IT3789 Cyber Security Attack & Defence

L09 – Buffer Overflow Exploitation 2

WITH KNOWLEDGE COMES RESPONSIBILITY

Buffer Overflow Exploitation

Win32 Assembly	Fuzzing	Buffer Overflows
Stack Exploitation	Heap Exploitation	Exploit Protection Mechanism

Exception Example

```
try
{
  wc = new WebClient(); //downloading a web page
  var resultData = wc.DownloadString("http://google.com");
}
catch (ArgumentNullException ex)
{
  //code specifically for a ArgumentNullException
}
catch (WebException ex)
{
  //code specifically for a WebException
}
catch (Exception ex)
{
  //code for any other type of exception
}
finally
{
  //call this if exception occurs or not
  //in this example, dispose the WebClient
  wc.Dispose();
}
```

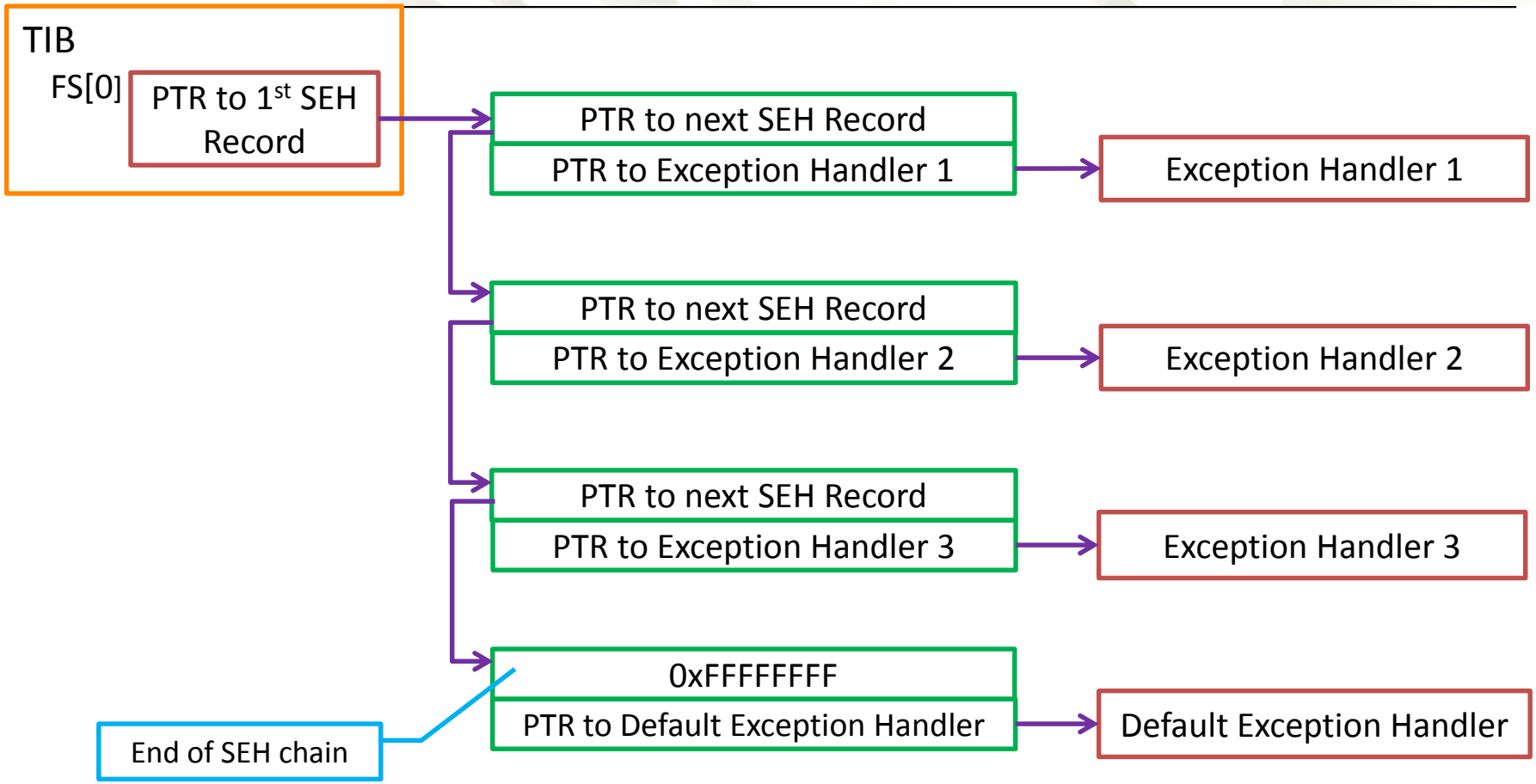
Windows Exception Handling

- Structured Exception Handler (SEH)
  - Recover operations when a program crash.
  - try/catch blocks adds a SEH record to the SEH chain
    - SEH Record is a 8 bytes structure.
      - First 4 bytes points to the next SEH record.
      - Next 4 bytes points to exception handler.
  - A default SEH is available to handle exceptions for within program.
    - Triggered when there is no user-defined exception handler.
    - Show a dialog box indicating program has crashed.
  - SEH records are stored on the stack.

Windows Exception Handling

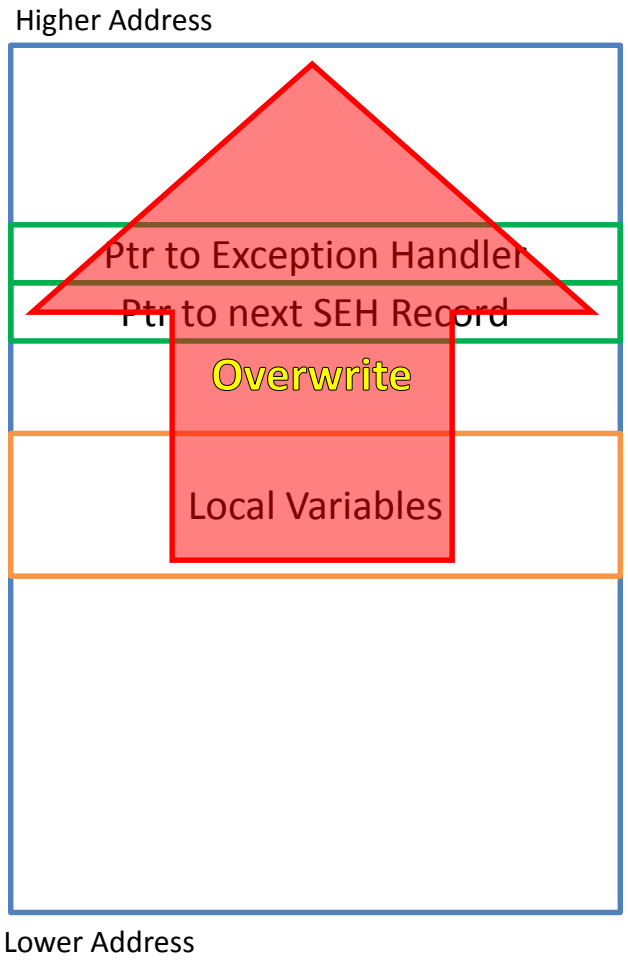
- When an exception occurs…
  1. Windows retrieves the first SEH record using the Thread Information Block (TIB) structure.
  2. Walks through the SEH chain.
  3. Finds the suitable handler to exit the application elegantly.

