By Breno Cunha on 2013-06-26 14:03:39 1 comentário

Perspectives on exploit development and cyber attacks

Browsers and related components are actively being exploited in the wild, and it is by now, the most targeted platform for cyber criminals. From exploit-kits, looking for new bots and bank credentials, to more developed attacks aimed at stealing intellectual property, there has been a steady uprising in client-side exploitation.

Back in 2009, Operation Aurora 1 echoed throughout the media landscape as a supposed Chinese attack aimed at Google and other companies such as Adobe Systems and Juniper Networks. Almost four years later, the U.S. Department of Labor website was compromised 2with the intent to host an exploit for Internet Explorer and further malicious code spreading.
By 2008, most of the known memory protections were already in place, but they didn't succeed, and are still failing in stopping large-scale client-side exploitation and malware distribution. Why is that?

To answer this question and understand the threat landscape, let's analyze the layered exploit mitigation scheme, in contrast to the exploits present in cyber-attacks. The forthcoming sections focus on memory protections as well as exploitation techniques, in order to try to explain why there are so many bugs being exploited in the wild.

Memory Protections Overview

In 2002, Microsoft introduced Buffer Security Check (/GS) 3, with the objective to protect the return address from being overwritten. With the release of GS v2 (2005), stack protection was extended in order to prevent variable and parameter overwrites.

SafeSEH (2003) 4 and SEHOP (2008) 5 bring additional checks and restrictions to the Structured Exception Handling mechanism. Albeit SEHOP can be bypassed 6, the technique described by Sysdream relies on the prediction of “ntdll!FinalExceptHandler” address, which is randomized by ASLR upon every system boot.

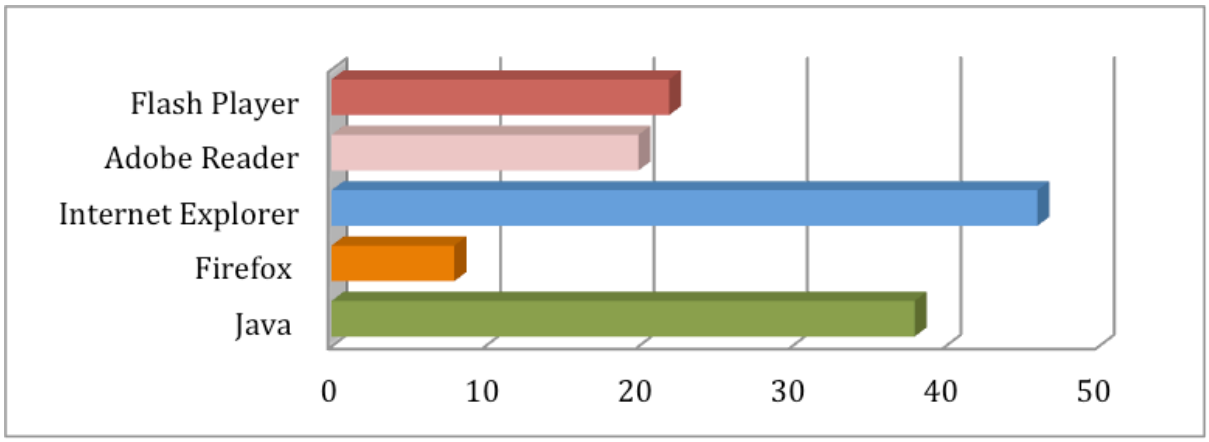
With the introduction of Windows Vista, Microsoft's Heap Manager undergone an extensive hardening (2006) 7. As a result, exploits that relied on the corruption of heap metadata had to face randomly encoded heap data structures. Also, heap spray 8 wasn't supposed to be so reliable anymore, since non-determinism was introduced by variations in the allocation algorithm 9.

According to Matt Miller’s presentation at Black Hat 2012 10, there are still classes of bugs, both in stack and heap, that Microsoft’s exploit mitigations fails to protect. But

these bug classes are either very rare or hard to exploit in the presence of DEP (2004) and ASLR (2006).

