Heap Sprays to Sandbox Escapes: A Brief History of Browser Exploitation

By Rahul Kashyap

This article looks at the evolution of various exploitation techniques used for widely exploiting client applications along with exploit prevention technologies available in the industry.

Browser exploitations have been on a steady rise for the past few years. The recent dearth of "wormable" Microsoft service vulnerabilities has made client exploitation a prime attack vector for malware deployments by the adversaries. In this article we look at the evolution of various exploitation techniques used for widely exploiting client applications along with exploit prevention technologies available in the industry. We'll use Internet Explorer and its closely associated plugins as the primary focus. Many of the exploitation and remediation technologies mentioned are applicable for other browsers, including other client applications like Adobe Reader, for example. The constantly evolving, huge code base in browsers provides a large surface for attackers to continuously exploit the browsers. Coupled with the fact that most of these attacks have a social engineering component, attackers are consistently successful in infiltrating enterprise infrastructures.

The constantly evolving, huge code base in browsers provides a large surface for attackers to continuously exploit the browsers.

This brief article is primarily focused on exploitation technologies that were adopted by attackers to actively exploit clients and use techniques popularly known as "first stage" shellcode. Post exploitation of the vulnerable application, the payload typically executes in more stages to finally install malware on the victim system. For the sake of brevity, we won't deal with kernel mode exploitation but rather focus on the widely used "user mode" exploitation primitives. We'll conclude with some recommendations and best approaches needed for protecting the end users.

Memory protection schemes

Memory protection schemes have been in vogue for several years as a part of layered defense on the endpoint. Since ample detailed information for each of these have already been published, we'll cover the basic concepts of some relevant schemes in this article and then examine how these have fared with the exploit writers in the last few years.

DEP – Data Execution Prevention

DEP was introduced in Windows XP SP2. It was designed to disallow an application from executing code from a non-executable memory region. Most of the modern CPUs available support hardware based DEP (NX).

Memory Protection: DEP	Introduced: 2004
OS Version Windows XP Service Pack 2	Internet Explorer Versions IE 6 and IE 7

ASLR – Address Space Layout Randomization

Introduced in Windows Vista, ASLR introduces randomization to the base address of executables, which basically means that every function entry points will be in somewhat unpredictable memory locations. This was designed to break the

Security can't stand still

Customers want more services. More ways to interact. You need to deliver timely, innovative business services along with new levels of security. Security solutions from CA Technologies provide safe, convenient access for users and powerful identity intelligence for IT. Now, you can secure collaboration and enable innovation.

+ **FIND OUT** how CA Technologies can help you secure IT to reduce risk. Watch our webcast on Mobile Authentication at **ca.com/securitywebcasts**

Join Webcast

Mobile Authentication: Key Considerations for Developing Your Strategy.
January 16th, 2013 | 1:00 p.m.

agility made possible™

ca technologies

traditional ret-to-libc[1] type of exploits which were designed to evade non-executable stack protections.

Memory Protection: ASLR	Introduced: 2007
OS Version Windows Vista and Windows 7	Internet Explorer Versions IE 7 and IE 8

There are several other schemes that broke some of the older exploitation techniques, like SAFESEH and SEHOP break exploits that used to overwrite the SEH handler for reliable exploitation. A combination of these memory protection schemes should provide a reasonable coverage from a layered defense standpoint. Let's see how the exploitation trends have evolved in past few years with the various flavors of Windows (XP/Vista/7/8).

The need for reliable exploitation

One of the most important aspects for any exploit to succeed is to ensure that the attacker payload can execute arbitrarily with reliability, ideally across various operating systems. This usually calls for a lot of "tweaks" in the exploit. This need has led to the development of some interesting techniques by exploit writers in the past few years. Some of the most commonly used exploitation mechanisms are listed below in brief.

Heap spray and enhancements

The heap spray[2] technique on client exploits is known to have been created around 2004, just when people were beginning to adopt windows XP SP2. The idea was straight forward: create enough heap blocks using scripting languages such as JavaScript so that a reliable location can be attained for the shellcode to reliably execute, without hunting for a reliable offset in the memory.

A typical way to launch a heap spray would look like:

```
// Create nopsled of heap blocks large enough to
// land at a pre-determined offset for the
// payload to land and execute arbitrary code
// post exploitation of the vulnerability

var memory = new Array();

for (i=0;i<heapblocks;i++) {

memory[i] = nopsled + shellcode;

   }
```

This helps to align the memory such that we have a bunch of consecutive NOPs (No Operations) followed by the shellcode (figure 1). After some spraying consecutive areas and taking advantage of the deterministic heap layouts, the final output can be a predictable entry point for the shellcode to execute.

