



X-Force Research & Development Newsletter

October 2006

“Heap Spraying” 101

If you've been following the current wave of Web browser vulnerabilities, you've probably heard the phrase “heap spraying” being touted. Within this short article we will explain what heap spraying actually is, when it is likely to be leveraged in an exploit, and what it means from a protection perspective.

For successful exploitation of most memory-corruption-based vulnerabilities, the attacker's ultimate goal is to redirect the running applications instruction pointer to the location of the attacker's newly uploaded shellcode payload. Stack-based overflows are typically used to overwrite the return address which then becomes the instruction pointer. Heap-based overflows are used to accomplish an arbitrary DWORD overwrite to point to a memory location that will ultimately become the instruction pointer (e.g. function pointers). However, a number of vulnerability scenarios may arise which do not allow for a simple overwrite of data that would eventually become the instruction pointer.

For example, in Microsoft's Internet Explorer createTextRange() vulnerability earlier this year (MS06-013), the vulnerability arose when Internet Explorer attempted to de-reference a ‘dangling’ function pointer – that is a function pointer that had not been correctly set to a valid location in memory. The value of the function pointer, however, seemed to be static and reliable within the application's memory. Repeated testing of the crash showed that Internet Explorer attempted to make a call to the exact same invalid place in memory every time. In essence, we encounter a scenario that is basically the opposite of what was described earlier for normal stack and heap-based overflows. Instead of controlling where the instruction pointer will go, the attacker needs to control where the shellcode payload will be in memory. If the attacker can force his shellcode to reside at that static memory location, when IE makes that invalid (and consequently vulnerable) call, it will be executing the attacker's payload.

So how could an attacker force his shellcode to be located at an arbitrary location in an application's memory? Unfortunately, when exploiting a Web browser, this is relatively trivial. The answer lies with JavaScript.

At the present time no modern Web browser sets limits as to the amount of memory that is allowed to be used by its JavaScript interpreter engine. This allows an attacker to create a giant array with JavaScript that contains repeated entries of a NOP sled and the attacker's shellcode. This array repeats itself enough times to fill up all unused memory on

the host to the point where the instruction pointer will eventually land on after the vulnerability is triggered.

See below for an example of what this would look like:

```
var shellcode = unescape("%uyadd%uayad%udaya...");
var nop_sled = unescape("%u9090%u9090");
while(nop_sled.length <= 40000)
    nop_sled += nop_sled;
var myArray = new Array();
for(var i=0; i<300; i++)
    myArray[i] = nop_sled+shellcode;
```

Interestingly, Web browsers will normally allow JavaScript-based applications to fill more than the 2GB of virtual memory that is commonly assigned to user space. Therefore, attackers have no problem getting their shellcode to any usable user space memory address.



When an attacker is attempting to write an exploit that needs to be portable across multiple versions of Microsoft Windows or various Web browser versions, simply hard-coding an address for the instruction pointer is also a bad idea. In many cases the return addresses and function pointers that can be overwritten need to be based upon an understanding of what specific version of vulnerable application (and operating system) the exploit is executing against.

Again in this case, if the attacker were able to intentionally overwrite a return address or function pointer with an arbitrary location in memory, he could point it to un-initialized heap memory. This would cause a forced simulation of the issue explained above (MS06-013). If the attacker could force his shellcode to exist at this arbitrary location, it would then be run regardless of operating system or Web browser version.

A third and final example of where heap spraying can be employed is when the attacker wishes to exploit applications running on Microsoft Windows XP SP2 and Windows Server 2003. A common technique is to overwrite the SEH pointer rather than a return address (the need for this is outside the scope of this article). Windows XP SP2 and Windows Server 2003 implement protection mechanisms to prevent pointing the SEH back into one of its own modules. However, both operating systems do not do anything to prevent pointing the SEH into heap memory. With heap spraying, the attacker can point the SEH to any arbitrary un-initialized memory, and force his shellcode there.

