but reserving IP ranges and then mapping them out would. Even today, we have reserved IP address space that nobody is supposed to use (224.0.0.0 upwards is reserved for multicast, 10.0.0.0/8 is part of RFC 1918's internal address space, and so forth). The work that needs to be done is the following:

- A committee of people must figure out how many IP addresses should be reserved for sending mail – such that we are not likely to run out of space in a couple of decades – and then reserve an appropriate range for it.
- IANA must then reserve that space and come up with rules for how to hand that out to the RIRs. The RIRs must then come up with rules for how to allocate it to the LIRs, who then have to figure out how to allocate it to their customers. They then have to manage the infrastructure necessary to maintain the mappings of who owns what.
- Next, RFCs need to be written on how to send and receive mail over IPv6.
- Then, software vendors need to write code to perform IPv6 email transactions that are able to implement these rules.
- Finally, IP blocklist maintainers need to start populating their lists in IPv6 notation pursuant to the restrictions that are built into the RFCs.

It's a ton of work – years of it – but if we want to start receiving mail over IPv6 then that's what needs to be done.

This restriction of IP space for mail solves one problem but it doesn't solve others. On the one hand, it makes management of IPs scalable for machines that are bots. Today, most spam is sent from botnets. However, botnets do not always send out all of their spam directly – many bots compromise legitimate mail hosts or email accounts and send out spam that way, or create a throwaway account at a free email service and send out small amounts of spam from it before discarding it. This technique is used today but on a smaller scale than spamming directly. If we successfully solve the problem of direct-to-spam botnets, spammers will simply shift the bulk of their spamming to compromised or throwaway accounts.

I guess that means those of us in the e-security industry will always have a job. There's a silver lining to everything!

REFERENCES

[1] Levine, J. A Politically Incorrect Guide to IPv6, part III. http://jl.ly/Internet/v6incor3.html?seemore=y.

[2] Leiba, B. IP Blocklists, Email, and IPv6. http://www.circleid.com/posts/ip_blocklists_email_and_ipv6/.

FEATURE 2

A BROWSER MALWARE TAXONOMY

Aditya K. Sood, Richard J. Enbody
Michigan State University, USA

In this paper, we propose a taxonomy of browser malware. We classify browser add-ons, emphasizing their privileges. Since privileges impact the capability of malware, we use the resulting classification as a basis for our taxonomy. We hope that this taxonomy will provide better insight into the techniques and tactics used by browser malware, and assist in the development of defences.

BROWSER MALWARE – SUBVERTING DESIGN PRINCIPLES

Browsers are becoming a prominent medium for spreading malware infections because they are the interface to the web. Browser malware thrives by exploiting flaws in browser design principles. The design shortcomings most extensively exploited are:

- Insufficient isolation of components in existing browser designs. Browser malware exploits the principle of isolation to run arbitrary code in the context of a running browser to modify other components and corrupt the normal functioning of running components.
- Unrestricted communication among browser components. Browser malware manipulates the principle of integrity by eavesdropping on the inter-component communication interfaces and performing malicious hooks on the HTTP interfaces to control the traffic flow.
- Flaws in same origin policy implementation. Browser malware exploits the principle of same origin policy because persistent browser state is not effectively partitioned. This weakness allows the malware to conduct attacks by exploiting flaws in the same origin policy.
- Insufficient browser customization restrictions. Browser design permits extensive customization to enhance flexibility in the browser, but does so with insufficient restrictions. Browser malware exploits the principle of flexibility by executing malicious code as a part of extensible code.
- Vulnerable privacy models. Existing browser designs have adopted a privacy model which is not robust. Website cookies, user history, browser storage and so on are the foundation of users' privacy. Browser malware overcomes the privacy model by exploiting the

internal browser state to manipulate the interfaces that control the privacy components.

Our focus is on the *Mozilla Firefox* and *Microsoft Internet Explorer* browsers.

BROWSER MALWARE TAXONOMY (BMT)

In developing our taxonomy we define browser malware as a malicious piece of code that results in the circumvention of browser functionality or which uses a browser as a platform to infect operating system (OS) layers thereby. Our taxonomy, BMT, is based on the following critical elements of security that are exploited by the browser malware:

- Browser malware exploits security vulnerabilities in the components, plug-ins and OS layers.
- Browser malware contains malicious extensions that reside in the browser itself and exploit the characteristics of the default browser architecture.
- Browser malware uses the OS as a base to hook and hijack critical browser functions in order to take control of the browser communication channel.

