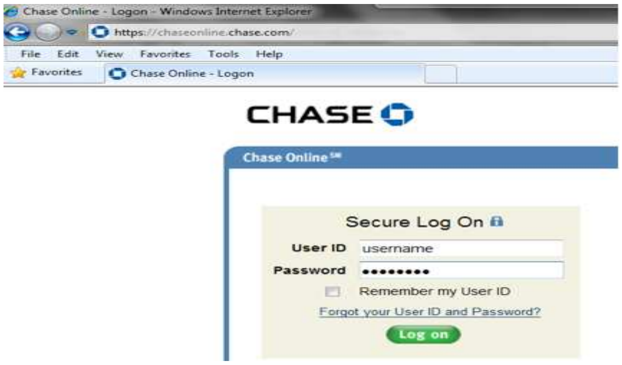


```
URL = https://chaseonline.chase.com/
passwdid = Password
```

Fig. 4. IE BHO for Password Sniffing: Step1 – the command

Figure 4 shows the update file we prepare for HDV on the update server, which specifies information about when to steal the password: upon the access of a particular URL.

When the BHO detects that the web browser is accessing the URL https://chaseonline.chase.com/ shown in Figure 5, it will read the password DOM element from the web page in the method of


Chase Online - Logon - Windows Internet Explorer
https://chaseonline.chase.com/
File Edit View Favorites Tools Help
Favorites
Chase Online - Logon
CHASE
Chase Online SM
Log on

Secure Log On

User ID username

Password ••••••••

□ Remember my User ID

Forgot your User ID and Password?

Fig. 5. IE BHO for Password Sniffing: Step2 – Password sniffing in progress

`OnDocumentComplete(IDispatch *pDisp, VARIANT *pvarURL)` as follows:

```
CComQIPtr<IWebBrowser2> spTempWebBrowser = pDisp;
...
CComPtr<IDispatch> spDispDoc;
hr = m_spWebBrowser->get_Document(&spDispDoc);
...
CComPtr<IHTMLElementCollection> spAll;
hr = pDocument->get_all(&spAll);
hr = spAll->item(svarItemIndex,
        svarEmpty, &spdispAll);
CComQIPtr<IHTMLElement> spElement = spdispAll;
if (hr == S_OK && pwId == "Password")
{
        BSTR innerText;
        spElement->get_innerText(&innerText);
}
```

After reading the password, HDV can save it to local disk or send it out with any networking protocol. Usually the password sniffing attack only starts when the user finishes all input and submits data to a server. In most cases a form is used to input and submit the account information. Thus the BHO can use a similar mechanism as that in the spamming attack to monitor form submission events. When the form data is finally submitted, the account and password information is read stealthily.

================

In the Chrome extension implementation, in order to access sensitive information in the Chrome browser, our extension needs to access the DOM tree of a web page. Therefore it needs the cross-site permission to insert the content script when a web page is rendered. The following manifest shows the permission specification.

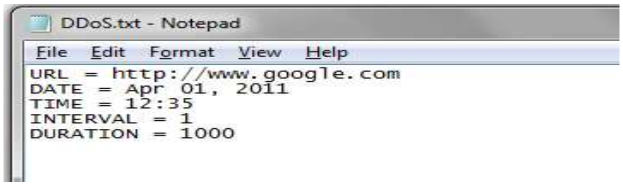
```
"content_scripts": [
   {
        "matches": ["https://online.citibank.com/*"],
        "js": ["jquery.js", "myscript.js"]
    }
  ],
"permissions": [
   "tabs", "https://online.citibank.com/*"
  ],
   ...
```

In the command file, we can instruct the extension to send the sniffed information to the designated email address. With the above specification, when the user browses the target web page, the content script is injected into the target web page, and the JavaScript has full privileges to access all DOM elements including the form with user name and password. It reads these values when the user inputs her password and send to the designated email address. Note the content script can also send sensitive information to the extension core, which in turn sends the data to the outside network.

D. DDoS Attack

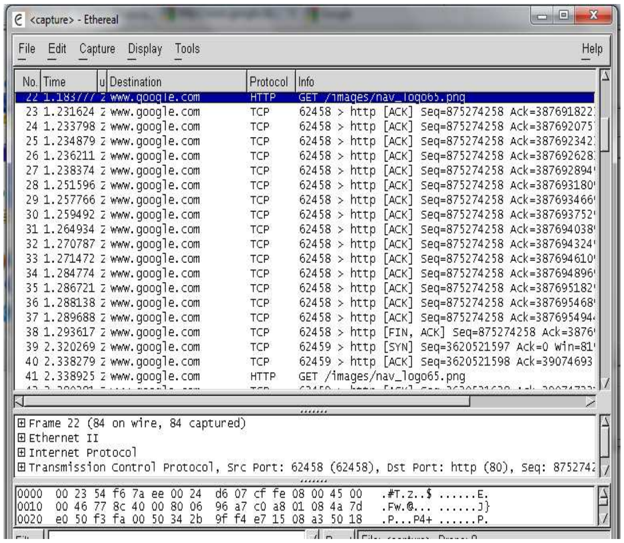
With HDV for IE, we can also launch DDoS attacks. The DDoS attack is implemented in a similar approach as before.