SEH Chain


TIB
FS[0]
PTR to 1st SEH Record
PTR to next SEH Record
PTR to Exception Handler 1
Exception Handler 1
PTR to next SEH Record
PTR to Exception Handler 2
Exception Handler 2
PTR to next SEH Record
PTR to Exception Handler 3
Exception Handler 3
0xFFFFFFFF
PTR to Default Exception Handler
Default Exception Handler
End of SEH chain

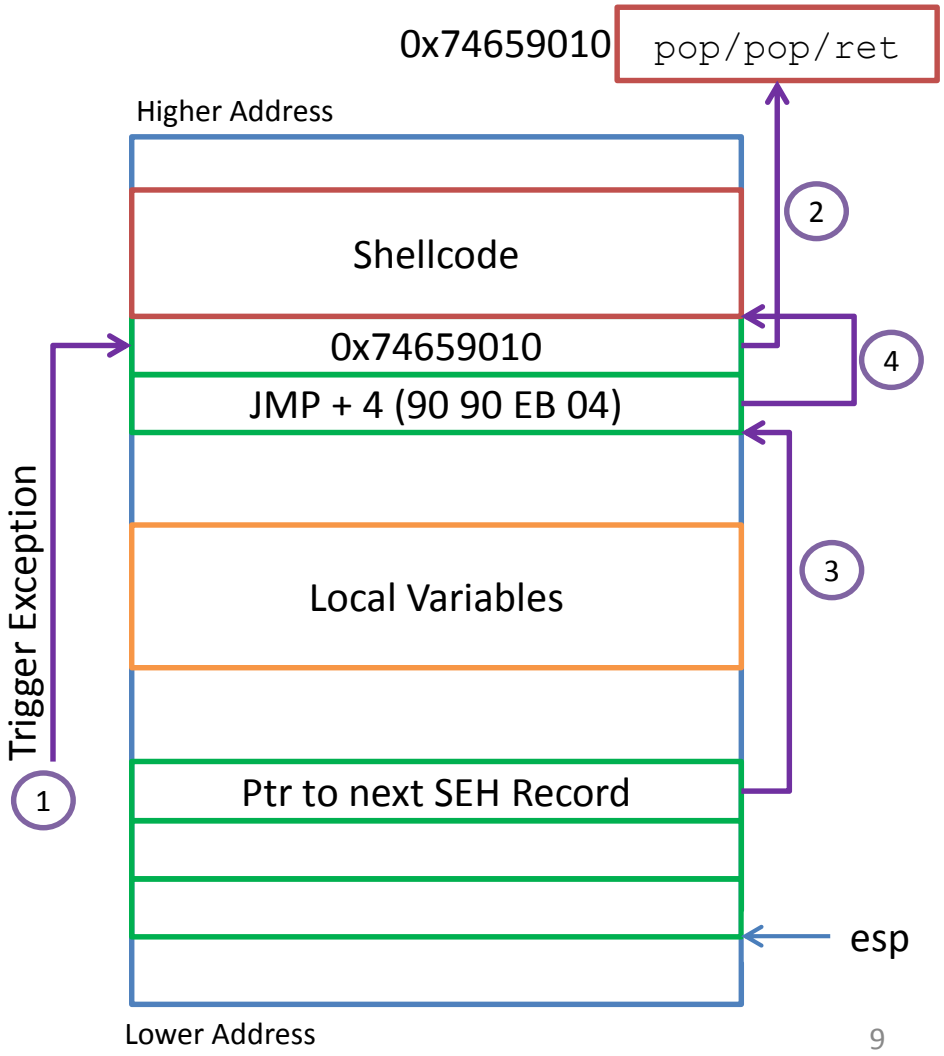
Techniques for Shellcode Execution in Stack Overflow

- SEH Overwrite
  - When a SEH record can be overwritten from a neighbour stack frame.
  - And an exception can be triggered to load the exception handler.
  - Able to bypass stack cookies (/GS) with this technique.



Techniques for Shellcode Execution in Stack Overflow

- SEH Overwrite
  1. Trigger exception.
  2. When exception is triggered, the pointer of the next SEH record is 8 bytes from top of the stack. Use pop/pop/ret to load the pointer into EIP.
  3. Execution flow is passed to the next SEH record.
  4. Instructions to jump to payload resulting in payload execution.



Buffer Overflow Exploitation

Win32 Assembly	Fuzzing	Buffer Overflows
Stack Exploitation	**Heap Exploitation**	Exploit Protection Mechanism

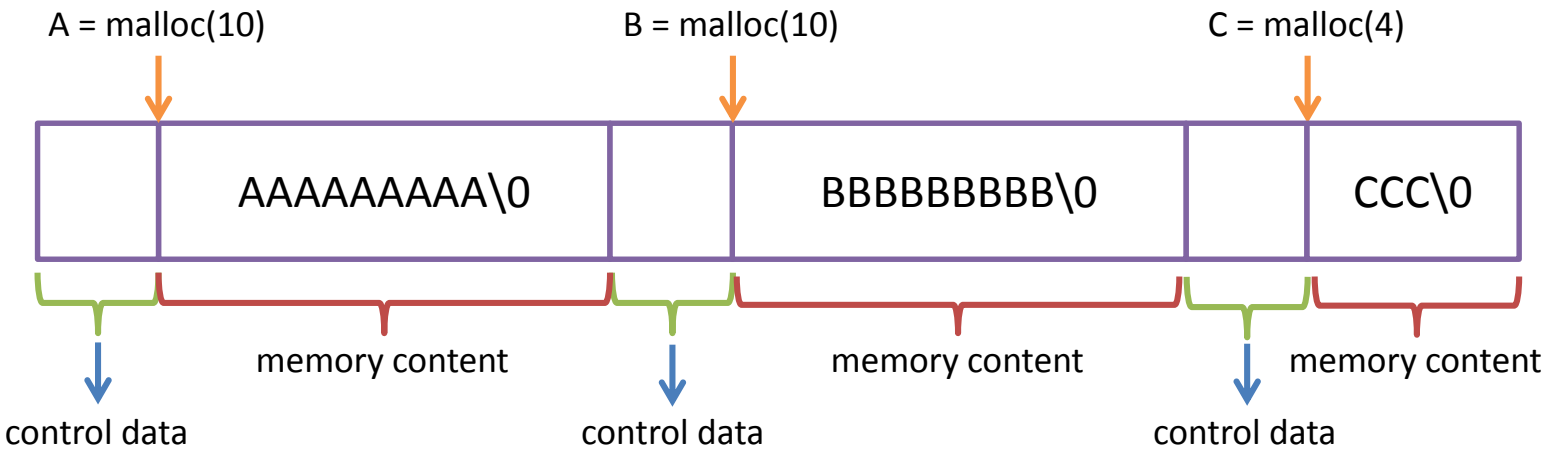
The Heap Memory

- Heap is a memory area used by an application and is allocated dynamically at runtime.
  - Heap memory can be allocated via malloc-type functions.
    - Windows: HeapAlloc()/HeapFree()
    - ANSI C: malloc()/free()
    - C++: new/delete
  - Storage for global variables or variables too large to be stored on the stack.

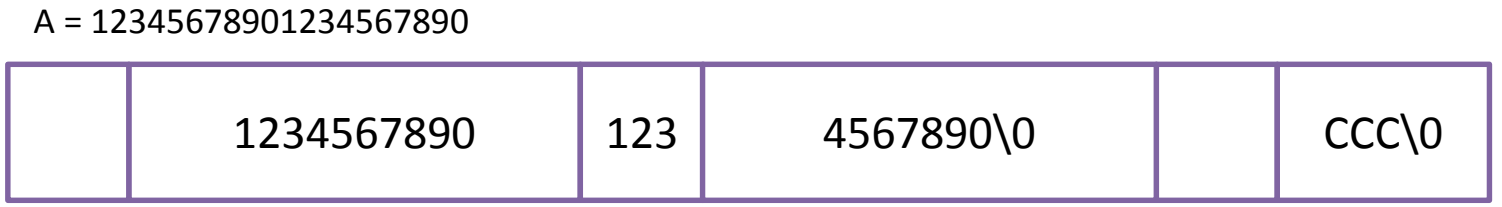
The Heap Memory

- Characteristic of the heap memory.
  - Heap memory is persistent between functions.
    - Memory will remain allocated until it is freed explicitly.
  - Heap and heap allocation are deterministic(?).
  - Each process has one default process heap but programs can create more heaps to store data.
    - A pointer to the default heap is stored in the PEB (*Process Environment Block*)
    - All heaps in a process are kept in a linked list.
  - No dedicated registers.
    - No concept of saved EIP in a heap as in the stack.