Effectiveness of memory protections in the real-world

Reliable exploits became harder to develop, requiring much more skilled writers and lots of time. By elevating the cost to perform an attack, one should expect a decrease in the number of attacks related to memory corruption bugs. but that's not what happened.

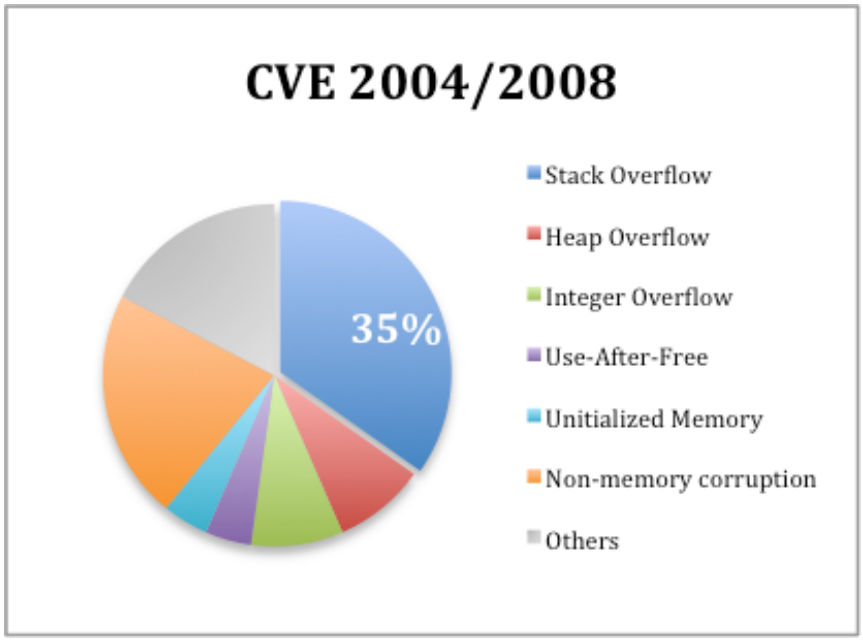


The above graph describes the top five platforms targeted by exploit-kits within the 2007/13 time period, and it is based on data provided by the malware research community 11. By far, the most targeted software are Java and Internet Explorer. The great majority of Java bugs are logic flaws, and thus, not directly affected by the aforementioned memory protections. Internet Explorer, as a Microsoft product, takes advantage of the latest OS security enhancements, and thereby, should impose a challenge for the guys trying to exploit it.

After classifying the Internet Explorer bugs in accordance to vulnerability type, two patterns emerged, and therefore are discussed in detail in the following sections.

The next sections contain two charts that do not represent all publicly disclosed bugs. Instead, they are based on 44 different vulnerabilities affecting Internet Explorer. Although CVE numbers date as early as 2004, they were first seen within exploit-kits and massive attacks in 2007. The complete CVE reference can be found at the appendix.