Heap spraying is just one of a number of weaknesses exposed and now exploited by JavaScript in modern-day Web browsers. Heap spraying allows attackers to exploit the previously un-exploitable, to write more reliable exploits, and to bypass protection mechanisms implemented by the base operating system. As long as JavaScript can be executed within Web browsers, hackers will continue to utilize it in their attacks to exploit new Web browser vulnerabilities.

Since the JavaScript code lying at the core of the Heap spraying exploitation technique is so specific and noisy (i.e. filling host memory up prior to actual exploitation), it is relatively easy to protect against. Consequently, ISS has had preemptive protection against the most common coding permutations for several months, and will be enhancing this class of protection in the months ahead. ISS will continue to stay several steps ahead of the attackers as they continue to obfuscate their code in the quest to evade legacy signature-based protection systems.

– David Dewey, X-Force Researcher

Shattering Records

Exponential Growth in Vulnerability Disclosure

For the entirety of 2005, X-Force uncovered, researched and recorded 5,195 new vulnerabilities. At 8:01 a.m. on September 25, 2006, X-Force researchers recorded the 5,196th vulnerability for 2006.

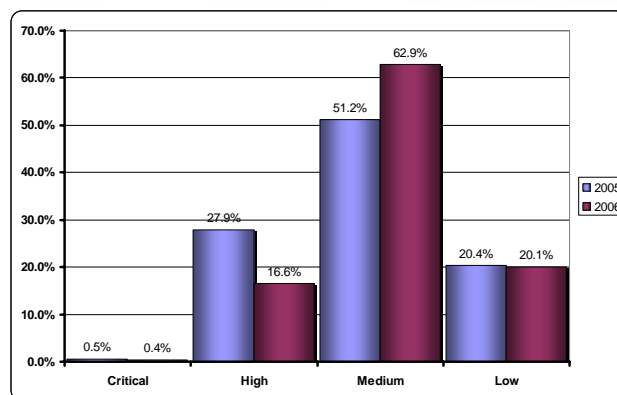
The default account vulnerability lying within the “FiWin SS28S WiFi VoIP SIP/Skype Phone” (<http://xforce.iss.net/xforce/xfdb/29114>) may have been somewhat boring as vulnerabilities go (and more than a little obscure), but it tipped the X-Force beyond last year’s ground-breaking record of vulnerabilities uncovered. As September came to a close, the total number of new vulnerabilities for the year reached an even 5,300.

Three-quarters of the way through 2006 and X-Force has already blown past last year’s total, with the busiest part of the year still ahead. Based upon vulnerability disclosure trends over the last decade, we expect to identify and research between 7,000 and 7,500 vulnerabilities by the end of the year.

Vulnerability Dynamics

Is there any good news for 2006? For the time being, yes, as the number of prolific and devastating network-based worms has decreased. While protection technologies have indeed advanced and most organizations have begun to take “defense in depth” seriously, a key contributor to worm reduction lies in the nature of the vulnerabilities being disclosed.

The following chart reviews 2005 and 2006 vulnerabilities (as of October 1, 2006):



Compared to the previous year, vulnerability risk ratings have shifted downward. The proportion of critical- and high-risk vulnerabilities dropped considerably (from 28.4 percent in 2005 to 17 percent in 2006). In turn, medium-risk vulnerabilities have increased.

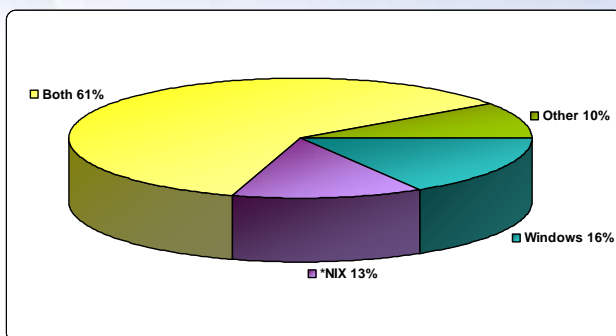
Does this mean that protecting an enterprise is getting easier? Not necessarily. Although critical- and high-risk vulnerabilities constitute a small wedge of the vulnerability pie, the pie is getting bigger and the absolute number of critical and high risk vulnerabilities is likely to remain unchanged over previous years.