Heap Overflow



Before Overflow



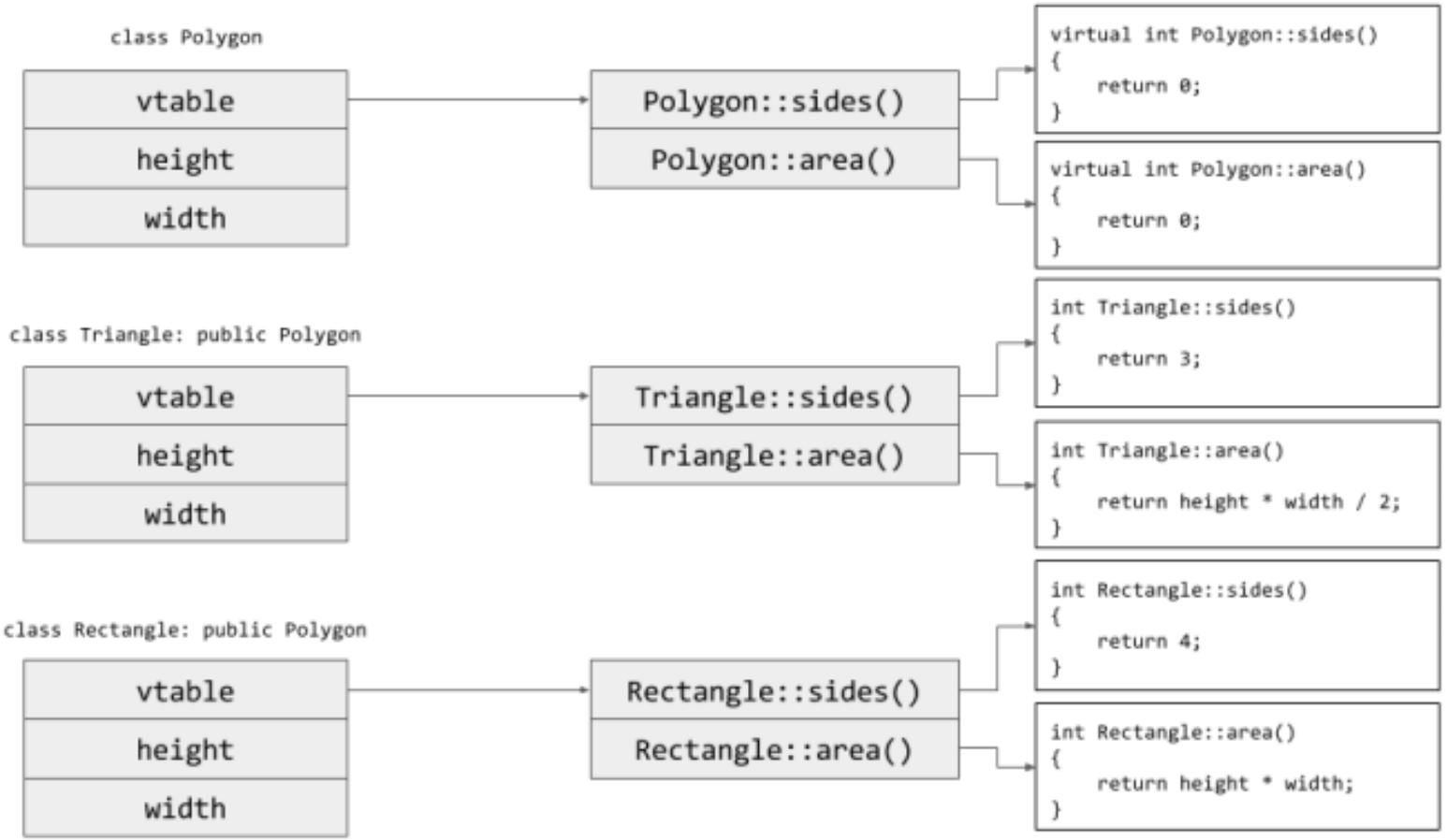
After Overflow

Attackers can modify control data such as function pointers to gain control of the path of execution.

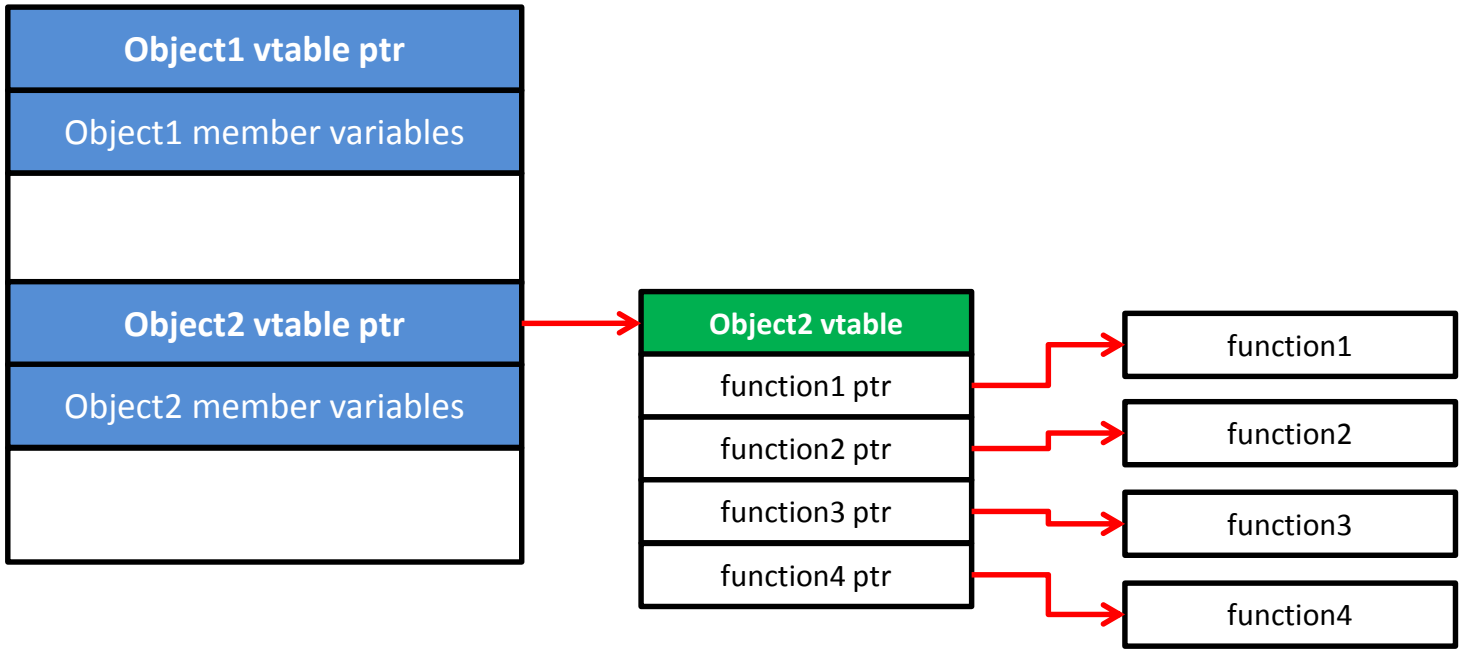
Corrupting Function Pointers

- Overwrite one heap object from another neighbouring memory chunk.
  - Possible methods
    - Overwrite function pointer found in the neighbouring memory chunk and causing a jump or call to the address when that function is called by the program.
    - Overwrite virtual-function table pointer (vtable pointer) of an object in a neighbouring memory chunk.

What is Vtable?