Stack overflow prevalence



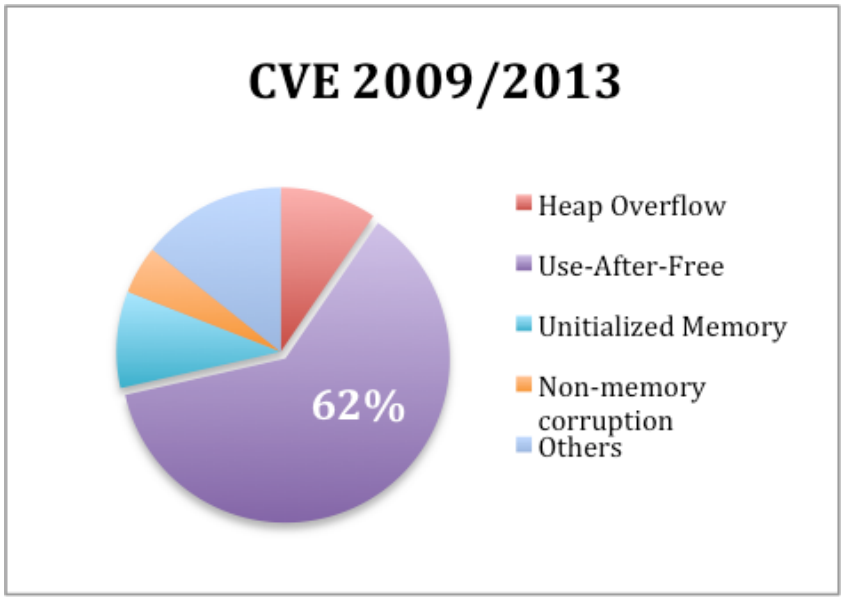
Until 2008, the most exploited bugs were stack buffer overflow, meaning that back then, the stack memory protection wasn't so effective. With the introduction of GS Cookie, other exploitation techniques emerged, and stack overflows could still be exploited by corrupting the SEH handler, which is also located in the stack.

SafeSEH was supposed to stop this exploitation technique, but it was surely failing. SafeSEH verifies if all executed SEH handlers are present in an already defined handler list (generated at compile time). Therefore, each loaded module must have its handler list. As a consequence, the arbitrary execution of a previously undefined SEH handler would terminate the application. Unfortunately enough, the protection relies on the presence of such handler list for every loaded module. So, if one single module is not compiled with the /SafeSEH switch, the protection mechanism cannot ensure the handler's authenticity as a forged one or, as a valid handler present in the unsafe module.

In case an executable, or one of its dependencies, is not SafeSEH enabled, it is trivial to build a SEH based exploit for a stack buffer overflow, and this explains the high number of stack exploits.

The introduction of SEHOP in 2008 successfully mitigated the earlier SEH based exploits. Because of its nature — by being a runtime protection, rather than a compile time protection — it benefits both new and legacy applications by default. After SEHOP stack exploits had a great decrease and, in terms of Internet Explorer, the last stack exploit found in the wild is CVE-2008-3008 12.

Rise of Use-After-Free exploits



The last four years were taken by Use-After-Free exploits, playing a role of 62%, thus placing it as the most exploited bug class in Internet Explorer. The previous analysis showed that a single weak point can break the "exploit's cost increase" strategy adopted by Microsoft mitigations, and it seems that again, such a weak point still exists.

Use-After-Free bugs are not related to heap metadata corruption itself, instead they are caused by the application misuse of a previously freed pointer (free part of use-after-free). The exploitation technique is pretty straightforward. It consists of inducing the heap to a predictable memory layout, in such a way that the freed memory block is reallocated by the attacker and filled with crafted data. Later on, the application dereferences the previously freed pointer (use part of use-after-free) and operates upon the crafted data.

Well, after all the work playing heap Tetris, the attacker must debug the program to know which actions performed on the controlled data can lead to code execution, generally through an arbitrary write. This requires great reverse engineering skills, which is not cheap at all, and cannot explain the use-after-free boom in the wild.

A little more on Use-After-Free

There is one specific kind of use-after-free exploitation technique, related to C++ Objects and the way they handle virtual functions. Basically, the first DWORD of a C++ Object is a pointer to its virtual function table (vtable) and accessing a method like "myObj.method1" results in the following assembly code:

```
MOV EAX,DWORD PTR DS:[ECX]  ; ecx holds a pointer to "myObj"
CALL DWORD PTR DS:[EAX+30h] ; call vtable function pointer at index 30h
```

The MOV operation stores myObj's vtable pointer into EAX. And after that, the CALL instruction transfers the execution to "method1", which is located at the index 30 of the vtable.

