FIT5124 Advanced Topics in Security

Hacking Techniques III – Web Browser Exploitation

Ron Steinfeld
Clayton School of IT
Monash University

April 2015

Hacking Techniques III

Web Browser Exploitation:

Web browsers are most often used application today – attractive target for hackers.
Today: A look at some known browser exploitation techniques.

Plan for this lecture: Exploitation techniques, examples, and defenses for:

- Heap Overflow Exploit techniques:
  - 'Heap Spray' technique
  - 'Heap Engineering' technique
  - 'Use After Free' technique
  - Example defenses

Heap Overflow Exploits

Recall: **Heap** is a segment in machine memory space

- Used to store dynamically allocated variables
- Managed by an OS heap allocation manager (called by the Browser via malloc system calls).
- Heap space allocated by OS to browser managed by a browser heap manager (e.g. storage of current web page HTML objects).

Observations about heap:

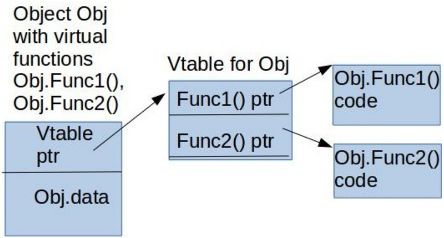
- Browser's Javascript engine creates objects on heap for Javascript program objects, e.g.
  - Javascript string objects,
  - Javascript 'ActiveX' objects

Heap Overflow Exploits

Observations about heap (cont.):

- Objects (e.g. 'ActiveX' objects) often include virtual functions.
  - Virtual functions – implementation can be overwritten by a subclass inheriting from parent class
  - Hence, address of virtual function implmentation not known at compile time
  - Implemented as a vtable: virtual function = ptr to address of implementation (ptr set at compile time).

Example:



Heap Overflow Exploits

Suppose: browser heap manager contains vulnerability

- e.g. buffer overflow into ActiveX object's virtual function vtable

Possible Exploit:

- Attacker's Javascript can write strings containing malicious code into heap.
- Attacker uses overflow vulnerability to overwrite vtable pointer to point to malicious code.

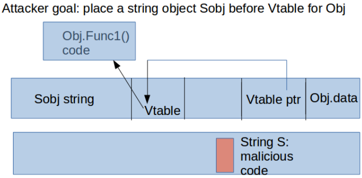


Figure : Before Overflow

Heap Overflow Exploits

Suppose: browser heap manager contains vulnerability

- e.g. buffer overflow into ActiveX object's virtual function vtable

Possible Exploit:

- Attacker's Javascript can write strings containing malicious code into heap.
- Attacker uses overflow vulnerability to overwrite vtable pointer to point to malicious code.

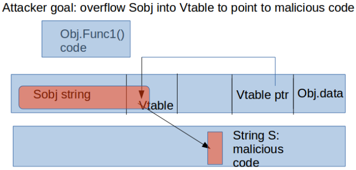
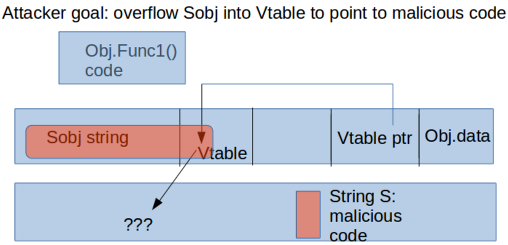


Figure : After Overflow

Heap Overflow Exploits

But... malicious code strings could be allocated anywhere in heap (attacker doesn't know where!).

Q: Where should attacker point his overflow vtable pointer?



Attacker unlikely to guess correctly location of string S....

Possible A: 'Heap Spray' technique!