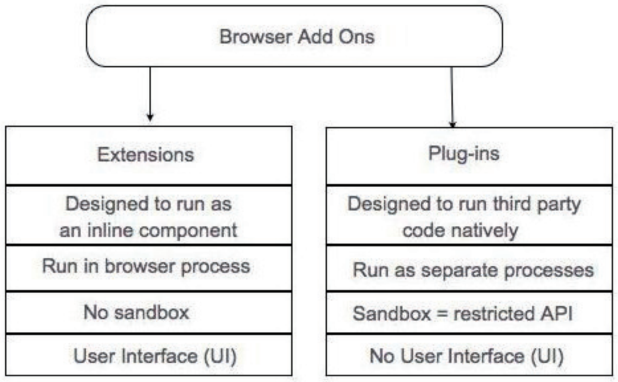

Figure 1: Generic browser extensibility model.

In order to discuss browsers we need to define some terms. To complicate this task, different browsers use different terminology. Browsers depend on a variety of ‘add-ons’ for extensibility, flexibility, and customization. With respect to browser malware we find it useful to divide add-ons into two categories: extensions and plug-ins (see Figure 1). Extensions run in the browser’s context so they have the same rights as the browser itself. An example of an extension would be *NoScript* (for *Firefox*). In contrast, plug-ins run as separate processes and interact with the browser through an API that is more restricted than that used by an extension (some refer to this restriction as a ‘sandbox’). *Adobe Flash* is an example of a plug-in. We have borrowed *Firefox* terminology, but the definitions fit other browsers. One can think of extensions as being part of the browser whereas plug-ins are separate, but intimately connected to the browser. It is the difference in privileges that differentiates add-ons with respect to malware.

Let’s briefly take a look at *Microsoft*’s browser terminology. Browser Helper Objects (BHO) are treated as a part of the *Microsoft Internet Explorer* extension model [1]. In our taxonomy we are treating BHOs as extensions and ActiveX Objects as supporting programs for proprietary plug-ins. An Active X Object has a wide variety of functionality in the way it is used. For example, when installing a BHO from a remote location an ActiveX Object can be used to download that BHO. If an ActiveX Object is allowed to run from a browser, it can perform malicious functions by directly calling OS objects. Custom designed or proprietary plug-ins require an Active X Object to run dynamically if the plug-in is not permanently enabled in the browser.

A BHO is a DLL (Dynamic Link Library) that runs automatically when *Internet Explorer* is loaded. Extensions in *Internet Explorer* use the COM interface to design inline components that run in exactly the same manner as other proprietary components. BHOs extend the browser functionality to a great extent, but can also be used for nefarious purposes. Most of the time Active X Objects and BHOs are installed as DLLs in the operating system because *Microsoft* makes extensive use of DLLs for all types of operations.

Our proposed BMT uses this extension and plug-in distinction along with their different privileges to differentiate between the types of malware.

CLASS A BROWSER MALWARE

Class A browser malware exploits the default monolithic architecture of browsers. It installs itself as a browser component and utilizes the browser model to conduct attacks. This type of malware is quite dangerous because it functions as an inline component. Malicious extensions and browser rootkits fall into this category. Also, class A browser malware can exploit inherent vulnerabilities in the browser native components to download binaries into the operating system. Figure 2 shows the class A browser malware model.

As a browser component, class A browser malware has the full access rights of the browser and runs in the same memory context (address space) as other extensions, so it can perform the following operations:

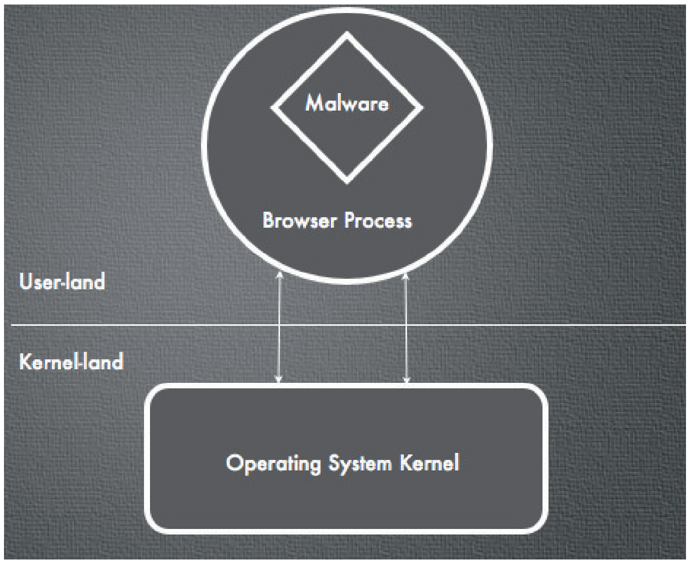

Figure 2: Class A browser malware.