When the freed memory block represents a C++ object, it is possible to achieve code execution by crafting an object with all vtable entries pointing to attacker-controlled data (heapspray), and therefore, calling any method on the fake object would cause the shellcode execution.

The above-mentioned technique is quite generic and easy to exploit. Thus, since it is a direct function pointer overwrite, there is no need to debug the process and try to nail down a complex arbitrary write. The low cost of developing such exploits seems to be compatible with the rise of Use-After-Free exploits, but it does not make it all possible by itself, there is still the need to overcome DEP/ ASLR in order to build a reliable shellcode.

ASLR Challenge

The first weak point is related to Use-After-Free bugs in conjunction with C++ objects, but to defeat full ASLR, one shall perform an information leak 13, which is generally accomplished by exploiting an arbitrary read or write operation. This means that to successfully bypass ASLR, the attacker must exploit an additional vulnerability or, in some cases, the same vulnerability twice.

After digging a little further and reading the actual exploit codes found in the wild, things became clearer and shed light towards a second weak point, which turned out to be 3rd party plugins present in almost all browsers. At least two different popular plugins were not compiled with the /DYNAMICBASE switch 14, and thus are not randomized by ASLR.

Java 6 and MS Office 2007/2010 were shipped including non-ASLR modules, such as the msvcr71.dll. This broke the layered mitigation scheme and was the kind of opportunity the attackers were looking for. With a single non-ASLR module loaded in the process, it is possible to build a reliable ROP-based shellcode and reuse it in various different exploits.

The above scenario, by being a low-cost attack (as stated by the aforementioned weak points), justifies the great number of Use-After-Free bugs present in exploit-kits. Although Microsoft is investing in state-of-the-art protection mechanisms, exploit writers are opportunists and whenever there is a weakness they will find it.

Exploit development trends and what to expect

Although Use-After-Free still plays a big role, a decrease in generic exploitation of such bugs in Internet Explorer is expected. Those attacks rely on outdated 3rd party plugins, and thus, have a limited lifetime.

Security researchers already stated that info leak [15] is the new trend in exploit development, but it takes time until it becomes popular to the average attacker. Also, attacks involving info leak are far more complex than the previously described Use-After-Free exploits, elevating the final cost of the campaign.

In january 2013 [16], the Cool Exploit Kit (aka Cool EK) was updated to include an advanced exploit for the CVE-2012-1876 [17], a heap-based buffer overflow in Internet Explorer 6 through 10. The vulnerability was first disclosed by Vupen [18] at 2012 Pwn2own Contest. It is the first case of a more sophisticated exploit, targeting Internet Explorer, to be incorporated into an exploit-kit.

At 2013 Pwn2own, held last march, Vupen demonstrated the exploitation of an Integer Overflow (CVE-2013-2551) [19] in Internet Explorer 10 running on Windows 8 [20]. The vulnerability allows for an attacker to update the size attribute of an Array object, without triggering the array reallocation. After changing the size to a max of 0xFFFF (two bytes), it's then possible to read and write arbitrary data after the end of the array.

They went even further and discussed a Dynamic ROP technique, which consists of leaking the whole vgx.dll and building a ROP payload based on the currently loaded version of the dll. This results in a universal exploit for IE 6 through 10 in all versions of Windows.

By the time of writing this article, a PoC for the CVE-2013-2551 had already been released on a chinese forum [21] and further incorporated to the Metasploit framework. As it is not the universal exploit, the default configuration in Metasploit targets IE 8 and uses a non-ASLR Java 6 ROP payload. Additionally, there is info leak support for two versions of ntdll.dll, present either in a fresh Win7SP1 or after the patch MS12-001.

It's a matter of time to see this exploit in the wild, and it is plausible that it will support more versions than the exploit provided by Metasploit. Cool EK already took the first steps in maximizing the scope of the attack and, as a matter of fact, its latest release includes 18 distinct ROP payloads targeting different Windows versions, languages and service packs.