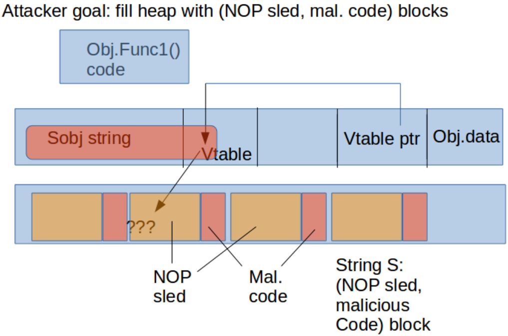
Heap Overflow Exploits: Heap Spray Technique

Idea:

- Fill large fraction of heap with NOP sleds leading to malicious code.
- Set overflowed vtable ptr anywhere in heap

Goal: High probability that vtable ptr points somewhere in (one of) NOP sleds!

- Hence: much higher probability of attack success!



Heap Overflow Exploits: Heap Spray Technique

Possible Heap Spray Implementation in Javascript [HFS07]:

```
var nop = unescape("%u9090%u9090");

// Create a 1MB string of NOP instructions followed by shellcode:
//
// malloc header   string length   NOP slide   shellcode   NULL terminator
// 32 bytes        4 bytes         x bytes     y bytes     2 bytes

while (nop.length <= 0x100000/2) nop += nop;

nop = nop.substring(0, 0x100000/2 - 32/2 - 4/2 - shellcode.length - 2/2);

var x = new Array();

// Fill 200MB of memory with copies of the NOP slide and shellcode
for (var i = 0; i < 200; i++) {
    x[i] = nop + shellcode;
}
```

Heap Overflow Exploits: Heap Engineering

Q: How does attacker place overflowed buffer object next to target object Obj?:

Possible A: Heap Engineering [HPS07,DHM08]

- Defragment the heap (‘plug’ the ‘holes’) with overflowing (vulnerable) objects.
- Create regular holes between overflowing objects.
- Insert target object into regular holes.

Hence, target object will likely be next to an overflowing object!

Figure 3: Defragmented heap with many allocations. We see a long line of same-sized buffers that we control.

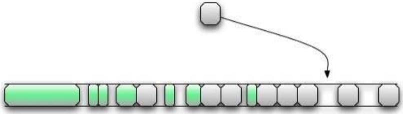


Figure 4: Controlled heap with every other buffer freed. The allocation of the vulnerable buffer ends up in one of the holes.

Heap Overflow Exploits: Heap Engineering

Heap Eng. Implementation in Javascript [DHM08]:

Step 1: Defragmenting the heap

```
var bigdummy = new Array(1000);
for(i=0; i<1000; i++){

     bigdummy[i] = new Array(size);
  }
```

Step 2: Create regular holes between overflowing objects.

```
for(i=900; i<1000; i+=2){
   delete(bigdummy[i]);
}
```

```
for(i=0; i<4100; i++){
  a = .5;
}
```

Step 3: Insert target object into regular holes.

```
for(i=901; i<1000; i+=2){
   bigdummy[i][0] = new Number(i);
}
```

Heap Overflow Exploits: Heap Engineering

Heap Eng. Implementation in Javascript [DHM08] (cont):

After step 3, have the following situation:

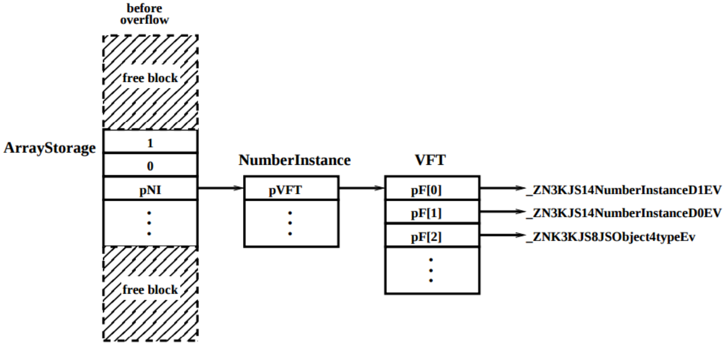


Figure 5: Details of an attacker controlled block just before the overflow is triggered.