Critical risk vulnerabilities (vulnerabilities that are network routable, do not require authentication and provide full administrative control over the host of a popular operating system) are the most important when creating devastating self-propagating worms. In 2005 there were 25 such vulnerabilities. This year there have been 19 to date with a quarter of the year remaining.

Understanding Risk Trends

It is unfortunate that the classical method of categorizing vulnerabilities uses the blunt term ‘risk’ (i.e. high risk, medium risk and low risk) because in reality we are actually ranking vulnerabilities based on their ‘impact after successful exploitation.’ The Common Vulnerability Scoring System (CVSS) will help this categorization process and make it easier for enterprises to manage threat responses to specific vulnerabilities – particularly when they include environmental factors such as operating system proliferation.

While there has been a continual marketing buzz around which operating systems are the most secure (refer to the July edition of the X-Force Newsletter for a comparison of Microsoft Windows XP with Mac OSX), it is actually pretty hard to discern.



For example, the pie-chart above reflects the breakdown of the 5,300 vulnerabilities X-Force recorded as of October 2006. Vulnerabilities that *only* affected the various Microsoft Windows OS platforms (e.g. Windows 95, XP, 2003, etc.) were 16 percent of the total, while those that *only* affected the various *NIX OS platforms (e.g. Linux, BSD, OSX, UNIX, etc.) made up 13 percent. Vulnerabilities that affected *both* OS platforms comprised 61 percent of the total (for example, a vulnerability in a popular PHP application designed to operate within Apache). Although the vulnerabilities affected the various OS platforms, this does not mean that the OS vendors were actually responsible for them – nor that security patches were released.

Identifying the specific OS with the most vulnerabilities is actually a complex endeavor since there is not a one-for-one mapping between vulnerabilities, advisories or operating systems.

For example, consider a single high-risk vulnerability in Apache:

The Apache development group may release an advisory about a vulnerability in some component of Apache, and indicating that the code fixes are available from a particular location. Every independent vendor that includes Apache with its operating system will then issue its own advisory about the vulnerability, how it affects its particular version and how to get the patches for the operating system (for an Apache vulnerability, this is likely to include more than 40 Linux OS vendors). At the same time, vendors that have embedded Apache into their products must also release advisories about the vulnerability (which could be more than 10 vendors). They must also list all the operating systems they've supply the embedded Apache to (such as Linux, Unix's and Microsoft). The vendors that have also ported Apache to the various Windows platforms will additionally list multiple Microsoft operating systems. However, there would not be an advisory from Microsoft itself because Microsoft didn't make the software and it isn't shipped by default with the OS.

In this scenario, the single Apache vulnerability can result in more than 60 individual advisory alerts. It

will also be listed as affecting more than 100 Linux operating systems (including major revision numbers), 20+ UNIX operating systems and approximately 15 Microsoft operating systems (including major revision numbers). In conclusion, if you were to count vulnerable operating systems based upon individual advisories, the tally would be 120 *NIX versus 15 for Microsoft.

A vulnerability in a Macromedia Flash plugin for Internet Explorer provides another example. One advisory would be published by Macromedia (the vendor) listing all Microsoft operating systems (perhaps 15 entries, including major service pack editions). Since the plugin is not designed to work on other browsers, no *NIX operating systems would be listed as affected.

As a final example, consider a vulnerability in Apple's QuickTime application. Apple would issue a single advisory listing all vulnerable operating systems – which would include approximately three Mac OSX versions and several Microsoft operating systems.

These examples hopefully shed light on the complex process X-Force researchers undertake to understand and correctly categorize each vulnerability.

Great Expectations

How can you protect the exponential growth of vulnerabilities?

