Monthly Research

History and Current State of Heap Exploit

FFRI, Inc
http://www.ffri.jp

Ver 2.00.01

What is Heap Exploit?

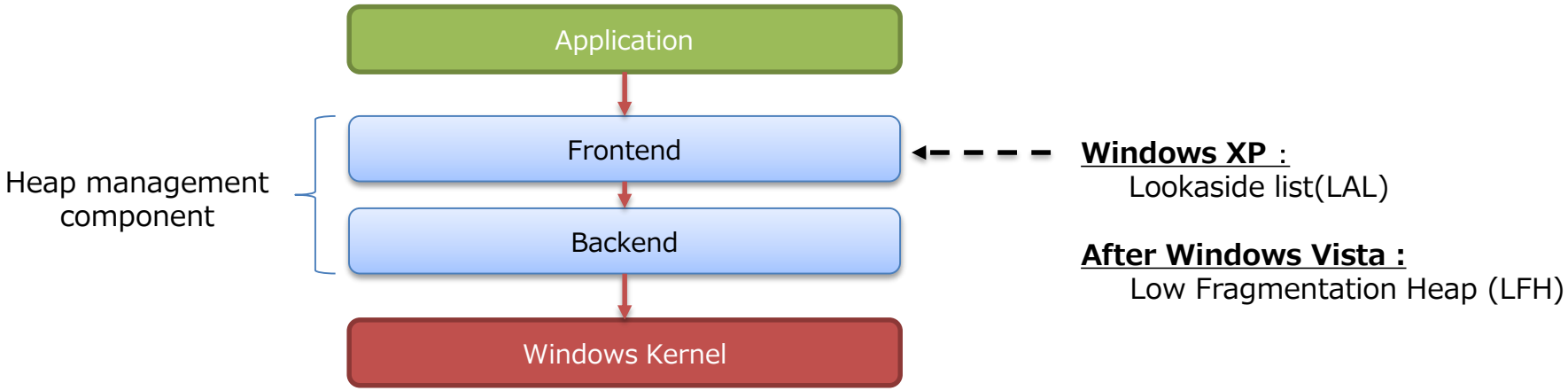
- Heap exploit is to attack a software problem of managing heap memory region.
- There are mainly 2 categories for memory corruption vulnerabilities.
  - Memory corruption on stack
  - Memory corruption on heap
- Main causes of them are “buffer overflow”
- This slides summarizes exploitation methods and mitigations so far for heap overflow.
- The target is userland heap on Windows XP and later.
- There are many exploitation techniques corresponding to structures and algorithms in heap, hence this slides summarize basic idea of them and mitigations.
- Note that the each explanation in this slide is a lot simplified to make it easy to understand main idea of heap exploit.

What is Heap?

- Heap is region of memory dynamically allocated during executing a code.
- Allocated memory area via malloc or new in C/C++
- The implementations of malloc and new are different among platforms
- In Windows mainly following APIs are used to operate on heap.
  - HeapCreate　　Create a heap
  - HeapAlloc　　Obtain specific size of memory from heap region
  - HeapFree　　Release obtained memory region

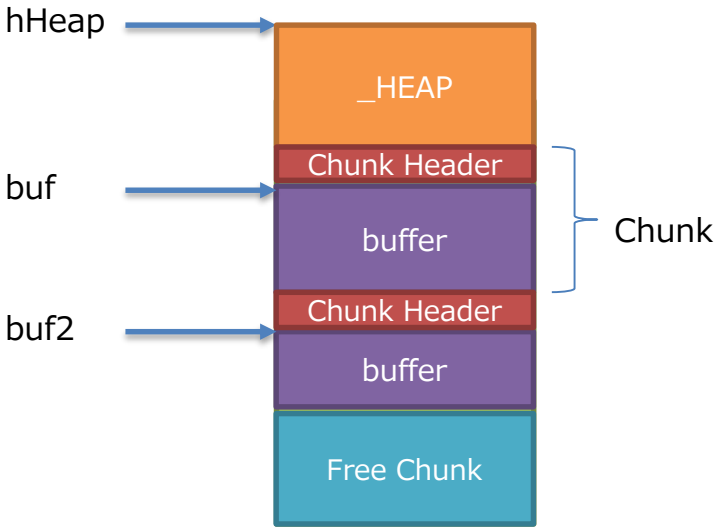
Overview of Windows Heap management components

- Windows heap manager consist of mainly following 2 components.
  - Frontend
  - Backend
- Frontend is an interface to an application
  - Optimizes allocating small memory blocks
  - If it is able to respond to a request, it returns a memory block
  - If not, pass the request to the backend.
  - There are 2 frontend implementations. Lookaside List(LAL) on Windows XP and Low Fragmentation Heap on Vista or later.