- It is capable of reading and writing to disks, controlling network sockets, tampering with the browser's user interface, stealing stored data, altering the registry and modifying other extensions.
- It can exploit other installed extensions by using JavaScript wrapper functions to change their functionality. It can interact with installed plug-ins such as PDF or *Flash* in order to launch malicious code. In this way it can act as a carrier for trojans.
- Like a rootkit it can hide itself in the browser. For example, *Firefox* extensions can be hidden by manipulating parameters in the install.rdf file and using CSS [2] to install malicious extensions with transparent style metrics. Basically, the aim is to remove entries in the extension manager so that malicious extensions cannot be enumerated. In *Internet Explorer*, it is possible to hide extensions by hooking objects in Security Manager [3] and creating new objects such as invisible tags.
- Class A browser malware can also exploit the security vulnerabilities present in the different browser components such as the rendering engine, JS interpreter, browser engine, XML parser, networking modules and user interface. Exploitation of these vulnerabilities can result in successful execution of drive-by download attacks [4]. JavaScript heap corruption [5, 6] plays a critical role in the successful execution of these types of attacks.
- Class A browser malware can make explicit use of asynchronous HTTP requests via AJAX with associated events to generate listener functions for communication with third-party servers. It can use encrypted protocols for data transfer in a secure manner.

Examples of class A browsers include Firefox FormSpy [7], FFSniff [8], Sothink Web Video Downloader 4.0 spreading Win32.LdPinch.gen [9], Master Filer spreading Win32.Bifrose.32.Bifrose Trojan [9], Download.ject [10, 11] and MyWay Searchbar [12, 13].

Note that browser exploit packs such as the Phoenix [14] and BlackHole [15, 16] do not install any extensions, but do exploit browser vulnerabilities to download malicious executables in the system. Therefore, they display some of the typical behaviour of Class A browser malware.

CLASS B BROWSER MALWARE

Class B browser malware exploits vulnerabilities in the plug-in interface in order to cause an infection. Since most plug-ins originate from third-party vendors, inherent vulnerabilities in plug-ins play a critical role in determining the success of class B browser malware. Generally, plug-ins run as separate processes and are usually placed in a sandbox so the interface with the browser is restricted.

Plug-ins are platform-independent code so vulnerabilities in third-party code such as *Adobe Flash* can broadly impact browser security. The impact is significant because the risk of exploitation in third-party software is transferred to the browser. This interdependency results in hybrid vulnerabilities that exist when the plug-in code is run simultaneously with the browser. A malicious plug-in can use JavaScript in a web page (loaded in a sub window) to

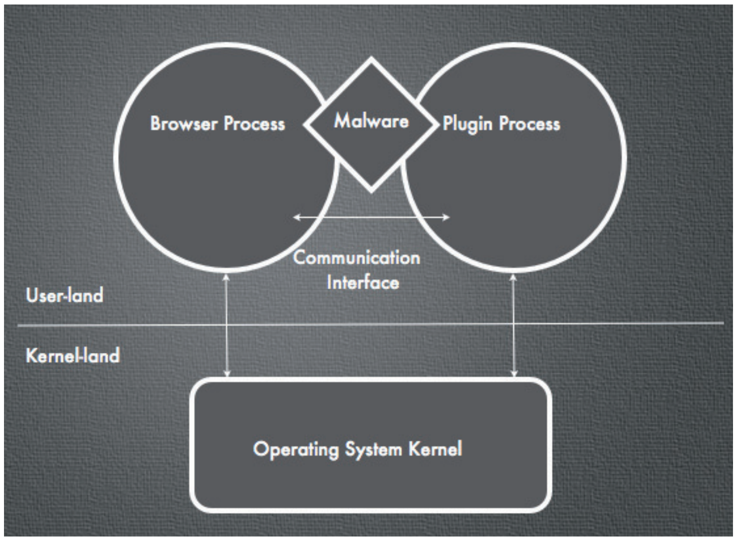

Figure 3: Class B browser malware.

perform various functions. A web page's JavaScript runs under restricted privileges (sandboxed) so it cannot interact much with the internal browser components. Of course, vulnerabilities in the sandboxing can allow modification of the internal browser components. In any case, plug-ins operate outside of browsers and can act as a parasite to use browser resources to spread infections.