A one-for-one mapping of vulnerability to 'signature' is not possible. Therefore, protection engines that rely upon regular expression signatures will be of limited value in the future. These outmoded protection engines can not cover every exploit example or vector for the 30,000+ vulnerabilities catalogued by X-Force. With 50,000 total vulnerabilities predicted by 2008, the outlook for signature-based protection is not good.

The secret to future protection lies in preemptive technologies which combat entire categories of vulnerabilities and the threats they represent. For example, the current top three vulnerability categories are Cross-site Scripting, SQL Injection and Buffer Overflows (14.5%, 10.9% and 10.8% respectively in 2006). Each category requires considerably different techniques to successfully exploit, and consequently requires different techniques in order to protect. However, once you have developed a protection technology for them, you are typically protected against all future vulnerabilities of that category. ISS has preemptively protected against these major vulnerability categories for several years already.

– Gunter Ollmann, Director of X-Force

A Month in Review

In this section of the newsletter, X-Force briefly covers some of the security content developed or processed in the month of September.

Vulnerabilities:

During September, X-Force observed yet another leap in new vulnerabilities during what is normally one of the quietest periods of the year. Compared to September 2005, X-Force researchers identified, catalogued and analyzed 51 percent more vulnerabilities.

Overall, thus far in 2006, X-Force has covered 41 percent more vulnerabilities over the same period in 2005.

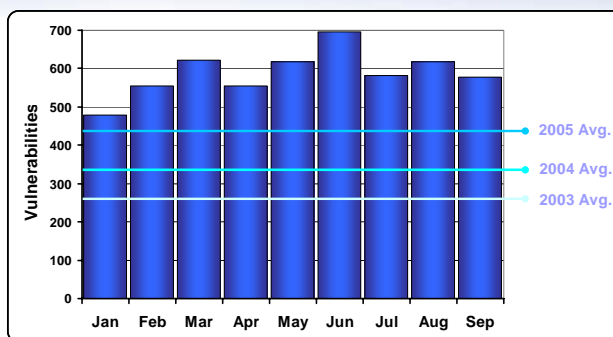
	Reported					YTD
	Crit.	High	Med.	Low	Total	
SEP	3	88	381	105	577	5300

Of these vulnerabilities, 520 were remotely exploitable, 62 could only be exploited locally and five could potentially be exploited both remotely or locally.

Vulnerability Breakdown	
Bypass Security	19
An attacker can bypass security restrictions such as a firewall or proxy, an IDS system or a virus scanner.	
Data Manipulation	77
An attacker is able to manipulate data stored or used by the host associated with the service or application.	
Denial of Service	50
An attacker can crash or hang a service or system, or take down a network.	
File Manipulation	9
An attacker can create, delete, read, modify or overwrite files.	
Gain Access	321
An attacker can obtain local and remote access. This also includes vulnerabilities in which an attacker can execute code or execute commands, because this usually allows the attacker to gain access to the system.	
Gain Privilege	21
An attacker can gain privileges on the local system only.	
Obtain Information	64
An attacker can obtain information such as file and path names, source code, passwords or server configuration details.	

* Right-hand column represents unique vulnerability count

Vulnerabilities per Month vs. Annual Average



Top 5 Vulnerable Vendors	
Microsoft	23
IBM	18
Apple	15
Sun	9
Linux Kernel, Cisco	8

* Right-hand column represents unique vulnerability count

Whiro:

The Whiro zero-day Web crawler project was kept busy last month identifying and classifying hundreds of new Web sites that utilize exploits to compromise vulnerable Web browsers and install malware or other malicious material.