Basics of Heap API

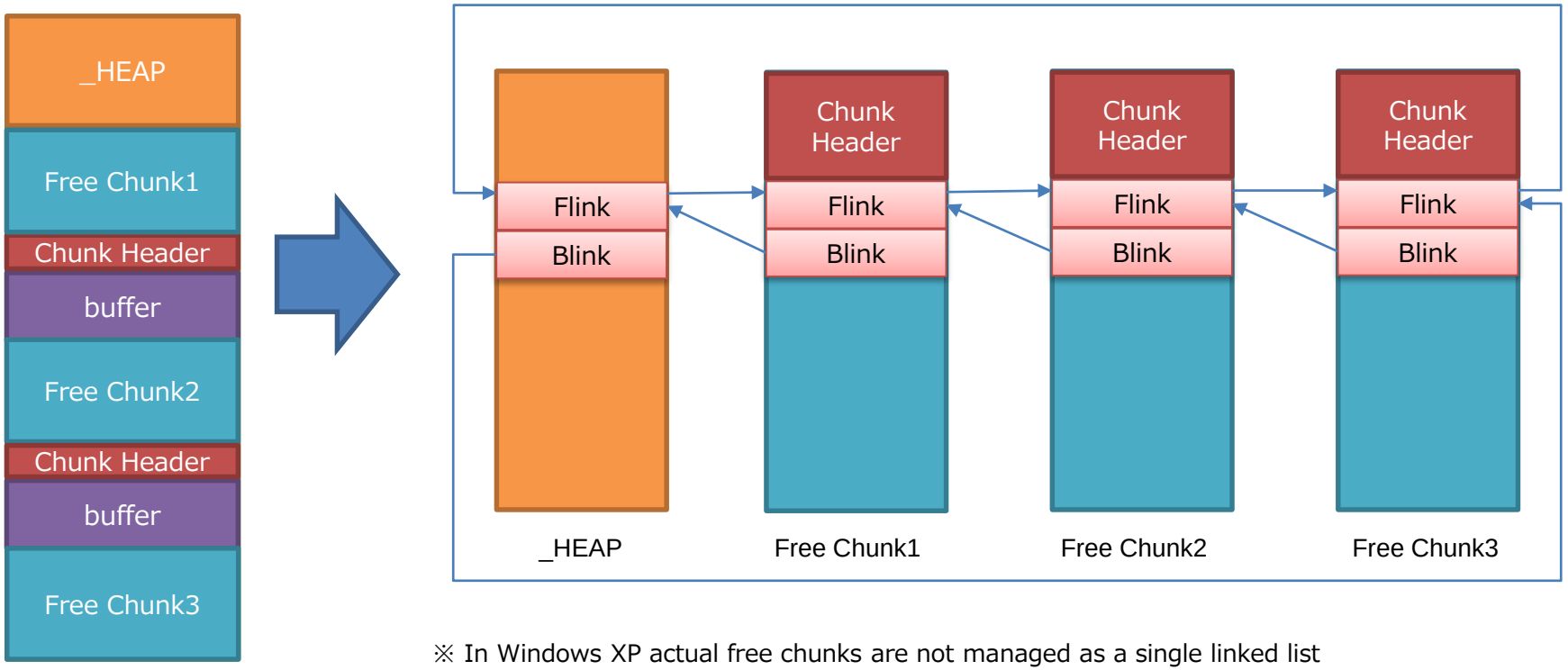
- Heap is created by HeapCreate.
- _HEAP structure resides at the beginning of a heap and the address is returned as a HANDLE.
- _HEAP structure contains information to manage the heap.
- By passing a HANDLE and size to HeapAlloc it returns the requested size of memory.
- Applications can save data into the returned memory region.
- To manage heap there is 2 bytes management information(Chunk header) just before allocated memory.
- Regions not allocated yet are managed as free chunks




```
HANDLE hHeap = HeapCreate(...);
LPVOID buf   = HeapAlloc(hHeap, ... );
LPVOID buf2  = HeapAlloc(hHeap, ... );
```