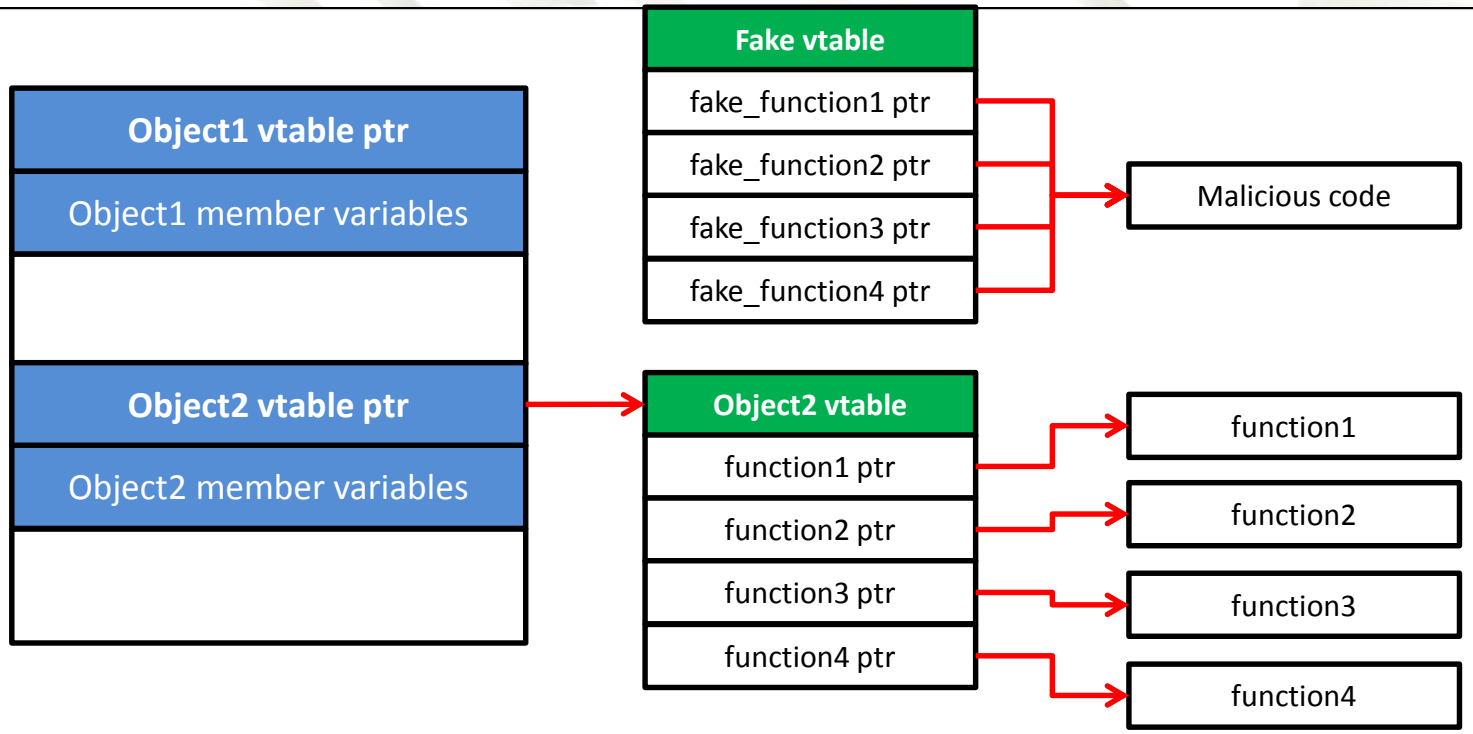


Vtable Overwrite



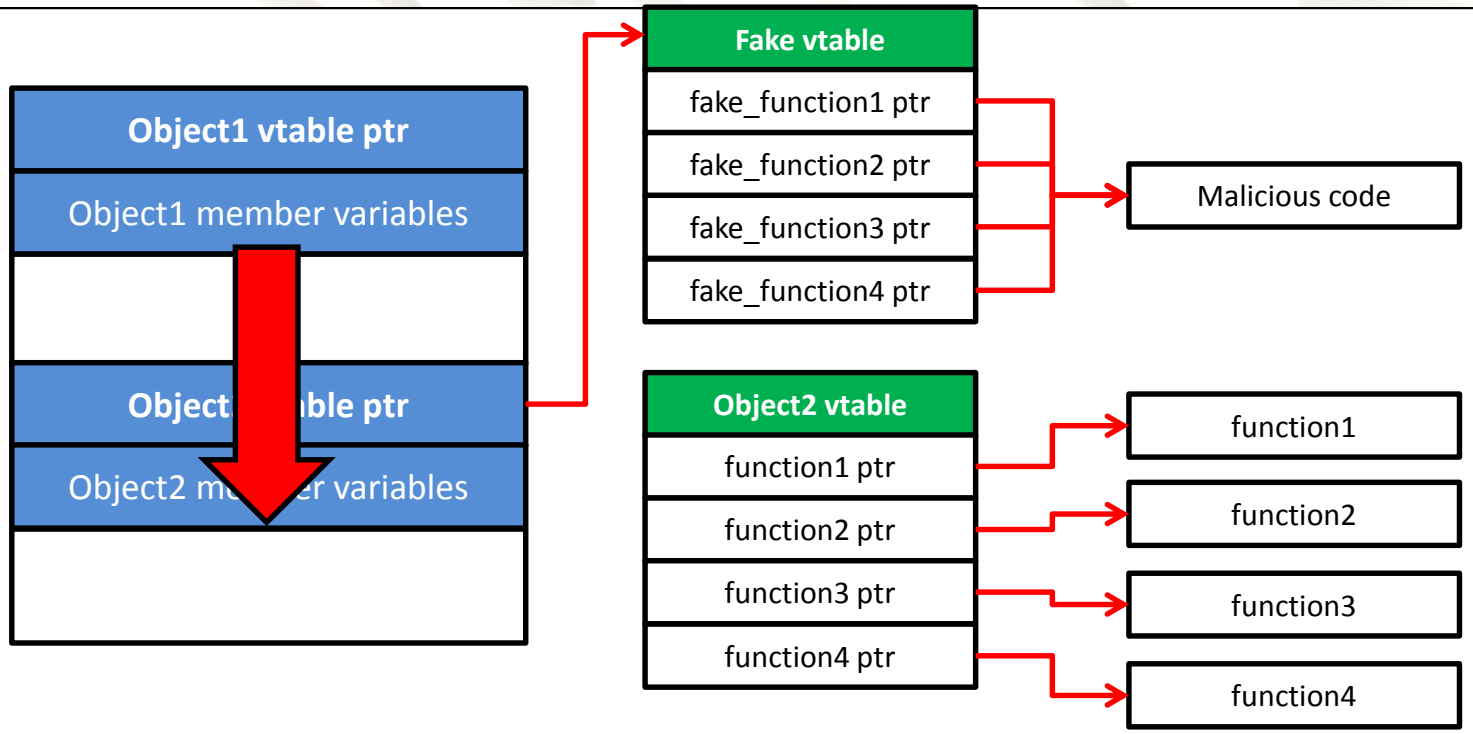
- In C++, vtables contain pointers to virtual functions to support polymorphism.
- Instance of a particular class will share the same vtable.
  - 1 vtable per class.
- Vtables contain method pointers that points to virtual functions in that class.

Vtable Overwrite



- Attacker creates a fake vtable pointing to malicious code (shellcode).

Vtable Overwrite

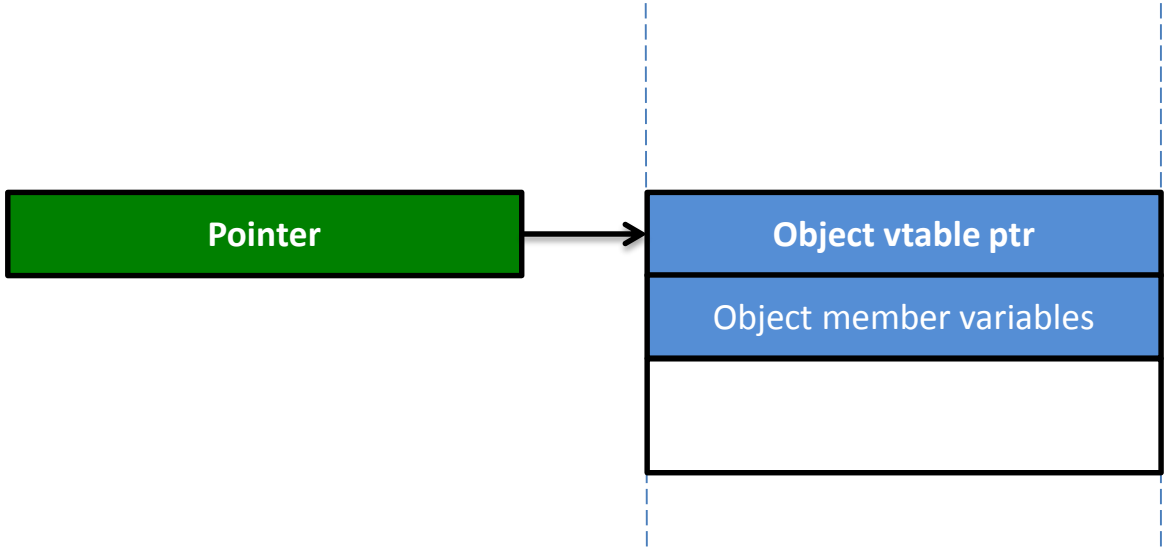


- Modify vtable pointer to point to fake vtable which points to malicious code.
- The fake vtable can cause malicious code to execute when one of the class functions is executed.
  - The destructor is a good function to target since it is always executed when object is deleted from memory.