Vupen also demonstrated the exploitation of a Use-After-Free vulnerability in Firefox [22], it seems they have found a way to circumvent DEP/ASLR on Windows 7. They did not release any technical information, and it is still unknown whether the technique is intrinsic to the vulnerability itself or, if it is generic enough to be used in different exploits with little effort for the attacker (low cost).

Appendix

CVE-2004-0549	CVE-2006-4777	CVE-2008-2463	CVE-2010-0806	CVE-2012-1889
CVE-2004-1043	CVE-2006-4868	CVE-2008-3008	CVE-2010-3962	CVE-2012-4792
CVE-2005-0055	CVE-2006-5559	CVE-2008-4844	CVE-2010-3971	CVE-2012-4969
CVE-2005-2127	CVE-2006-5745	CVE-2009-0075	CVE-2011-1255	CVE-2013-1347.
CVE-2006-0003	CVE-2007-0024	CVE-2009-0076	CVE-2011-1260	
CVE-2006-4704	CVE-2007-0038	CVE-2009-1136	CVE-2011-1996	
CVE-2006-0005	CVE-2007-2222	CVE-2010-0248	CVE-2012-0003	
CVE-2006-1359	CVE-2007-5779	CVE-2010-0249	CVE-2012-1875	
CVE-2006-3643	CVE-2008-0015	CVE-2010-0483	CVE-2012-1876	
CVE-2006-3730	CVE-2008-0624	CVE-2010-0805	CVE-2012-1880	

Stack Overflow
Heap Overflow
Integer Overflow
Use-After-Free
Unitialized Memory
Non Memory Corruption
Others

References

[1] http://www.mcafee.com/us/threat-center/operation-aurora.aspx

[2] http://eromang.zataz.com/2013/05/10/department-of-labor-watering-hole-campaign-review/

[3] http://msdn.microsoft.com/en-us/library/8dbf701c%28v=vs.80%29.aspx

[4] http://msdn.microsoft.com/en-us/library/9a89h429%28VS.80%29.aspx

[5] http://blogs.technet.com/b/srd/archive/2009/02/02/preventing-the-exploitation-of-seh-overwrites-with-sehop.aspx

[6] http://www.sysdream.com/article-outrepasser-sehop

[7] http://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Marinescu.pdf

[8] http://www.blackhat.com/presentations/bh-europe-07/Sotirov/Presentation/bh-eu-07-sotirov-apr19.pdf

[9] http://blogs.technet.com/b/srd/archive/2009/08/04/preventing-the-exploitation-of-user-mode-heap-corruption-vulnerabilities.aspx

[10] http://media.blackhat.com/bh-us-12/Briefings/M_Miller/BH_US_12_Miller_Exploit_Mitigation_Slides.pdf

[11] https://docs.google.com/spreadsheet/ccc?key=0AjvsQV3iSLa1dE9EVGhjeUhvQTNReko3c2xhTmphLUE&usp;=sharing

[12] http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-3008

[13] http://zhodiac.hispahack.com/my-stuff/security/Flash_ASLR_bypass.pdf

[14] http://msdn.microsoft.com/en-us/library/bb384887.aspx

[15] http://media.blackhat.com/bh-us-12/Briefings/Serna/BH_US_12_Serna_Leak_Era_Slides.pdf

[16] http://blogs.technet.com/b/mmpc/archive/2013/05/07/cve-2012-1876-recent-update-to-the-cool-exploit-kit-landing-page.aspx

[17] http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-1876

[18] http://www.vupen.com/blog/20120710.Advanced_Exploitation_of_Internet_Explorer_HeapOv_CVE-2012-1876.php

[19] http://osvdb.org/show/osvdb/91197

[20] http://www.vupen.com/blog/20130522.Advanced_Exploitation_of_IE10_Windows8_Pwn2Own_2013.php

[21] http://binvul.com/viewthread.php?tid=311

[22] https://twitter.com/VUPEN/status/309505403631325184