Windows 8 Exploit

2013. 07.13

서만기 연구원

CodeEngn

www.CodeEngn.com

2013 CodeEngn Conference 08

AhnLab

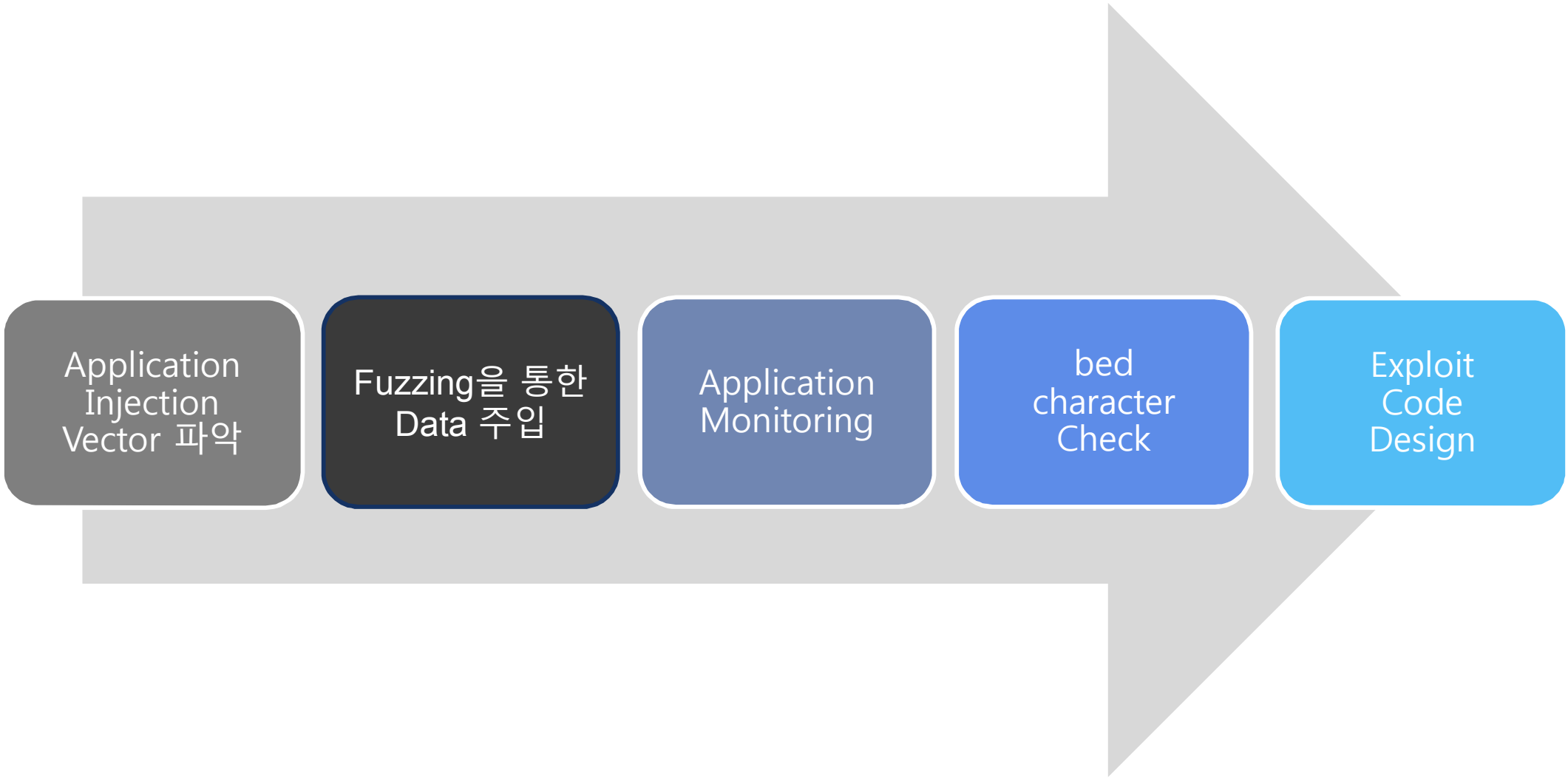
Contents

AhnLab

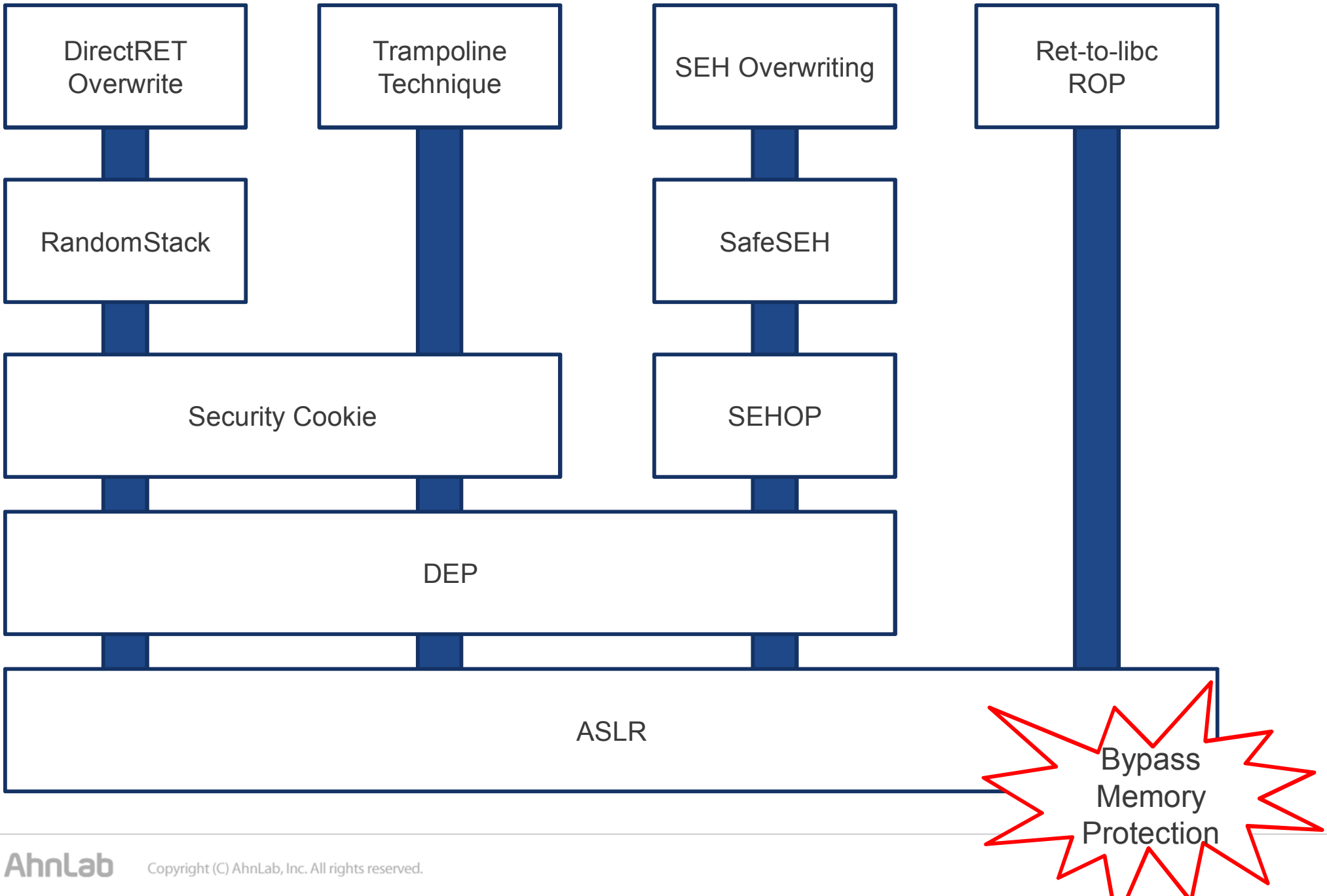
01 Exploiting 개요

AhnLab

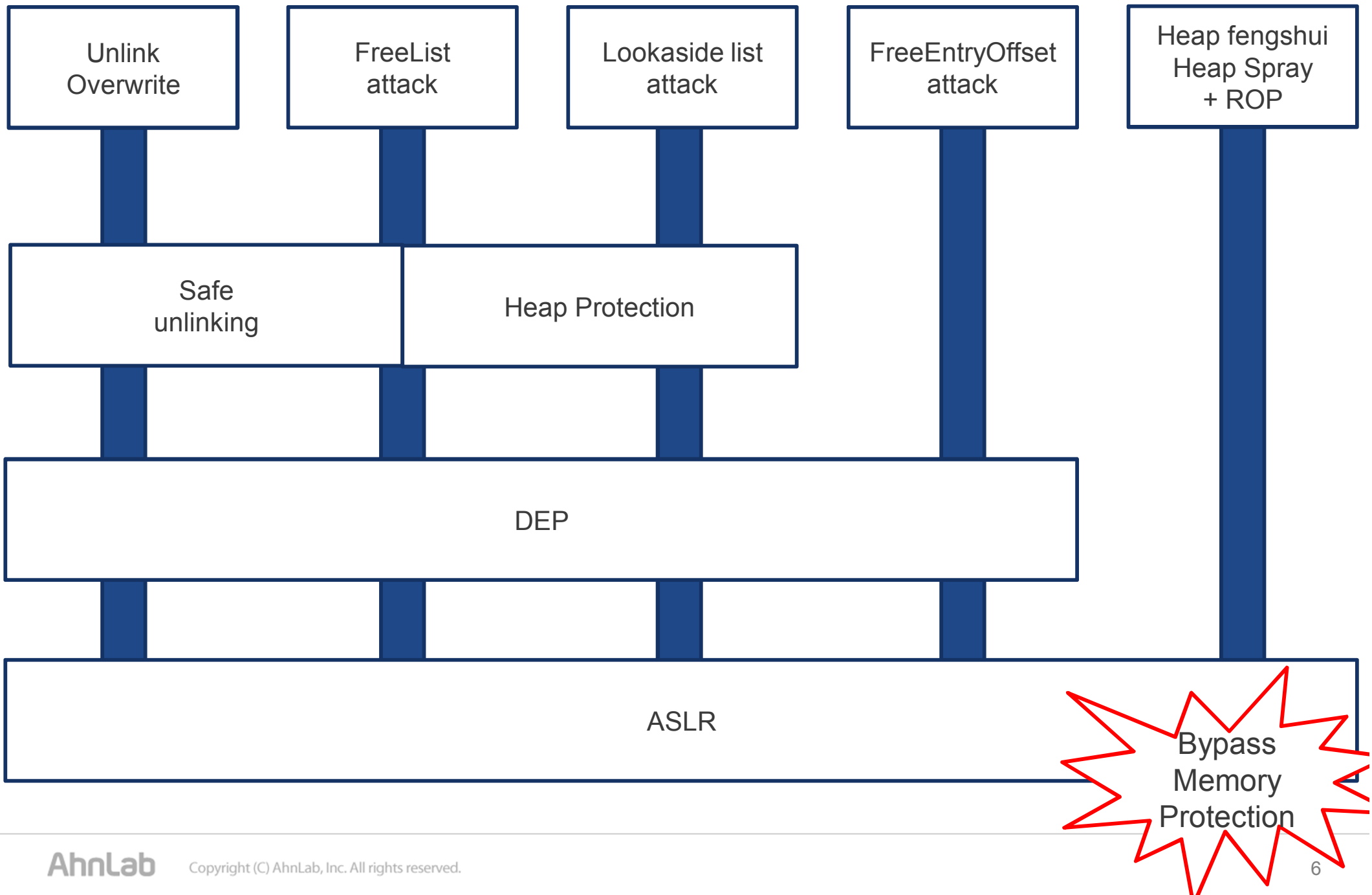
Exploit Writing Process



History Of Stack Exploiting