For the month of October, Whiro repeatedly identified exploits embedded within Web sites that targeted several vulnerabilities affecting Microsoft Internet Explorer, including the two zero-day exploits (VML and WebViewFolderIcon). The following table lists these exploit attempts by prevalence:

Most Prevalent Web Exploits	
MS-ITS CHM file code execution (MS05-013)	*****
MS VML Vulnerability (MS06-055)	*****
InstallVersion.compareTo (MFSA2005-50)	*****
IE Media Player Plugin (MS06-006)	*****
RDS.Dataspace ActiveX (MS06-014)	****
WMP PNG Chunked Decoding (MS06-024)	****
IE WebViewFolderIcon ActiveX (MS06-056)	***
IE createTextRange() (MS06-013)	**
DHTML Objects (MS05-020)	**
IE daxctle.ocx Memory Corruption (CVE-2006-4777)	*
RealPlayer .SMIL (CVE-2005-0455)	*
IE ActiveX Memory Corruption (MS06-021)	*
JavaScript Navigator Object (MFSA2006-45)	*

The most popular exploit overall is still MS05-013, though in the later part of the month, we have observed a spike of MS06-055 exploits in the wild

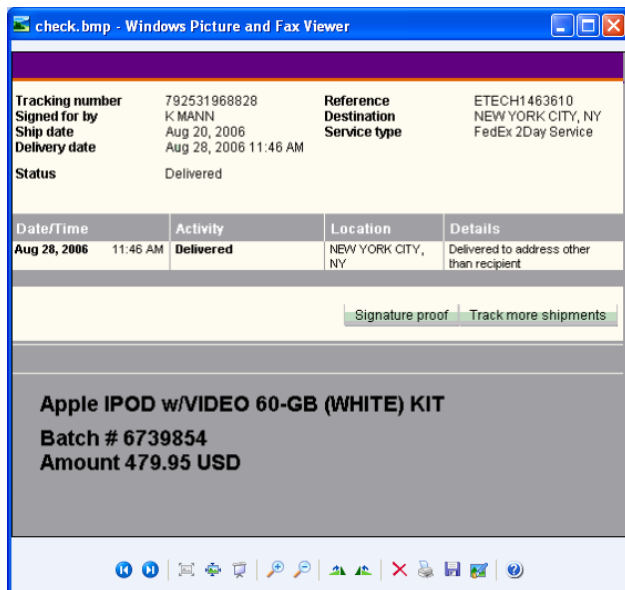
largely due to the inclusion of the vulnerability into the WebAttacker toolkit, also referred to as InetLux.

InetLux Latest Version	
MS-ITS CHM file code execution (MS05-013)	*****
MS VML Vulnerability (MS06-055)	*****
InstallVersion.compareTo (MFSA2005-50)	*****
IE Media Player Plugin (MS06-006)	*****
RDS.Dataspace ActiveX (MS06-014)	*****

Malcode:

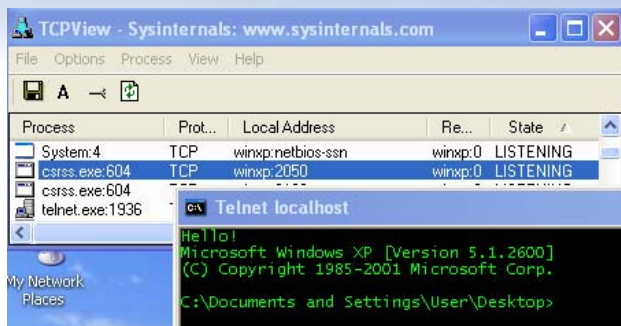
For the month of September, Catfish uniquely identified two zero-day Trojans and more than 46 variants of the Stration/Warezov worm.

On September 3, Catfish identified another new variant of the Goldun downloader (W32.Downloader.Goldun.DN). The said malcode installs a malicious BHO (Browser Helper Object) on the affected system. Once loaded by Internet Explorer, the BHO will then download and install a password-stealing Trojan on the affected system.