Figure 3 shows the high-level view of class B browser malware.

Class B browser malware shows the following characteristics:

- Custom designed plug-ins can run malicious scripts in the browser and exploit DOM to carry out XSS for stealing sensitive information, phishing, malvertisements and social engineering attacks. In *Firefox*, malicious plug-ins make use of the Netscape Plug-in Application Programming Interface (NPAPI), which is used to design platform-independent code. NPAPI plug-ins are specifically used by class B browser malware for malicious Internet functionality which is a potential playground for spreading infections. In *Internet Explorer*, custom designed plug-ins use Active X Objects to execute arbitrary code that can manipulate browser resources.
- Malicious plug-ins can carry exploit code in order to drop binaries in the OS by exploiting vulnerabilities in the plug-in interface – so-called drive-by download attacks. Examples include the exploitation of vulnerabilities in the *Adobe Reader*, *Flash* and *Silverlight* plug-ins.
- Apart from JavaScript heap corruption, class B browser malware also uses evasive techniques such as RC4 encryption, splitter modules including variables and arrays, multiple level compressions and encoding as well as cross referencing of objects in order to evade detection. However, it supports both JavaScript and non JavaScript-based exploits.

Real-world examples of class B browser malware include Trojan.Pidief [17], Exploit.PDF-JS.Gen [18], SWF AdJack Gnida [19], Trojan:SWF/Redirector.I [20] and TrojanDownloader:SWF/Nerner.A [21].

Note that browser exploit packs such as BlackHole and Phoenix show characteristics of class B malware because these packs exploit vulnerabilities in third-party support plug-ins such as *Adobe Flash* and Java.

CLASS C BROWSER MALWARE

Class C browser malware typically resides in the operating system and exploits the browser interface. It also exploits OS layer vulnerabilities by using the access rights of the user of the operating system. In general, class C browser malware uses system-level APIs to attack the browser or plug-in processes in order to control the browser communication interface. This malware also contains system-level rootkits specifically aimed at exploiting the browser interface with the OS. Class C browser malware establishes itself in either the user-land or kernel-land layers of the OS, or some hybrid of the two. As noted earlier, both class A and class B browser malware play a crucial role in dropping class C browser malware in the OS by exploiting certain vulnerabilities in browsers and plug-ins. This observation indicates that dependencies exist between all classes of browser malware that are presented in this taxonomy. Figure 4 shows a high-level view of class C browser malware.

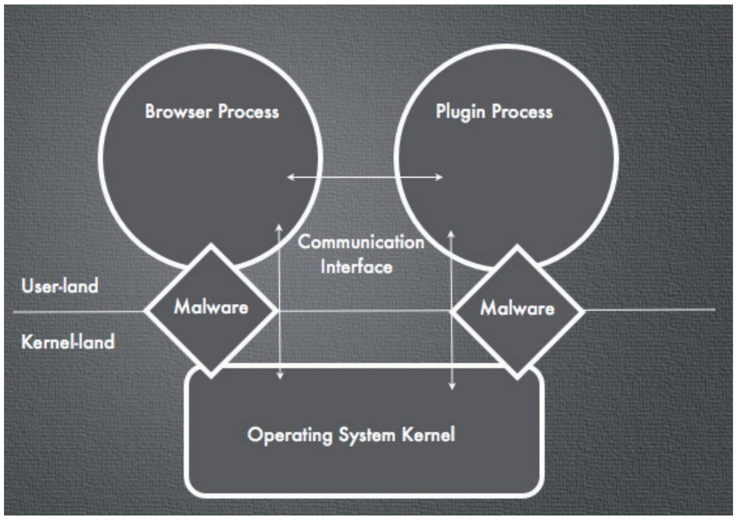

Figure 4: Class C browser malware.

Class C browser malware shows the following characteristics:

- This class of malware typically behaves like a rootkit and hides itself in the OS so that the possibility of detection is reduced. Class C browser malware basically aims to control the browser's communication interface with the Internet to manipulate the traffic flow. Class C browser malware implements hooking [22] in order to take control of the execution flow. In user-land space, this class of malware uses Import Address Table (IAT) hooking, inline hooking and DLL hijacking using APIs. In kernel-land space, this class of malware primarily performs hooking through the System Service Descriptor Table (SSDT) in order to control the browser-related functions in the kernel.

