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
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.
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 **integer overflow** vulnerability
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 double free vulnerability
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

```c
// Initialize the heap library
VAR heap = new heaplib.ie();

// Messagebox shellcode
VAR shellcode = unescape('"
"%u434%u434%u54eb%u7b3c%u357%u0378 " +
"%5f5%u76b%u032%u3f%u49c%u4d4%u0db33 " +
"%0f3%u1be%u382%u74f%u2c10%u0dcb%u0a03 " +
"%e4%u38f%u575%ue1e7%u5e8b%u032%u6d6d " +
"%0c8b%u8b4%u1c5%u0dd0%u048b%u038b%u3c35 " +
"%72%5%606%e6%6f%64%e6%c6%c4%00%u3a3 " +
"%e2e5%7%86%u065%uc33%u036%u304%u0c78 " +
"%40%b8%80%cu1c7%u88ad%u0840%u09eb%u408b " +
"%8d3%u7c4%u0b5%u93%uebf%u0e4%ue8ec " +
"%ff84%fff%ec83%u830%u24c%u2ff3%u95d0 " +
"%bf5%u1a3%u72%u6e8%u2ff%u86%u2454 " +
"%d%u52%u35%u35%eb%u52%u532%u0df " +
"%bf%u98%u8%u53%e8%u2ff%u83%u04ec " +
"%2c%e2%u0d0%u7eb%u2d8%u873%u2ff4 " +
"%ff%u2f2%ue8%uffd%u0ff7%ufff";_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 looksaside that will not to the vtable
VAR fakeObjPtr = heap.lookasideAdd (vtable);

// Build the heap block with the fake object address
// len padding fake obj to point padding null
// 4 bytes 0x20c-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

Behavior
- Sandbox
- Honeyclients

Static
- Signatures
- Heuristics
- Characteristics

Reputation
- Source
- Links
- Neighbors
- Owner

Other
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
Method 2

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

1. Fresh Machine State
2. Launch Single Browser Process
3. Visit URL (Allow All State Modifications)
4. Compare Modified State to Original State; Determine if State Changes Were Unauthorized
5. Malicious, Suspicious, or Benign?
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!

- Launch Multiple Browser Processes
- Malicious or Benign?
- Visit URLs (Disallow All State Modifications)
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