Use-after-free Vulnerabilities

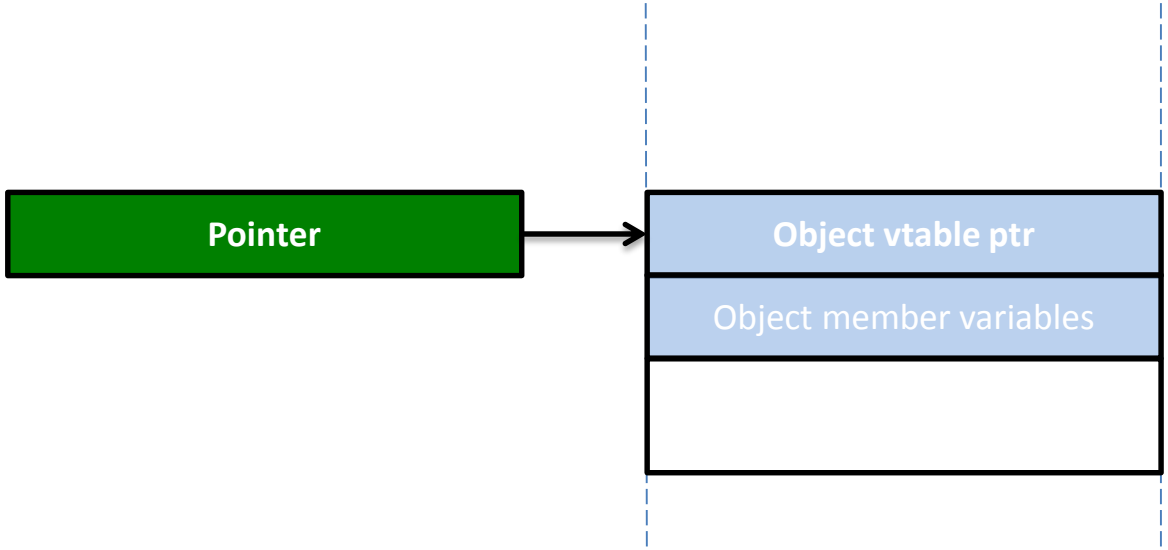
- Heap entry must be created and then freed.
- Overwrite the freed memory area with attacker data (e.g. pointer to malicious code).
- The pointer to freed memory is being used again after the free operation.
- Opportunity to take control of the execution process.

Use-after-free Vulnerabilities Explained



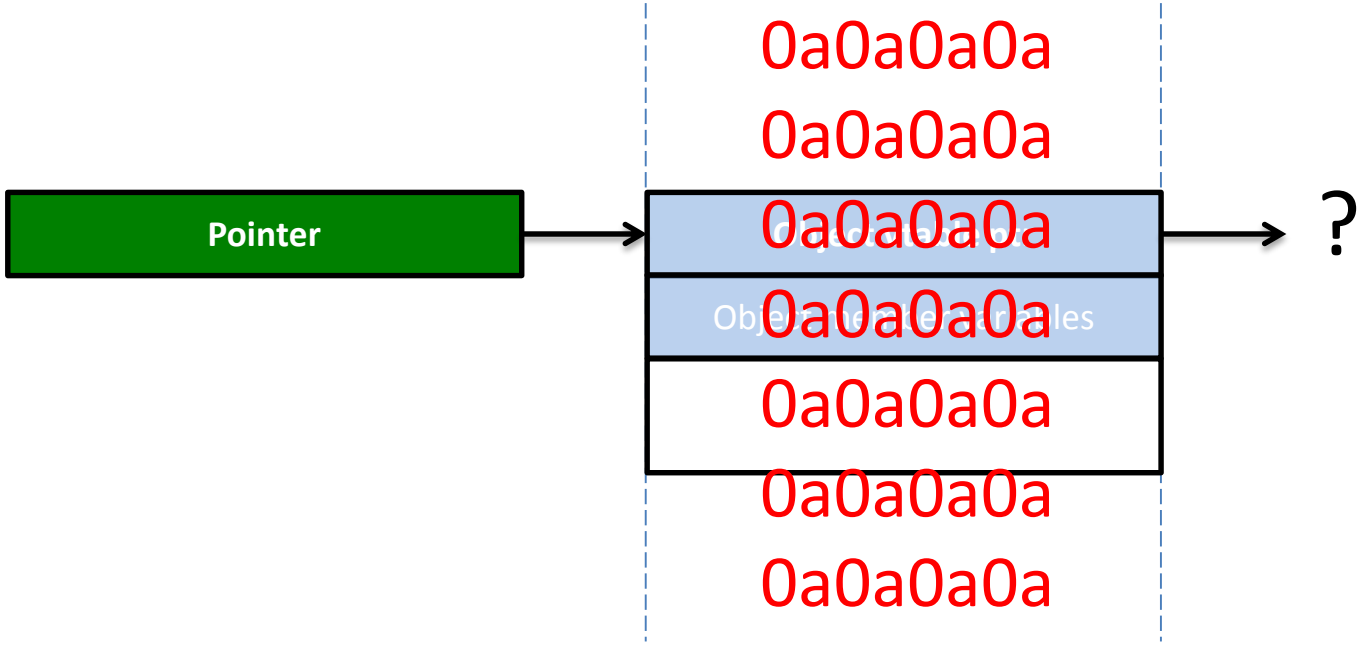
- Heap entry created when Object is instantiated.
- Pointer variable is assigned the pointer that points to object created.

Use-after-free Vulnerabilities Explained



- Object is destroyed and heap memory is freed.
- However, pointer is still pointing to freed memory.

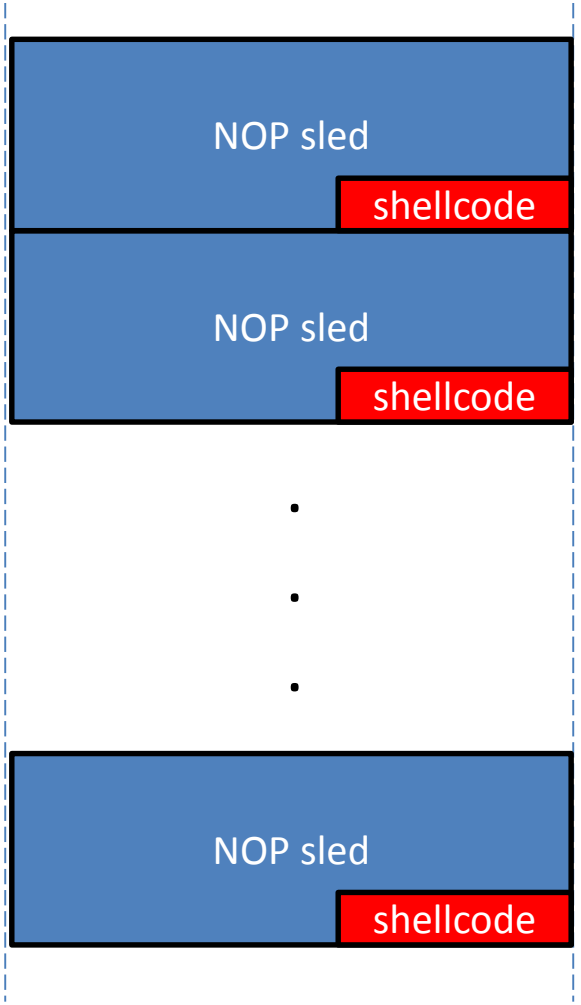
Use-after-free Vulnerabilities Explained