This technique was eventually used by several malware variants for exploiting numerous vulnerabilities, most of which were discovered in IE 6 & 7. Around that time, Internet Explorer was reported to be the most widely used browser with over 90%[3] market share.

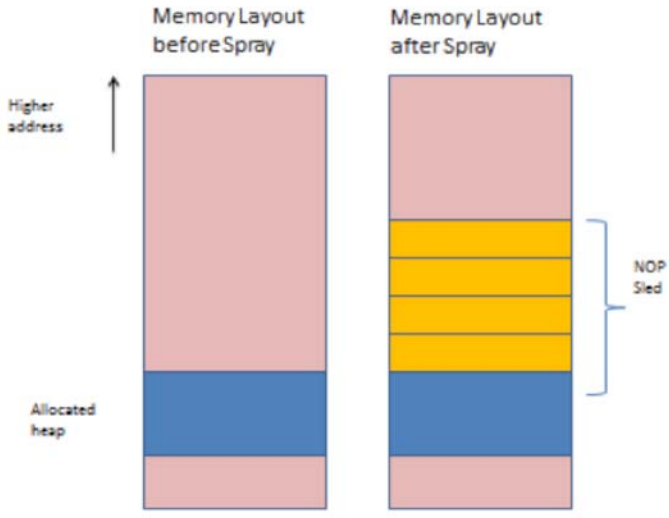

Figure 1 – Before and after heap spraying

The heap spray technique was leveraged by malware for exploiting several high profile zero-day vulnerabilities in Internet Explorer. Formats like .ANI (CVE-2007-0038), VML (CVE-2006-4868), CreateTextRange (CVE-2006-1359), and Operation Aurora exploit (CVE-2010-0248) are just a few of the widely exploited vulnerabilities. It comes as no surprise that this technique quickly became the favorite of exploit writers in short time due to its ease of implementation.

The overall reliability of the standard heap spray technique was reasonably fair and there were some enhancements made, notably one of which was introduced as "heap feng shui."[4] In this the author pointed out the importance of controlling the heap state for higher reliability of heap sprays by using the fact that the heap allocation routines were deterministic. This technique dealt uniquely with situations in which the heap allocations were made by custom heap allocation engines like OLEAUTH32.DLL. It optimized the heap allocation by reverse engineering the algorithm and providing predictability in aligning heap chunks.

The algorithm was published, and associated malware activity was discovered in the wild exploiting ActiveX components vulnerability (CVE-2006-4777).

Heap spray and ROP

Along with Windows Vista came DEP and ASLR enforcement capabilities that by default got consumed by browsers like IE 8. This basically meant that the old traditional heap sprays would no longer be reliable. However, if one borrowed the tricks of the traditional ret-to-libc exploits, one could craft a chain of instructions that returns one instruction into the address of the next instruction which can then finally lead to

1 http://www.phrack.org/issues.html?issue=58%5C&id=4.

2 http://skypher.com/wiki/index.php/Hacking/Heap_spraying.

3 http://en.wikipedia.org/wiki/Internet_Explorer#Desktop_Market_share_by_year_and_version.

4 http://www.phreedom.org/research/heap-feng-shui/heap-feng-shui.html.

the API to be executed. This technique was christened ROP,[5] short for Return Oriented Programming due to its nature; each of these chains is popularly known as a "gadgets." So to craft the payload, one could spray the heap using a technique like heap feng shui such that the target address would be well aligned at the beginning of the gadget. Leveraging ROP techniques usually needs some amount of research and improvisation per vulnerability as the exploitability of the environment is likely to vary.

Several malware samples leveraging heap spray and ROP were found in the wild, for example, CVE-2012-4969 with vulnerable versions of IE 6/7/8/9 on Windows 7 and older windows platforms.

Heap spray and ROP gained prevalence with the malware authors. ASLR still made exploitation difficult. However, attackers found several loopholes in the browser infrastructure like plugins to bypass restrictions of ASLR. More specifically, if a plugin was not compiled with ALSR, then the exploitation became much easier. The "Same ID Property use after free vulnerability" (CVE-2012-1875) discovered in the wild[6] is used exactly thus: the exploit used the non-ASLR compiled Java virtual machine `msvcr71.dll` to execute code leveraging heap spray and ROP techniques.