Managing Free Chunks(Backend)

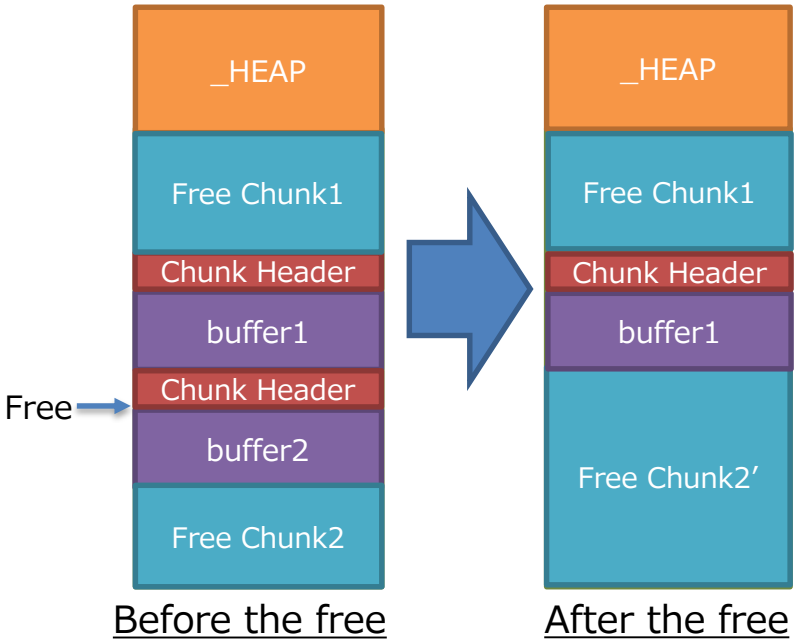
- As an application calling HeapAlloc or HeapFree in variety of order allocated chunks and free chunks are fragmented.
- To manage free chunks Windows heap manager uses doubly cyclic linked list.
- Free chunks also have a chunk header



※ In Windows XP actual free chunks are not managed as a single linked list but as a multiple linked lists. It is simplified here for the purpose of explanation of the exploit.

HeapFree and Coalesce

- When an allocated chunk is freed, and if the next chunk is also free these chunks are coalesced
- The exploit explained next slide utilizes the process of unlinking of an element from a doubly linked list.
- In figure below, when “buffer2” is freed coalescing happens.



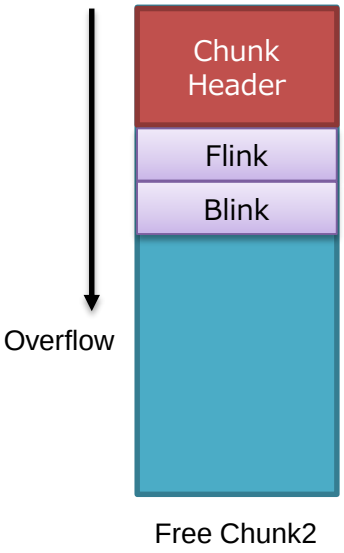
Before freeing “Buffer2”, “Free Chunk1” and “Free Chunk2” is linked as linked list.
When “buffer2” is freed “buffer2” and “Free Chunk2” make bigger free chunks(Free Chunk2’) and “Buffer2” and the “Free Chunk2’” will be linked.
In this process, “Free Chunk2” is removed from the linked list by the code like following.

```
// Remove “Free chunk2” from linked list
[Free Chunk2]->Blink->Flink = [Free Chunk2]->Flink
[Free Chunk2]->Flink->Blink = [Free Chunk2]->Blink
```

* This process happens only when the frontend(Lookaside list) is not used and the backend is used for process of freeing. To make use of this process for actual attack one must make such conditions.