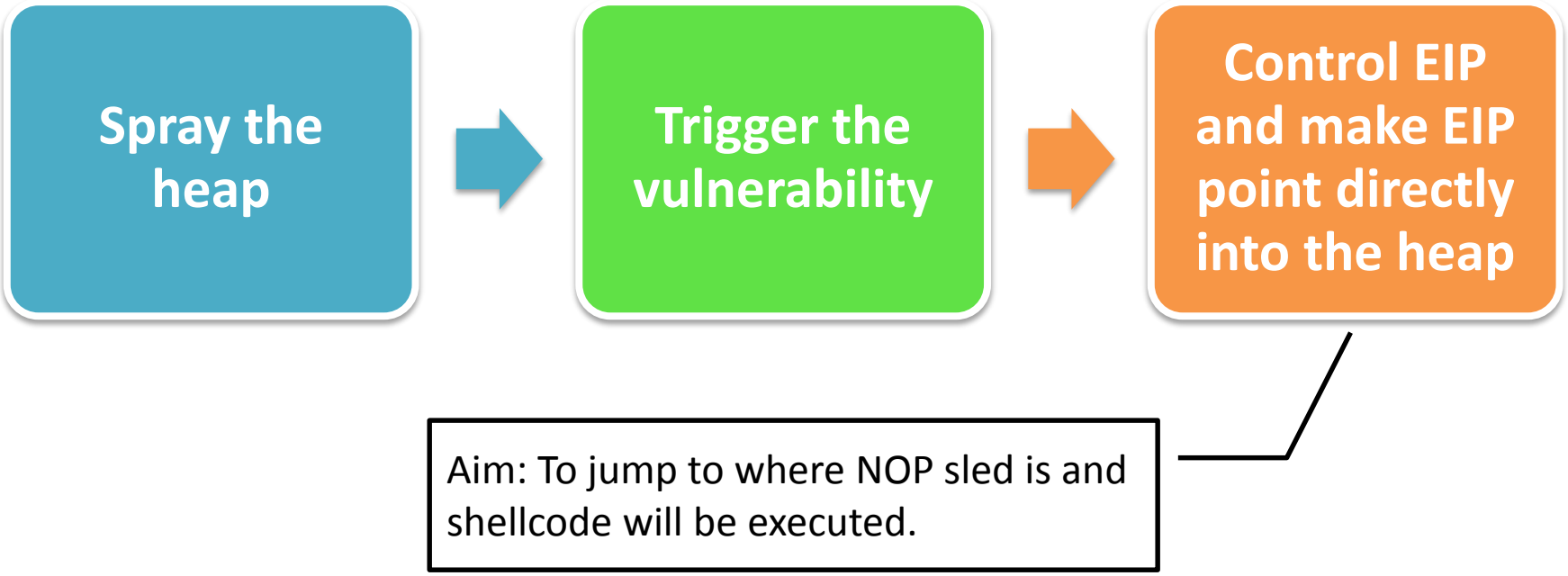
- Overwrite the freed memory area with address that points to malicious code.
- Invoke function in deleted object instance via pointer.

Heap Spray

- A payload delivery technique
  - Create large contagious memory chunks in the heap.
  - Each memory chunk consists of NOP sled with shellcode appended at the end.
  - An ideal heap spray will allocate memory chucks at predictable memory locations.
  - No need to worry about bad data (e.g. NULL byte) in heap spray.
    - Unless input to trigger the bug has some restrictions.



Steps for Heap Exploitation



Buffer Overflow Exploitation

Win32 Assembly	Fuzzing	Buffer Overflows
Stack Exploitation	Heap Exploitation	Exploit Protection Mechanism

Stack Cookies (Canary)

Stack Cookies (Canary)

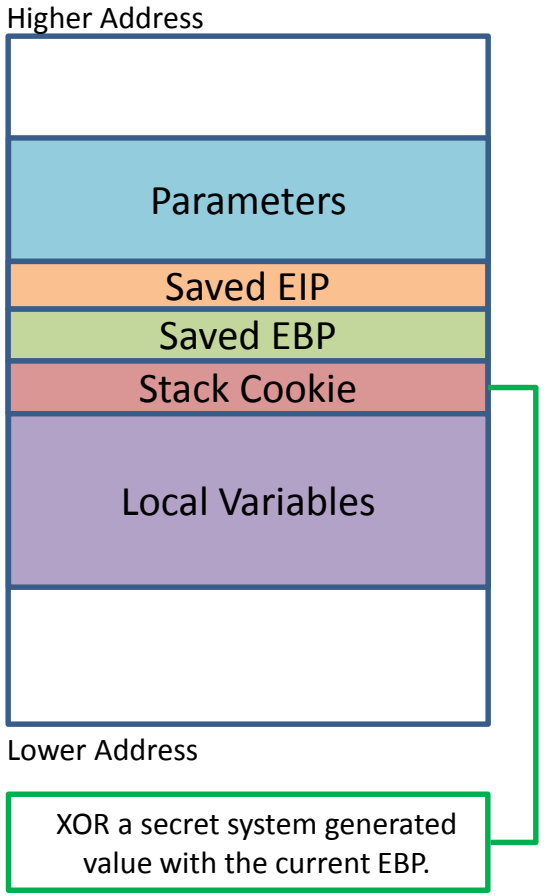
- /GS compiler option
- Protection mechanism of /GS
  - Stack cookie (canary)
  - Variable reordering
- Detect and prevent saved EIP address overwrites.
- Value inserted during function prologue and checked during function epilogue.

Stack Cookies (Canary)

- Situation where compiler option is not applied.
  - Functions that do not contain a buffer.
  - Optimizations not enabled.
  - Functions marked with the *naked* keyword (only in x86).
  - Functions containing inline assembly on the first line.
  - Functions defined to have a variable argument list.
  - Buffers less than 4 bytes in size.

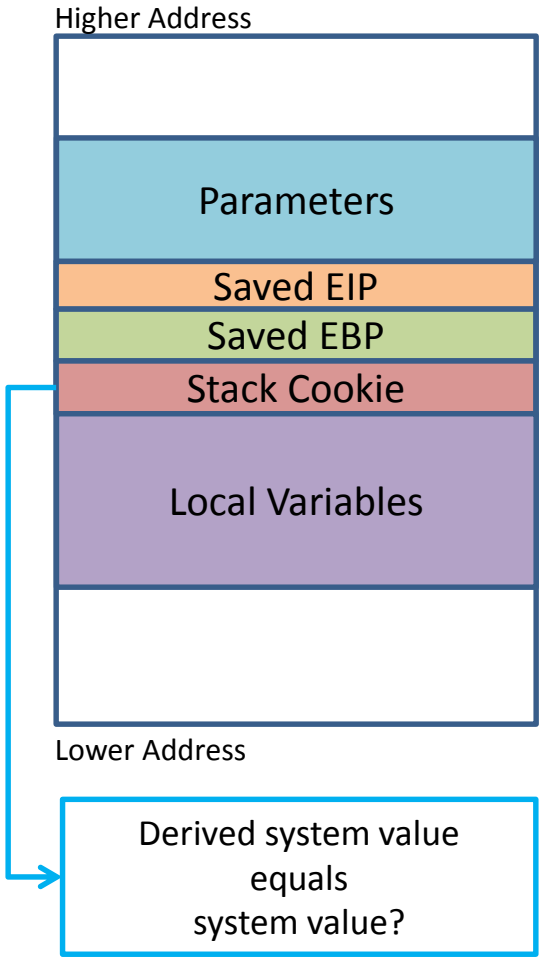
Stack Cookies (Canary)

- Function prologue with stack cookie.
  1. Function parameters are pushed onto the stack.
  2. EIP pushed onto stack.
  3. EBP pushed onto stack.
  4. Generate stack cookie and push the value onto the stack.
  5. Allocated space for local variables on the stack and values are populated.