JIT spray

The JIT spray[7] technique was designed to evade DEP + ASLR available in the newer Windows OS releases. The concept is interesting: the attacker would take advantage of the fact that JIT (Just-In-Time) compilers are allowed to create executable code which was obviously designed for optimization and to be compatible with DEP. By carefully generating enough JIT compiled code (NOP sled), which is crafted such that it results into the shellcode post exploitation of the vulnerable condition, a reliable second stage payload can be executed leveraging stage 0 payload. This mechanism is ideal for exploiting plugins like Flash that uses JIT compilers along with browsers.

Non-memory corruption vulnerabilities

So far we've been focused on memory corruption vulnerabilities that have been exploited by malware to bypass memory protection schemes in the recent past. There are other genres of serious issues which cannot be mitigated by memory protection schemes as discussed above.

Java is a very popular application used widely by many users and developers. Recent trends clearly indicate that attackers are targeting Java as one of the prime attack vectors because of its ubiquity, multi-platform exploit portability, and the failures of its security technologies like memory protection schemes. The wide exploitation of Java 7 vulnerability (CVE-2012-4681) was particularly concerning because it was able

5 https://www.blackhat.com/presentations/bh-usa-08/Shacham/BH_US_08_Shacham_Return_Oriented_Programming.pdf.

6 http://blogs.mcafee.com/mcafee-labs/active-zero-day-exploit-targets-internet-explorer-flaw.

7 http://www.semantiscope.com/research/BHDC2010/BHDC-2010-Paper.pdf.

to escape the sandbox – in which untrusted code was able to reach out to restricted packages that were not supposed to be accessible by the inherent design of the security policy of Java sandbox. The ensuing result was that Java applets were able to access sensitive OS instructions and execute arbitrary code. Similarly, the Java Atomic Reference Array (CVE-2012-0507), Java Rhino (CVE-2011-3544), and Java Applet Field Byte code verifier (CVE-2012-1723) were sandbox design issues that were actively abused by malware.

Since there was no memory corruption involved in these vulnerabilities, the available memory protection schemes were of little help, and to make the situation worse, exploits for those could work reliably on most of the popular operating systems (Windows, Mac OSX, etc.) using multiple browsers (IE, FireFox, Chrome, etc.).

IE 10 and Windows 8

At the time of writing this article, IE 10 for Windows 8 is not officially released. However, there are some great improvements[8] like enhanced SmartScreen Filter, HTML5 Sandbox, ForceASLR, HEASLR, all of which will provide significant gains against well-know exploitation techniques. Besides this, Microsoft has also released EMET, which is a tool to plug most of the known reliable exploitation vectors.

Looking at the history of memory protection scheme evolution and how it has fared, it will be worthwhile to see how well these stand up with some of the best minds in the security industry focused on "breaking" them.

Client exploit evasions

Besides implementing several interesting evasions designed to bypass memory protection schemes, most client exploits use a myriad set of evasions designed to make the exploits essentially blind to technologies like traditional deep-packet-inspection devices on the network or application-hooking-based detection mechanisms on the endpoint. The use of scripting engines on the client software provides just enough abilities[9] for an average attacker to disguise the attack from a network inspection device. Some of the commonly used techniques by malware are payload obfuscation, algorithmic script code obfuscation, and code position obfuscation[10] to name a few. Most of these obfuscation techniques are readily available in underground exploit kits[11] which make wider exploitation just a matter of a few hundred dollars for an attacker. Some included well-known tricks like hook hopping[12] to evade host-based security software. Most of the underground exploit kits have the capabilities to test their attack payloads.

The exploitability and evasive nature of threats targeting the client have resulted in many of the emerging threats going

8 http://blogs.msdn.com/b/ie/archive/2012/03/12/enhanced-memory-protections-in-ie10.aspx.

9 http://metasploit.com.

10 http://securitylabs.websense.com/content/Blogs/3401.aspx.

11 http://contagiodump.blogspot.com/2010/06/overview-of-exploit-packs-update.html.

12 http://www.phrack.org/issues.html?issue=62&id=5&mode=txt.