We use the update file as shown in Figure 6(a). In this example, the DDoS information includes the victim's URL (`www.google.com`), attack start time (`12:35`), request interval (`1 second`), and attack duration (`1000 seconds`). After obtaining the victim's URL, the extension can send HTTP requests to the victim as instructed by the DDoS command. Upon receiving the update file, HDV will parse the command. When the specified attack time comes, it will start to send DDoS traffic. Figure 6(b) shows the captured traffic information of this BHO once the attack starts.



```
URL = http://www.google.com
DATE = Apr 01, 2011
TIME = 12:35
INTERVAL = 1
DURATION = 1000
```

(a) Step 1: receiving DDoS command via BHO update



```
No. Time      u Destination       Protocol Info
 22 1.183777 z www.google.com     HTTP     GET /images/nav_logo65.png
 23 1.231624 z www.google.com     TCP      62458 > http [ACK] Seq=875274258 Ack=387691822
 24 1.233798 z www.google.com     TCP      62458 > http [ACK] Seq=875274258 Ack=387692075
 25 1.234879 z www.google.com     TCP      62458 > http [ACK] Seq=875274258 Ack=387692342
 26 1.236211 z www.google.com     TCP      62458 > http [ACK] Seq=875274258 Ack=387692628
 27 1.238374 z www.google.com     TCP      62458 > http [ACK] Seq=875274258 Ack=387692894
 28 1.251596 z www.google.com     TCP      62458 > http [ACK] Seq=875274258 Ack=387693180
 29 1.257766 z www.google.com     TCP      62458 > http [ACK] Seq=875274258 Ack=387693466
 30 1.259492 z www.google.com     TCP      62458 > http [ACK] Seq=875274258 Ack=387693752
 31 1.264934 z www.google.com     TCP      62458 > http [ACK] Seq=875274258 Ack=387694038
 32 1.270787 z www.google.com     TCP      62458 > http [ACK] Seq=875274258 Ack=387694324
 33 1.271472 z www.google.com     TCP      62458 > http [ACK] Seq=875274258 Ack=387694610
 34 1.284774 z www.google.com     TCP      62458 > http [ACK] Seq=875274258 Ack=387694896
 35 1.286721 z www.google.com     TCP      62458 > http [ACK] Seq=875274258 Ack=387695182
 36 1.288138 z www.google.com     TCP      62458 > http [ACK] Seq=875274258 Ack=387695468
 37 1.289688 z www.google.com     TCP      62458 > http [ACK] Seq=875274258 Ack=387695494
 38 1.293617 z www.google.com     TCP      62458 > http [FIN, ACK] Seq=875274258 Ack=3876
 39 2.320269 z www.google.com     TCP      62459 > http [SYN] Seq=3620521597 Ack=0 Win=81
 40 2.338279 z www.google.com     TCP      62459 > http [ACK] Seq=3620521598 Ack=39074693
 41 2.338925 z www.google.com     HTTP     GET /images/nav_logo65.png

⊞ Frame 22 (84 on wire, 84 captured)
⊞ Ethernet II
⊞ Internet Protocol
⊞ Transmission Control Protocol, Src Port: 62458 (62458), Dst Port: http (80), Seq: 875274

0000  00 23 54 f6 7a ee 00 24  d6 07 cf fe 08 00 45 00   .#T.z..$ ......E.
0010  00 46 77 8c 40 00 80 06  96 a7 c0 a8 01 08 4a 7d   .Fw.@... ......J}
0020  e0 50 f3 fa 00 50 34 2b  9f f4 e7 15 08 a3 50 18   .P...P4+ ......P.
```

(b) Step 2: sending DDoS packets

Fig. 6. IE BHO for DDoS: Attack in progress

=============

In the Chrome implementation, we use the same update file as shown in Figure 6(a). With the same command, our extension can command its bots to launch DDoS attacks against the same victim server. We omit the snapshot for brevity.

IV. Discussion

In practice, a botnet may consist of different types of bots, some bots could be IE add-ons and some bots could be Chrome extensions. The botmaster is capable of using the automatic update mechanism to prepare and distribute various command and control information and deliver to all bots. For example, even the same update file could be used for both the IE add-on and the Chrome extension as we have demonstrated.

IE and Chrome use a single-process architecture and a multi-process architecture, respectively. While each of them has its unique advantages considering many design factors such as performance and parallelization, we believe that a multi-process architecture such as Chrome has more advantages on security and reliability. However, the current Chrome extension security model assumes all extensions are benign and only target at preventing malicious web pages. This is insufficient to defend extension-based botnets, especially with coarse-grained permission management. On protection side, efforts are demanded in the following aspects.

First of all, while the permission specification for an extension is good to confine the behavior of the extension, we believe more fine-grained permission management and enforcement in browsers should be mandatory. For example, in Chrome, the permission of injecting content scripts into web pages should be separated from that of cross-site access. This can make the password sniffing attack more difficult as at least two different permissions are needed.