Heap Eng. Implementation in Javascript [DHM08] (cont):

Then heap is sprayed and overflow is triggered, and we get:

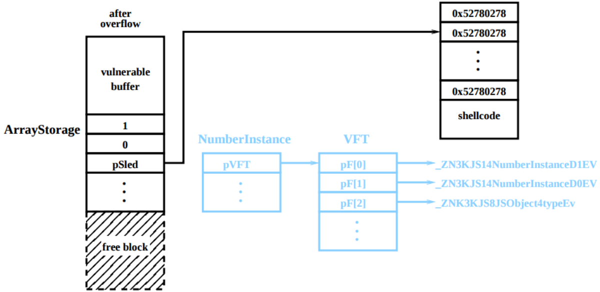


Figure 6: Details of an attacker controlled block just after the overflow is triggered.

Practical Difficulty: overflow into vtable ptr, not vtable itself!

- Double indirection - execution jumps to *(* psled)!!

Q: How to spray?

Heap Overflow Exploits: Heap Engineering

Heap Eng. Implementation in Javascript [DHM08] (cont):

Possible A: Spray heap with 'NOP sled' dword value 0x52780278

How does it work?

- First indirection: (* psled) points to 'magic NOP sled' with high probability (spray trick).
  - Points to address 0x52780278 - hope that this falls in spray area too (self-referential).
- Second indirection: *(* psled) points back into to 'magic NOP sled' with high probability.
  - Opcode meaning of dword 0x52780278:

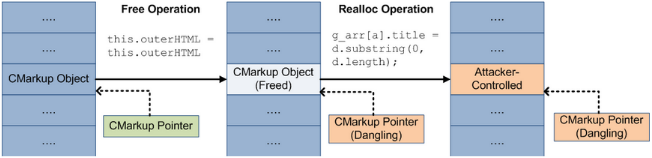
```
78 02:   js +0x2
78 52:   js +0x52
```

  - Whether first condition is true or false, will always jump to next dword (+2 words)!

Heap Overflow Exploits: Use After Free (UAF) Vulnerabilities

In 2012-2014, several heap vulnerabilities were discovered and exploited in the field in Microsoft IE [Yas13,Yas14].
They fall into the class of 'Use After Free' (UAF) vulnerabilities
UAF Vulnerability: Browser code frees heap allocation for object, but later dereference the freed object!

- Exploit: Attacker trigger the free, then reallocates the freed memory to object containing attacker malicious code!



Heap Overflow Exploits: Defenses

Several proposed (partially effective) countermeasures:

- Browser heap isolation (e.g. Microsoft's 2014 'Isolated Heap').
    - Use a separate heap for string objects as for other (e.g. ActiveX) objects (heap eng. technique with strings not possible, some UAF exploits prevented).
- Randomized heap allocation (against heap eng.)
- Heap overflow protection using non-writable pages between writable ones (e.g. FreeBSD)
- Nozzle (Microsoft) [RLZ09]: Detect a lot of code in the heap (detect spraying).

Counter-countermeasures by hackers are being devised, e.g. UAF exploit against IE's 'Isolated Heap' [D15]

References referred to in the Slides

DHM08 M. Daniel, J. Honorrof, C. Miller, Engineering Heap Overflow Exploits with Javascript, In Proceedings of Usenix WOOT 2008.

HFS07 A. Sotirov, Heap Feng Shui in Javascript, In Proceedings of Blackhat Europe 2007.

RLZ09 P. Ratanaworabhan and B. Livshits and B. Zorn, Nozzle: A defense against Heap-Spraying Code Injection Attacks, In Proceedings of Usenix Security 2009.

Yas13 M. Yason, Use-after-frees: That pointer may be pointing to something bad, Security Intelligence, April 2013. Available at http://securityintelligence.com/use-after-frees-that-pointer-may-be-pointing-to-something-bad

Yas14 M. Yason, Understanding IEs New Exploit Mitigations: The Memory Protector and the Isolated Heap, Security Intelligence, August 2014. http://securityintelligence.com/understanding-ies-new-exploit-mitigations-the-memory-protector-and-the-isolated-heap

D15 J. DeMott, Use-after-Free: New Protections, and how to Defeat them, Bromium Labs Call of the Wild Blog, Jan. 2015. http://labs.bromium.com/2015/01/17/use-after-free-new-protections-and-how-to-defeat-them/