Heap Taichi: 
Exploiting Memory Allocation Granularity in Heap-Spraying Attacks

Yu Ding¹, Tao Wei¹, Tielei Wang¹, Zhenkai Liang², Wei Zou¹
¹Peking University, ²National University of Singapore
A Typical Heap-Spraying Attack

```javascript
var cc = unescape("%u0C0C");
var sc = unescape("%u785C%u3334...");
while (cc.length * 2 < 0x100000)
{
    cc+=cc;
cc=cc.substring(sc.length + 0x38, 0x100000/2);
}
var m = new Array();
for(i = 0; i < 200; i++)
{
    m[i] = cc + sc;
}
```
Existing Solutions

Measure how much of the heap can be used as nop-sledge

Because of randomization (ASLR), the large amount of nop instructions is required.

A direct detection method is to measure how much of the heap can be used as nop-sledge and flows to a common sink.

Surface Area  \( \text{sizeof(heap data that can be used as nop-sledge)} \)

Normalized Surface Area = Surface Area / sizeof (all heap data)

\( \approx \)

If the NSA > threshold, generate an alarm.

But how reliable the randomization is?

Nozzle [USENIX'09]
The memory allocation is much more predictable than expected.
· E.g. address 0c0c0c0c can only be 8 possible positions in a 512kbytes block
Proof of Concept: Heap Taichi

- Memory allocation granularity makes defense mechanisms less flexible
- We discuss a few transformations to reduce heap-spraying attacks’ footprint without loosing its …
- Countering force with agility 以柔克刚
Basic Structure of TypeA (Passing Flower)
alignment size = 64Kbytes
blocksize = 512Kbytes

NSA(sledge) ≈ 7 / 512K ≈ 0%
The Structure of Type A Attack (Detail)

- A sample JavaScript code creating Type A heap objects

```
Jump to next landing point

Type1_64k: 0xCC 0xCC...0xCC 0xE9 0xFB 0xFF 0x00 0x00 0xCC
           0x0 3084 bytes 0x0C 6 bytes 0xC12 62446 bytes 0xffff

Type2_64k: 0xCC 0xCC...0xCC shellcode 0xCC 0xCC...0xCC
           0x0 3084 bytes 0x0C 6 bytes 0xffff

Final block: Type1_64k Type1_64k ... Type1_64k Type2_64k
```

- The size of each heap memory block is 1Mbytes.
- Try to allocate 200 heap memory blocks to cover address 0x0c0c0c0c0c
Basic Structure of TypeB (Jumping together)
alignment size = 64Kbytes
blocksize = 512Kbytes

NSA(sledge) ≈ 7 / 512K ≈ 0%
Basic Structure of TypeC (Returning Home)
alignment size = 64Kbytes
blocksize = 512Kbytes

NSA ≈ sizeof(shellcode) / 512K ≈ 0%
A Mix Type of TypeA, B, C

What if the shellcode is larger than the alignment size?
The TypeD Attack
(alignment size = 32bytes)

sc piece  sc piece  sc piece  ......  sc piece

To a shellcode entry

5 bytes

header

shellcode sc piece

less or equal to 25 bytes

AVAILABLE LANDING POSITIONS

To the next sc piece

2bytes

tail

a linked shellcode
The TypeD Attack
(alignment size = 32bytes)
More complicated jumps
Relationship between Attack Type and Alignment Size

**Summarize**

<table>
<thead>
<tr>
<th>Alignment size</th>
<th>Type A</th>
<th>Type B</th>
<th>Type C</th>
<th>Type D</th>
</tr>
</thead>
<tbody>
<tr>
<td>64 kbytes</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>32 bytes</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>8 bytes</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>4 bytes</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

- Small amount of instructions keep high success rate with help of memory allocation granularity.
Evaluation of Heap Taichi Attack
Surface Area Calculation

Normalized Surface Area

Threshold proposed by Nozzle (50%)
No false positive threshold of Nozzle (20%)
Max top 150 Alexa sites (12%)

B8  A8  A32  B32  A4k  B4k  D32  A65k B64k  C4k  C64k

12/6/2010
Detecting Heap Taichi Attacks
## Detecting Heap Taichi

<table>
<thead>
<tr>
<th>Alignment size</th>
<th>Type A</th>
<th>Type B</th>
<th>Type C</th>
<th>Type D</th>
</tr>
</thead>
<tbody>
<tr>
<td>64 kbytes</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>32 bytes</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>8 bytes</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

1. Decrease the alignment size.
2. Improve the normalized surface area calculation.

Coarse memory allocation granularity helps hide the shellcode and jump instructions.

Nozzle algorithm can be evaded using multiple copies of shellcode.
Our Approach: We modified JSarena and Firefox to support 8-byte alignment size.

Alignment size = 2Mbytes

Alignment size = 8 bytes?
8-byte alignment is feasible

- The performance overhead is less than 5%

**Sunspider Benchmark**

- Firefox
- Modified Firefox

**V8 Benchmark**

- Firefox
- Modified Firefox
Enhance Nozzle
(Alignment size = 8 bytes)
Enhance Nozzle
(Alignment size = 8 bytes)

On offset 0x2

Success rate on offset 0x2
= 1/3 = 33%

On offset 0x5
RELATED WORK

- Shellcode-based detection
  - Egele et, al. DIMVA'09
  - Sam Small et, al. CCS'09
  - Yingbo Song et, al. CCS'07

- Heap spraying with ASLR
  - The granularity is still 64KBytes

- Heap spraying with DEP
  - DEP can be bypassed
    - Peter Vreugdenhil, Pwn2Own

- Solid Heap Memory Allocator
  - DieHard, DieHarder

- Memory exploit detection and prevention
Conclusion

- We analyze the weakness of memory alignment size.

- We present Heap Taichi, a new heap-spraying skill utilizing the weakness of memory alignments.

- We present an enhanced nozzle algorithm. The new algorithm can recognize these Heap Taichi attacks cooperating with finer memory allocation granularity.
Thank you!