Basic Heap Exploit

- Heap exploitation feasible up to Windows XP SP1
- By rewriting Flink and Blink of a free chunk, writing arbitrary 4byte into arbitrary address.
- Assume that the head of “Free Chunk2” in the previous slide is rewritten by overflow.
- On coalescing, the values of Flink and Blink are referenced and the values are written into the arbitrary addresses.



// [Free Chunk2]->Flink and [Free Chunk2]->Blink is rewritten.
[Free Chunk2]->Blink->Flink = [Free Chunk2]->Flink
[Free Chunk2]->Flink->Blink = [Free Chunk2]->Blink

Write an arbitrary value into an arbitrary address

By overwriting a function ptr with an arbitrary address, arbitrary code can be executed.
→ Using a function ptr in PEB (Process Environment Block) is one of the well know methods.
→ 2 lines of the code above are executed at the same time, attempting overwriting a function pointer automatically writes the address of the function pointer itself(e.g. 0x7FFDF01C) into the head of the address where the function ptr points to. To make the attack successful the value(0x7FFDF01C) must be able to be executed by CPU.

Mitigations in Windows XP SP2

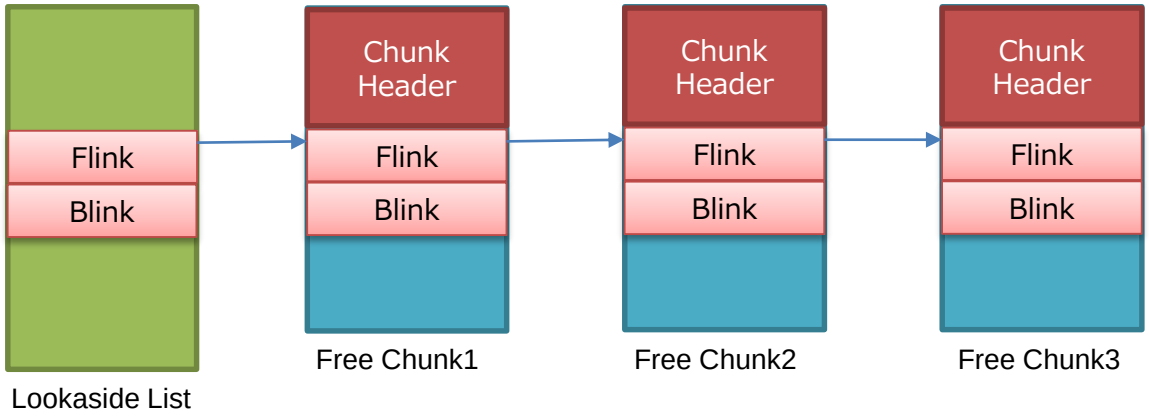
- Mainly following 3 mitigation are introduced into Windows XP SP2
- Cookie in chunk header
  - 8bit checksum(cookie) is introduced in chunk header.
  - By validating its value it can detect overwrite of a chunk header.
  - The value of a cookie is based on the address of the chunk.
- Safe unlinking
  - Before removing an element from doubly linked list it checks if following condition is met.

    `[Chunk]->Flink->Blink == [Chunk]->Blink->Flink == [Chunk]`

    (Confirming if next/prev elements also point to the element)
- PEB Randomization
  - Randomize the address of PEB(Process Environment Block).
  - PEB contains values and addresses used by attackers not only in heap exploit.
  - This randomization makes the success rate of attacks low.