- Class C browser malware is capable of hooking the browser communication channel, traffic monitoring, injecting malicious traffic into the browser, circumventing OS firewalls, generating fake website pages and keylogging POST requests and all system-related activities. This ability is critical because class C browser malware has complete control of the standard HTTP functions in OS libraries which are used heavily by browsers for all types of work.

Real-world examples include the Zeus bot [23], SpyEye bot [24], IRC bots [25] and PRRF [26].

Note that browser exploit packs such as BlackHole and Phoenix actually exploit issues in browsers and plug-ins to drop class C browser malware into the system.

CONCLUSION

In this browser malware taxonomy we have presented three different classes of browser malware classified on the basis of a browser architectural model. In particular, differences in privileges are critical in the classification. Class A browser malware resides in the browser process and acts as an inline component of the browser. Class A browser malware exploits vulnerabilities in the browser process. Class B browser malware takes the form of malicious plug-ins and also exploits vulnerabilities in third-party vendor software. Class C browser malware resides in the operating system and controls the browser communication channel from outside.

We hope that this taxonomy will provide better insight into the techniques and tactics used by browser malware, and assist in the development of defences.

REFERENCES

[1] Microsoft. Verified Security for Browser Extensions. http://research.microsoft.com/pubs/141971/tr.pdf, 2010.

[2] Devaux, C.; Lenoir, J. Browser Rootkits. Hack.lu, 2008. http://archive.hack.lu/2008/lenoir_presentation.pdf.

[3] Wueest, C.; Florio, E. Firefox and Malware: When your browser bytes you. Virus Bulletin International Conference 2009. http://www.virusbtn.com/pdf/conference_slides/2009/Wueest-Florio-VB2009.pdf.

[4] Barwinski, M.; Irvine, C.; Levin, T. Empirical Study of Drive-by-Download Spyware. http://www.cisr.us/downloads/papers/06paper_spyware.pdf.

[5] Sotirov, A. Heap Feng Shui in JavaScript. Black Hat Europe 2007. http://www.blackhat.com/presentations/bh-europe-07/Sotirov/Presentation/bh-eu-07-sotirov-apr19.pdf.

[6] Chennete, S.; Joseph, M. Detecting Web Browser Heap Corruption Attacks. Black Hat USA 2007. https://www.blackhat.com/presentations/bh-usa-07/Chenette_and_Joseph/Presentation/bh-usa-07-chenette_and_joseph.pdf.

[7] Oswald, E. Trojan Hides Itself as Firefox Extension. BetaNews. http://www.betanews.com/article/Trojan-Hides-Itself-as-Firefox-Extension/1153934797.

[8] Costoya, J. Malicious Firefox Extensions. Trend Micro blog. http://blog.trendmicro.com/malicious-firefox-extensions/.

[9] Claburn, T. Mozilla Removes Two Malicious Firefox Add-Ons. Information Week. http://www.informationweek.com/news/security/vulnerabilities/222700171.

[10] Schneier, B. Schneier Micros. Schneier on Security. http://www.schneier.com/blog/archives/2004/10/schneier_micros.html.

[11] Wikipedia. Download.ject. http://en.wikipedia.org/w/index.php?title=Download.ject&oldid=423105396.

[12] Leyden, J. Dell rejects spyware. The Register. http://www.theregister.co.uk/2005/07/15/dell_my_way_controversy/.

[13] Wikipedia. MyWay Searchbar. http://en.wikipedia.org/w/index.php?title=MyWay_Searchbar&oldid=427757462, 2005.

[14] SecNiche Security. Phoenix Exploit Kit (2.4) – Infection Analysis. Malware At Stake Blog. http://secniche.blogspot.com/2010/10/phoenix-exploit-kit-24-analysis.html.

[15] Websense Blog. Black Hole Exploit Pack. http://community.websense.com/blogs/securitylabs/pages/black-hole-exploit-kit.aspx.

[16] SecNiche Security. Java OBE + Blackhole Dead Man Rising. Malware At Stake Blog. http://secniche.blogspot.com/2011/02/java-obe-tookit-exploits-blackhole-dead.html.

[17] Symantec Security Report. The Rise of PDF Malware. http://www.symantec.com/content/en/us/