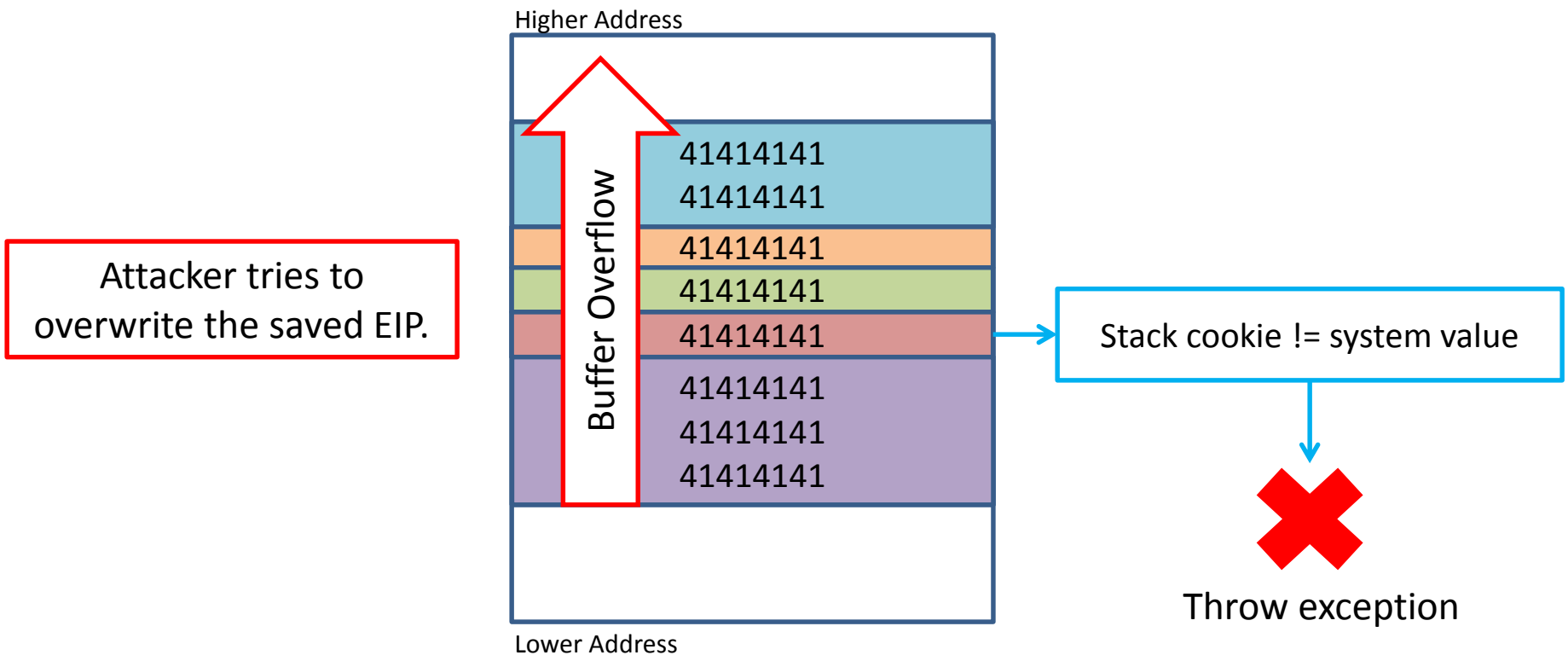


Stack Cookies (Canary)

- Function epilogue with stack cookie.
  1. Obtain the cookie.
  2. XOR the cookie with the current EBP.
  3. Check if the value matches the system value.
  4. If the values matches, return to caller, else stop program execution.

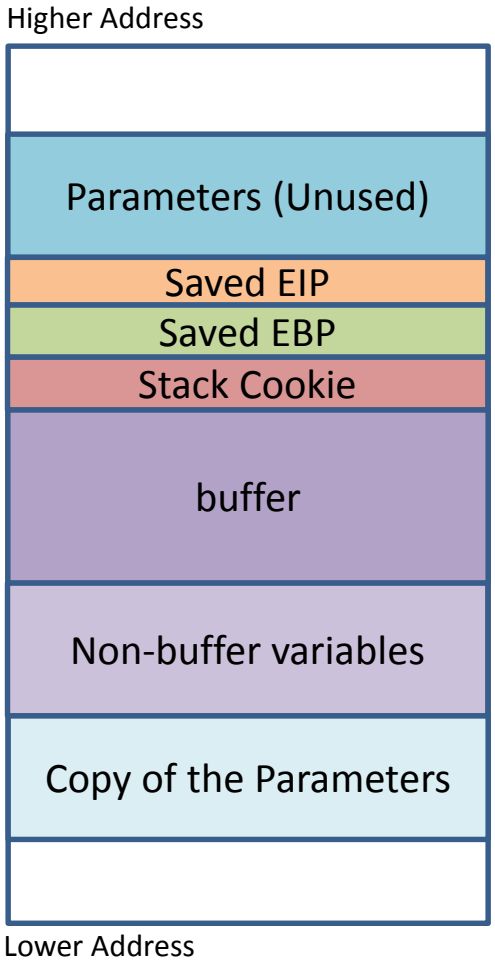


Stack Cookies (Canary)



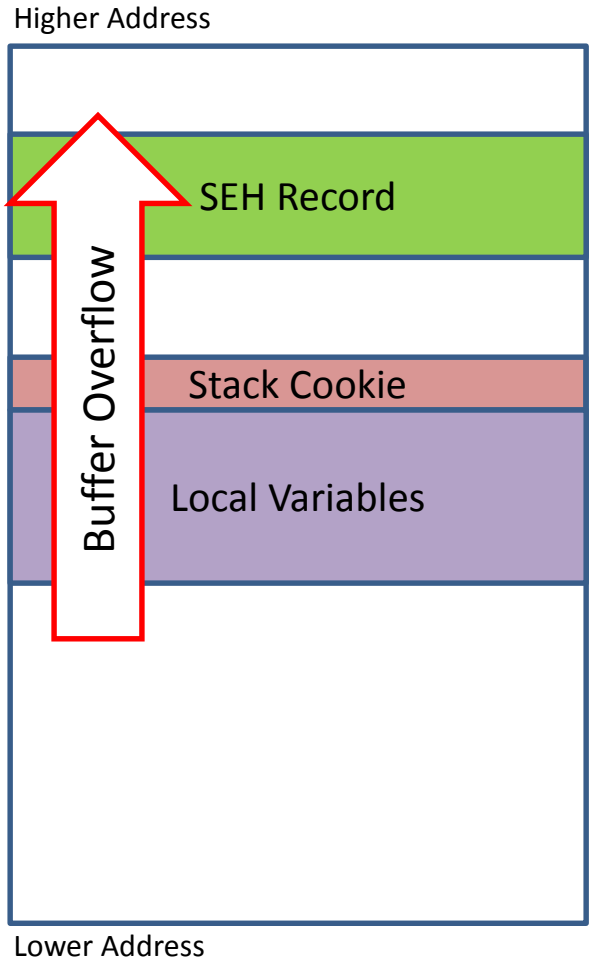
Variable Reordering

- Compiler rearranges layout of the stack.
  - Move string buffers to higher address space non-buffer variables.
  - Copy of the parameters is copied to the top of the stack frame.
  - Non-buffer variables and the copy of parameters cannot be overwritten.



Bypassing Stack Cookie

1. Overwrite SEH record.
2. Trigger exception.
   - Causes a call exception handler in fake SEH record.



Safe Structured Exception Handling (SafeSEH)

- Compiled with /SafeSEH linker option (only for x86).
- Header of compiled binary will contain a table of valid exception handlers (SafeSEH table).
- When exception handler is called, table is checked.
- Prevent overwrite and use of SEH structures on the stack.

Bypassing SafeSEH

- For effective protection, all binaries in a process must be compiled with SafeSEH.
- Pointing exception handler to a binary not compiled with SafeSEH.