Bypassing Windows XP SP2 mitigations

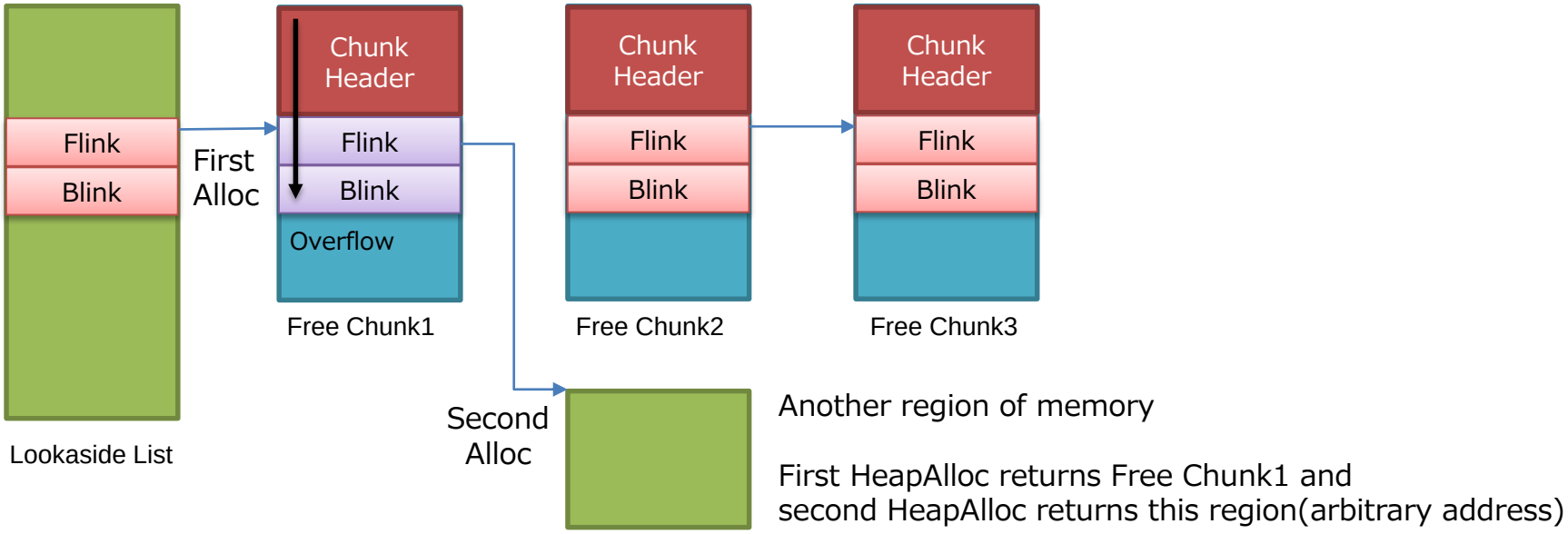
- Using lookaside list
  - Freeing a chunk via HeapFree is handled by lookaside list first
  - Lookaside lists are singly linked lists of free chunks in each sizes.
  - If memory chunk is requested via HeapAlloc, it first checks if there is a free chunk in lookaside list and returns it if one exists.



- Lookaside list is multiple singly linked list for each size of chunks.
- In one singly linked list the same size of chunks are linked.
- (Figure above depicts one of the lists)
- 4 chunks at maximum in one list.
- (If more chunks of the same size are freed, it is handled by the backend)
- Adding/Removing a chunk is done on first entry of a list.

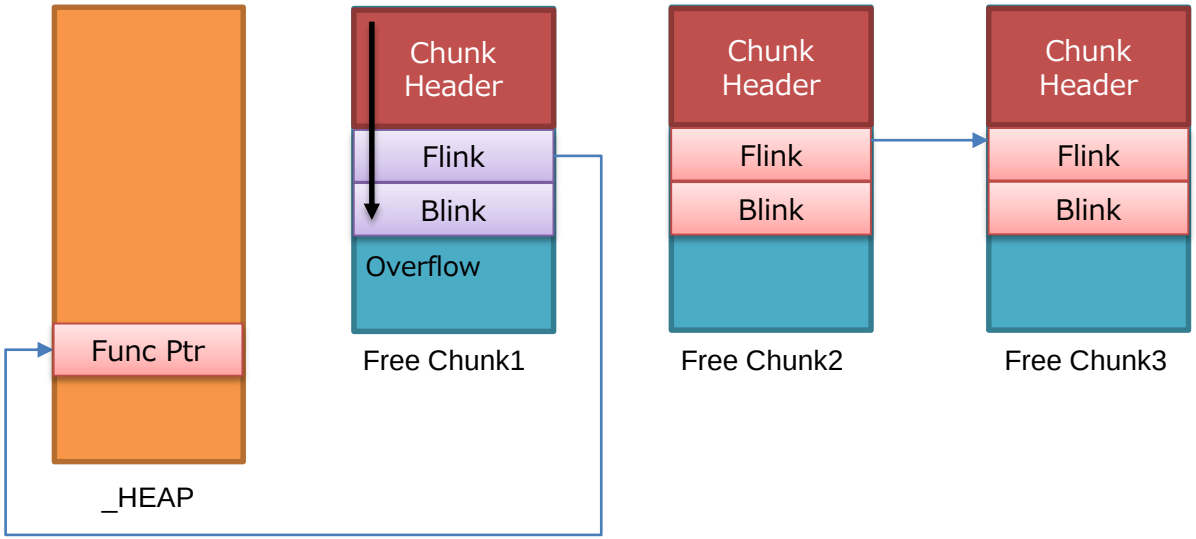
Bypassing Windows XP SP2 mitigations(Cont.)