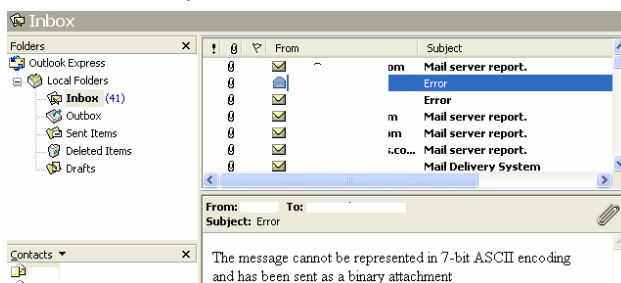
W32.Downloader.Goldun.DN – Check.BMP. After the malcode executable (CheckScan.exe) runs, it displays the image above as a way to hide its malicious intent.

The second Trojan, W32.PWS.LdPinch.OF, was identified on September 10, very similar to its earlier variant last August. This malcode attempts to steal the stored password of several applications (eg: Firefox, Trillian, AIM). The gathered information is then sent to a remote server via a PHP page. Additionally, it serves as a backdoor by starting an FTP server on TCP port 21 and a command shell on TCP port 2050.



W32.PWS.LdPinch.OF – Hello!. The malcode executable, csrss.exe (dropped in the %Windows% folder and disguised as a familiar Windows executable) opening a command shell on TCP port 2050.

Finally, On September 24 another wave of W32.Worm.Stration (aka Warezov) variants was unleashed. This time, 32 new variants were identified in a span of 10 hours.



W32.Worm.Stration – Spammed. Several emails generated by W32.Worm.Stration, each containing a copy of the worm.

Some AV vendors started to notice the Stration attack and raised the awareness level for this malcode. Similar to what happened last month, some of the major AV products are still failing to deliver the appropriate solution, requiring pattern/engine updates to detect the new variants. Again, the ISS Virus Prevention System was able to block the attack without the need for an update, continuously protecting ISS customers.

Sample Harvesting:

It is worth noting that as part of X-Force's continued strengthening of ISS antivirus, anti-spyware and anti-malware protection, we investigated and added another 44,274 new samples to our malcode zoo this month.

X-Force Security Content:

Level-0 XPU's:

In September, the X-Force team responded to four level-0 vulnerabilities and their public exploits, and we remained *Ahead of the threat*.

New and Extended Parser Support:

During October, X-Force engineers added one new protocol parser and extended four existing PAM

protocol parsers. This work included the following protocols:

- Biff
- Java script
- HTML
- MSRPC
- HTTP
- Compound_File

New Vulnerability Protection Algorithms:

X-Force added the following security coverage to ISS products, with 21 new events to address new vulnerabilities:

- JavaScript_DirectAnimation_Overflow
- HTML_VML_Overflow
- MSRPC_Invalid_Request
- JavaScript_WebViewFolderIcon_Overflow
- UDP_Ares_Galaxy
- DPS_String_Overflow
- DPS_IpAddr_Overflow
- DPS_Magic_Number_DoS
- CompoundFile_Excel_Style_BO
- HTTP_Apache_Expect_XSS
- HTML_Asp_Dot_Net_XSS

- CompoundFile_Word_Malformed_Stack
- CompoundFile_Improper_Memory_Access
- CompoundFile_Word_Malformed_String
- CompoundFile_Word_SmartTag_BO
- CompoundFile_Word_Table_CodeExec
- CompoundFile_PowerPoint_TextByte_BO
- CompoundFile_PowerPoint_TableProp_Overflow
- CompoundFile_Excel_DateTime_BO
- BIFF_Excel_Lotus123_CodeExec
- HTTP_Microsoft_Error_Report

New Default Blocking Algorithms:

X-Force added the following default blocking to ISS products:

- JavaScript_DirectAnimation_Overflow
- JavaScript_WebViewFolderIcon_Overflow
- HTML_VML_Overflow

Copyright© 2006 Internet Security Systems, Inc. All rights reserved worldwide.

Internet Security Systems, the Internet Security Systems logo, Proventia, the Proventia logo, SiteProtector and Ahead of the Threat are trademarks or registered trademarks of Internet Security Systems, Inc. Other marks and trade names mentioned are the property of their owners, as indicated. All marks are the property of their respective owner and used in an editorial context without intent of infringement. Specifications and content are subject to change without notice.