The default extension update mechanism should be improved to make it more user-aware. However, research efforts are needed to make the trade-off between users' interferences and friendly usage. Alternatively, taint analysis or code verification tools can be used to study every update file downloaded from the network. The downside, however, is the significant cost, as the entire update package needs to be tracked and software updates may be very frequent. Thus, more research is required in this aspect.

Offloading expensive taint analysis and software verification operations to cloud can be another option for a regular use. With the cloud computing facilities, a user may submit the downloaded software to some software verification service on a cloud to validate its functions through static and/or dynamic analysis, and uses it only after it has been thoroughly analyzed. On the other hand, a software credit system could be established to let users score the security perspective of software. The popular online social networks can also help in this regard. However, this approach may take a while to be effective as it purely relies on common user's efforts for validation. In addition, such a system could also be attacked by malware developers.

V. Conclusion

Botnet developers are constantly improving their development in order to produce more and more stealthy malware for all kinds of attacks to make profit. While various approaches have been studied or used for botnet attacks, the risk of exploiting widely used browser extensions and their automatic browser extension update mechanisms for command and control channel has not been practically investigated. In this study, we show that it is not difficult to construct stealthy botnet via browser extensions. Given the large user base of browser extensions, it is imperative to devise an effective prevention scheme to mitigate such risks.

Acknowledgement

We appreciate constructive comments from anonymous referees. The work is partially supported by NSF under grants CNS-0746649 and CNS-1117300 and AFOSR under grant FA9550-09-1-0071.

References

[1] "Most spam comes from just six botnets, http://en.wikipedia.org/wiki/Usage_share_of_web_browsers."

[2] J. Kristoff, "Botnets," in *The 32nd Meeting of the North American Network Operators Group*, October 2004.

[3] T. H. Project, "Know your enemy: Tracking botnets," http://www.honeynet.org/papers/bots, March 2005.

[4] S. Racine, "Analysis of internet relay chat usage by ddos zombies," Swiss Federal Institute of Technology Zurich, April 2004.

[5] "One of the most prolific pieces of windows malware has expired," http://news.softpedia.com/news/One-of-the-Most-Prolific-Piece-of-Windows-Malware-Has-Expired-51466.shtml.

[6] N. Daswani, M. Stoppelman, the Google Click Quality, and S. Teams, "The anatomy of clickbot.a," in *Proceedings of the First Workshop on Hot Topics in Understanding Botnets*, Cambridge, MA, April 2007.

[7] Z. Chen, C. Chen, and Q. Wang, "Delay-tolerant botnets," in *Proceedings of IEEE SecureCPN*, 2009.

[8] J. Grizzard, V. Sharma, C. Nunnery, B. Kang, and D. Dagon, "Peer-to-peer botnets: Overview and case study," in *Proc. of the First Workshop on Hot Topics in Understanding Botnets*, 2007.

[9] A. Karasaridis, B. Rexroad, and D. Hoeflin, "Wide-scale botnet detection and characterization," in *Proc. of the First Workshop on Hot Topics in Understanding Botnets*, 2007.

[10] R. Schoof and R. Koning, "Detecting peer-to-peer botnets," http://staff.science.uva.nl/∼delaat/sne-2006-2007/p17/report.pdf, Feburary 2007.

[11] P. Wang, S. Sparks, and C. Zou, "An advanced hybrid peer-to-peer botnet," in *Proceedings of the First Workshop on Hot Topics in Understanding Botnets*, Cambridge, MA, April 2007.

[12] G. Starnberger, C. Kruegel, and E. Kirda, "Overbot: a botnet protocol based on kademlia," in *Proceedings of SecureComm*, 2008.

[13] K. Krips, "The security analysis of browser extensions," http://comserv.cs.ut.ee/forms/ati_report/downloader.php?file=43f05a1a6fa7981ca3422bc3d73b66b8711bc006.

[14] E. Stinson and J. C. Mitchell, "Characterizing the remote control behavior of bots," in *Proceedings of DIMVA*, 2007.

[15] "Internet explorer protected mode," http://msdn.microsoft.com/en-us/library/bb250462(v=vs.85).aspx.

[16] "Mozilla extension versioning, update and compatibility," https://developer.mozilla.org/en/Extension_Versioning,_Update_and_Compatibility.

[17] "Chrome extension autoupdating," http://code.google.com/chrome/extensions/autoupdate.html.

[18] S. Bandhakavi, S. T. King, P. Madhusudan, and M. Winslett, "Vex: Vetting browser extensions for security vulnerabilities," in *Proc. of USENIX Security*, 2010.

[19] "Trojan poses as fake google chrome extension," http://www.bitdefender.com/NW1487-en–Trojan-Poses-as-Fake-Google-Chrome-Extension.html.

[20] "Sun software product map,http://www.oracle.com/us/sun/sun-products-map-075562.html."

[21] Y. Wang, R. Roussev, C. Verbowski, A. Johnson, M. Wu, Y. Huang, and S. Kuo, "Gatekeeper: Monitoring auto-start extensibility points (aseps) for spyware management," in *Proc. of LISA*, 2004.