- Using lookaside list
  - Important 2 mitigations does not work on lookaside list
    - Allocating a chunk from lookaside list does not make use of cookie
    - Safe Unlinking is not done (because it is not doubly linked list)
  - Assume in following figure header region of “Free Chunk1” is overwritten.
  - Then subsequent 2 calls of HeapAlloc allocates a chunk from an arbitrary address specified by the overflow



Bypassing Windows XP SP2 mitigations(Cont.)

- Using lookaside list
  - Overflow to make Flink have an address of a region containing a function pointer
  - If data written in the region allocated from subsequent second HeapAlloc can be controlled, the pointer can be rewritten by an arbitrary address.
  - In the figure below, Flink uses a function pointer in _HEAP structure (This function pointer is used and called in heap management process)



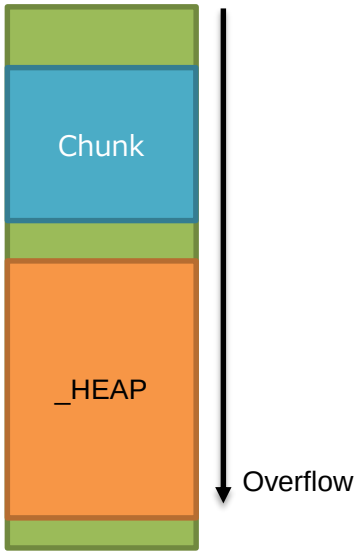
- There are other tactics for exploitation for mitigation in XP SP2.

Mitigation introduced in Windows Vista

- Low Fragmentation Heap
  - Lookaside list is replaced with Low Fragmentation Heap
  - Impossible to attack using lookaside list
- Randomizing Block Metadata
  - Chunk header is xored with _HEAP->Encoding
  - Overwriting chunk header with predicted cookie still results in unexpected state of the chunk header.
- Enhanced entry header cookie
  - Cookie value also checks values in chunk header
  - Cookie had been calculated based on the chunk address but now it is calculated/validated with values in a chunk header.
- Heap base randomization
  - The base address of _HEAP structure is randomized
  - Makes it difficult to overwrite data in _HEAP structure
- Heap function pointer encoding
  - A function pointer in _HEAP structure is xored with a value
  - Mitigations for rewriting a function pointer in _HEAP structure

Bypassing mitigations introduced in Windows Vista

- Overwriting _HEAP structure
  - Overwrite _HEAP structure using heap overflow
  - Overwriting a function pointer in _HEAP structure makes it possible to execute arbitrary address
  - But, several conditions must be met
    - Chunk to be overflowed must be located lower address of _HEAP structure
    - The address of chunk to be overflowed is predictable
    - The size of overflow is enough big



A function pointer in _HEAP structure is xored but the value xored with is also stored in _HEAP structure. Overwriting both values will result in rewriting the function pointer with an arbitrary address.

Bypassing mitigations introduced in Windows Vista(Cont.)