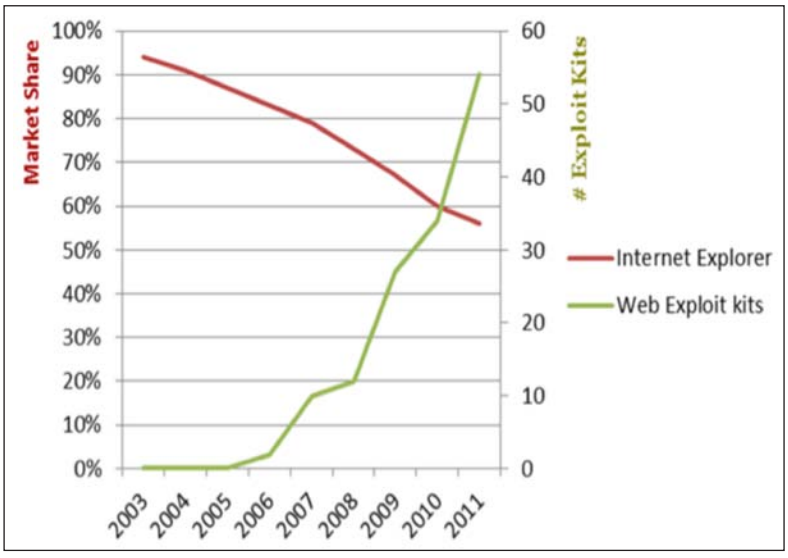

Figure 2 – IE market share vs. number of web exploit kits in the wild

undetected by traditional protection mechanisms – sometimes for months. The recent *Verizon 2012 Data Breach Investigations Report*[13] states that 85% of the breaches took more than two weeks to be discovered.

Does security impact the ubiquity of an application?

The more ubiquitous the application, the greater incentive for attackers to exploit and target the application, as it justifies their time and investment with higher returns in the underground malware world.

However, is it true that if an application is prone to too many vulnerabilities and malware, the user base will stop trusting it and move on to another application with similar capabilities? Figure 2 shows how Internet Explorer vulnerabilities grew over the years and how its user base has constantly declined in parallel with the large scale exploitation of browser vulnerabilities.[14]

Although admittedly it's probably not fair to only factor security, as there are several other factors like performance, functionality, and user experience that play an important role in public adoption of a new browser. There have been several high profile incidents in which governing authorities have expressed distrust of the browser due to security issues.[15]

Accurate calculation of mass exploitation of vulnerabilities is a non-trivial exercise, but we've attempted to extrapolate the decline of Internet Explorer market share and look at the growth of the underground exploit packs market that were used to launch mostly client-focused attacks on applications like Internet Explorer, Java, and Adobe Reader. The increase in use of these exploit kits provides an indicator that many of these vulnerabilities (most of which are exploiting old, unpatched software) are being actively exploited.

Remediation and countermeasures

Since memory protection schemes have not been reliable, alternate technologies like sandboxing came up several years back, which introduced the idea of exploit containment. But it has been well known that application sandboxes are not adequate as they rely on inherent OS primitives and are vulnerable to bugs in exposed kernel interfaces.

Newer emerging technologies like virtualization-based security, leveraging Intel VT-x, introduce a new approach by not focusing on exploit vectors or detecting exploits, but by providing security via isolating the attacks and leveraging the hardware (VT-based technologies) to mitigate risks and reduce the attack surface on the endpoint.

The limitations of each of the available technologies on the endpoint or network should be studied and well understood before deploying any layered approach solution. It is important to focus on non-signature-based approaches as long as these are implemented without generating false positives. It is highly recommended to have a dedicated security operations center (SOC) team in every enterprise with the sole mission to monitor and notify suspicious activity.

Conclusion

It's obvious that we have come a long way from SEH overwrites to the current state of the art for memory corruption exploits. Memory protection schemes provide significant gains to users by raising the bar for exploit writers.

The latest release of Windows 8 changes the entropy significantly for ***known*** exploitation vectors (kudos to Microsoft for being proactive this time around). However, history has proven many times that almost every time there's been a new memory protection scheme, it has been bypassed by a new exploit vector. All the known versions of Internet Explorer in the recent past (IE 6/7/8/9) were breached by exploits. As observed in the past, it has taken years to come up with robust memory protection schemes to counter the exploit vectors, and often times it's too late - as the adversary has already taken advantage of the vulnerability.

About the Author

Rahul Kashyap is Chief Security Architect and heads security research at Bromium Inc. Previously Rahul led the Worldwide Security Research team at McAfee Labs and has contributed on several preventive security technologies. Bromium is the pioneer of Trustworthy Computing technology designed to protect enterprises and users from sophisticated attacks. He can be reached at Rahul@bromium.com.

13 http://www.verizonbusiness.com/resources/reports/rp_data-breach-investigations-report-2012_en_xg.pdf.

14 Source: Wikipedia Internet Explorer Market share, Symantec, and http://contagiodump.blogspot.com/.

15 http://www.zdnet.com/germany-warns-users-to-ditch-internet-explorer-over-security-hole-7000004489/.