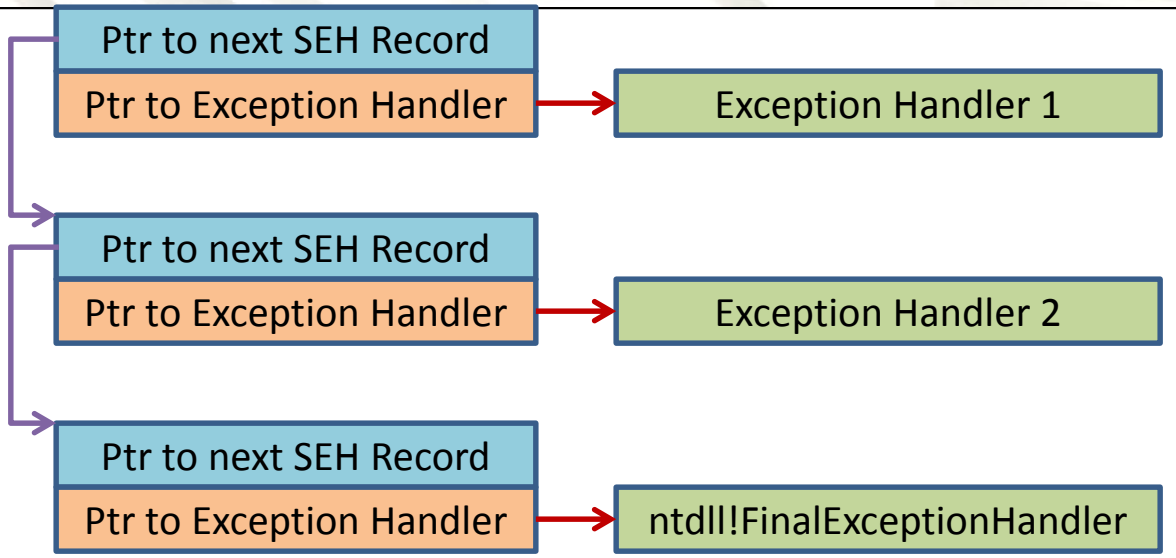
```
00400000 004cc000 sipXezPhone          /SafeSEH OFF
10000000 10125000 sipXtapi             /SafeSEH OFF
5d090000 5d12a000 COMCTL32             /SafeSEH ON  /GS
71aa0000 71aa8000 WS2HELP              /SafeSEH ON  /GS
71ab0000 71ac7000 WS2_32               /SafeSEH ON  /GS
71ad0000 71ad9000 WSOCK32              NO_SEH
76080000 760e5000 MSVCP60              /SafeSEH ON  /GS
763b0000 763f9000 comdlg32             /SafeSEH ON  /GS
76b40000 76b6d000 WINMM                /SafeSEH ON  /GS
774e0000 7761d000 ole32                /SafeSEH ON  /GS
77c10000 77c68000 MSVCRT               /SafeSEH ON  /GS
77dd0000 77e6b000 ADVAPI32             /SafeSEH ON  /GS
77e70000 77f02000 RPCRT4               /SafeSEH ON  /GS
```

SEH Overwrite Protection (SEHOP)

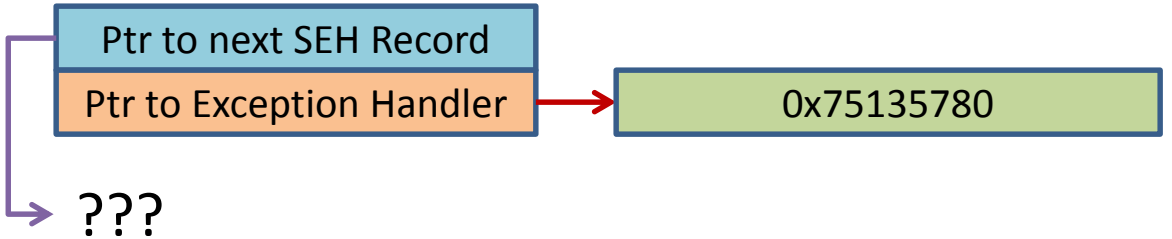
- Also prevent SEH overwrite exploitation technique.
- Dynamic checks in the exception dispatcher.
  - Verify a thread exception handler list is intact before allowing to call any registered handler.
  - If the SEH chain is broken due to SEH overwrite, the exploitation will be stopped.

SEH Overwrite Protection (SEHOP)

Valid SEH Chain



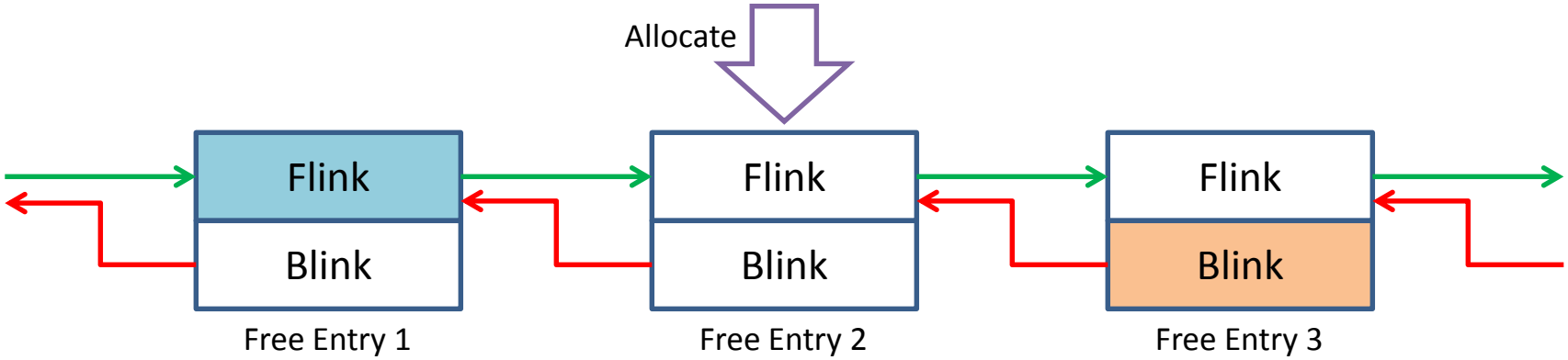
Invalid SEH Chain



Cannot reach FinalExceptionHandler function in ntdll.dll

Heap Protection

- Safe Unlinking
  - OS verifies the forward and backward pointers point to the same chunk before unlinking.
  - Detect heap memory corruption.



1. Free Entry 2 -> Flink -> Blink == Free Entry 2 -> Blink -> Flink
2. Free Entry 2 -> Flink -> Blink == Free Entry 2

Heap Protection

- Heap Metadata Cookies
  - One byte value added to each heap entry header.
  - Cookie validated during operations such as heap chunk deallocation.
  - Heap Entry Metadata Cookie Randomization
    - Header fields is XOR'ed with a random value.
      - Protect the integrity of metadata.
    - Heap manager will unpack and verify the integrity of the entry before operating on it.

Data Execution Prevention (DEP)

- Prevent execution of code in heap, stack or data sections of memory.
  - Hardware DEP relies on processor hardware to determine if memory page as executable or not.
  - Software DEP performs additional checks on exception handling mechanisms in Windows.

Data Execution Prevention (DEP)

- Four possible configurations
  1. OptIn
     - Protection only enabled for applications that have explicitly opted in.
       - Only Windows system binaries are covered by DEP by default.
     - May be turned off at runtime by application or loader.
  2. OptOut
     - All processes are protected by DEP except those on exception list.
     - May be turned off at runtime by application or loader.
  3. AlwaysOn
     - DEP is always on and cannot be diabled at anytime.
  4. AlwaysOff
     - DEP is always off and cannot be enabled at anytime.

Address Space Layout Randomization (ASLR)

- Introduce randomness into the memory addresses used by a process.
  - Hinders attacks by preventing an attacker from being able to easily predict target addresses.
  - Randomly arranging the positions of key data areas of a program.
    - Executable images, DLL images, stack, heap, PEB/TEB
    - Subjected to brute force as some memory sections have lesser entropy.

Buffer Overflow Exploitation (2)

Stack Exploitation

- Techniques for Shellcode Execution in Stack Overflow: SEH Overwrite

Heap Exploitation

- Overflowing Dynamic Strings
- Corrupting Function Pointers in C++

Exploit Protection Mechanism

- Stack Cookie
- SAFESEH
- SEHOP
- Heap Protection
- DEP
- ASLR