- Other exploits such as followings have been presented.
  - Rewriting _HEAP structure via LFH overflow
  - Rewriting application objects via LFH overflow
  - Rewriting a function pointer via freeing and reallocating _HEAP structure

Mitigations introduced in Windows 8

- Prohibit freeing _HEAP structure
  - Mitigations for the previous slide “Rewriting a function pointer via freeing and reallocating _HEAP structure”
- Enhancement of a function pointer encoding in _HEAP structure
  - The value xored with a function pointer in _HEAP structure had been also stored in _HEAP structure. Overwriting both the value and the function pointer resulted in rewriting the function pointer to an arbitrary address.
  - The value is now saved in global and not in _HEAP structure
- Introducing guard page
  - Guard pages are placed between various memory regions managed by a heap
  - This restricts regions overwritten by overflow
- LFH allocation randomize
  - If the placement of chunks in LFH is predictable, overflow may be able to rewrite application’s important values. This randomization makes this technique difficult.
- Termination on exceptions in heap
  - Makes process terminate when a non-fatal exception occurs in heap
  - This makes difficult attackers to retry exploitation again and again.
- There are other mitigations not mentioned here.
  - http://blogs.technet.com/b/srd/archive/2013/10/29/software-defense-mitigation-heap-corruption-vulnerabilities.aspx

Conclusion

- Successful heap exploit is getting really hard after introducing randomization in chunk header and a function pointer encodings in _HEAP structure in Windows Vista.
- Furthermore, randomizing allocations and introducing guard pages in Windows 8 make it much harder to exploit heap overflow.
- But structures and algorithms used in current Windows heap implementation are complex and it may lead new exploit techniques.
- There are a lot of exploitation tactics in public. See references for the details.

References

- Third Generation Exploitation
  - http://www.blackhat.com/presentations/win-usa-02/halvarflake-winsec02.ppt
- Windows Heap Overflows
  - http://www.blackhat.com/presentations/win-usa-04/bh-win-04-litchfield/bh-win-04-litchfield.ppt
- Reliable Windows Heap Exploits
  - http://www.cybertech.net/~sh0ksh0k/heap/CSW04%20-%20Reliable%20Heap%20Exploitation.ppt
- Windows Heap Exploitation(Win2KSP0 through WinXPSP2)
  - http://www.cybertech.net/~sh0ksh0k/projects/winheap/XPSP2%20Heap%20Exploitation.ppt
- Exploiting Freelist[0] On XP Service Pack 2
  - http://www.orkspace.net/secdocs/Windows/Protection/Bypass/Exploiting%20Freelist%5B0%5D%20On%20XP%20Service%20Pack%202.pdf
- Windows Vista Heap Management Enhancements
  - https://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Marinescu.pdf
- Understanding and bypassing Windows Heap Protection
  - http://mirror7.meh.or.id/Windows/heap/Heap_Singapore_Jun_2007.pdf
- Heaps About Heaps
  - https://www.insomniasec.com/downloads/publications/Heaps_About_Heaps.ppt
- Attacking the Vista Heap
  - https://www.lateralsecurity.com/downloads/hawkes_ruxcon-nov-2008.pdf

References

- Practical Windows XP/2003 Heap Exploitation
  - http://www.blackhat.com/presentations/bh-usa-09/MCDONALD/BHUSA09-McDonald-WindowsHeap-PAPER.pdf
- Preventing the exploitation of user mode heap corruption vulnerabilities
  - http://blogs.technet.com/b/srd/archive/2009/08/04/preventing-the-exploitation-of-user-mode-heap-corruption-vulnerabilities.aspx
- Understanding the Low Fragmentation Heap
  - http://illmatics.com/Understanding_the_LFH.pdf
  - http://illmatics.com/Understanding_the_LFH_Slides.pdf
- Windows 8 Heap Internals
  - http://media.blackhat.com/bh-us-12/Briefings/Valasek/BH_US_12_Valasek_Windows_8_Heap_Internals_Slides.pdf
- Advanced Heap Manipulation in Windows 8
  - https://media.blackhat.com/eu-13/briefings/Liu/bh-eu-13-liu-advanced-heap-WP.pdf
- Software Defense: mitigating heap corruption vulnerabilities
  - http://blogs.technet.com/b/srd/archive/2013/10/29/software-defense-mitigation-heap-corruption-vulnerabilities.aspx

Contact Information

E-Mail ：research—feedback@ffri.jp
Twitter：@FFRI_Research