History Of Heap Exploiting



02 Windows 8 Memory Protection

AhnLab

❖ ASLR

- ◆ Address Space Layout Randomization의 약자
- ◆ Windows Vista에 처음 활용 되었음
- ◆ PE파일 포맷을 가지는 파일(이미지)이 메모리에 로드 될 때 base 주소를 랜덤하게 생성
- ◆ Image base뿐만 아니라 stack, heap, 프로세스의 메모리 공간 역시 랜덤

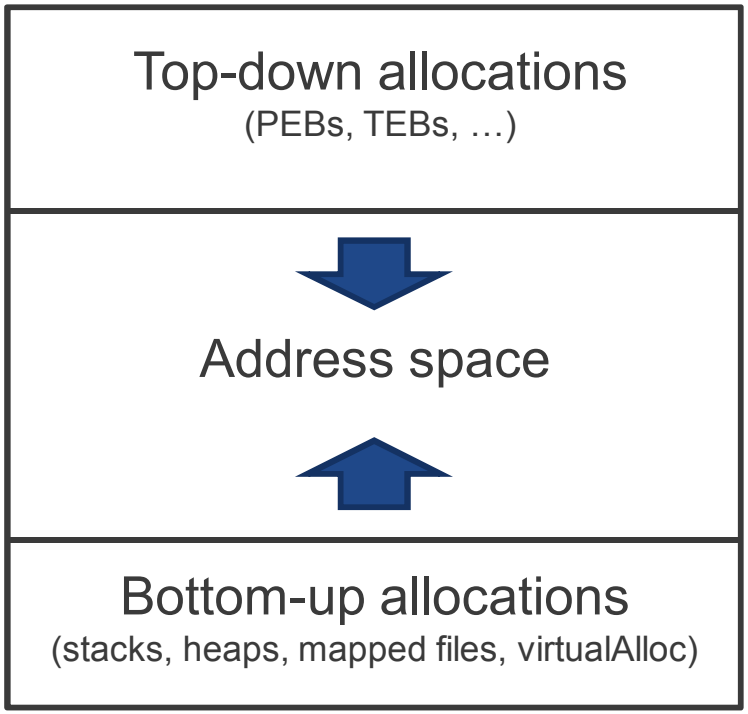
❖ ASLR 우회

- ◆ ASLR 적용되지 않은 모듈 활용
- ◆ Spraying 활용
- ◆ Memory 주소 예측 및 Bruteforcing

Improvement ASLR

- Force ASLR
  - Exploit Code가 Non-ASLR Module을 활용한다는 점에서 착안
  - Non-ASLR images를 강제로 Randomized하게 변경
    - ImageBase값을 사용하지 못하는 것 처럼 동작함
    - Bottom-up Randomization

- Bottom-up & top-down



DEP

❖ DEP?

- ◆ 데이터 실행 방지(Data Execution prevention)
- ◆ stack/stack의 일부분을 non-executable page로 설정하여 stack에서 shellcode가 실행되지 못하게끔 함

❖ DEP 모드

- ◆ Hardware DEP
  - ▪ NX bit((No Execute page protection - AMD)
  - ▪ XD bit(execute Disable - INTEL)
  - ▪ 하지만 지원하지 않는 CPU일 경우 활용할 수 없다.
- ◆ software DEP
  - ▪ CPU가 지원하지 못할 경우 Windows DEP는 Software SEP로 동작

DEP(cont)

❖ DEP 설정 값

- ◆ OptIn
  - 일부 시스템 바이너리와 프로그램에 대해 DEP 적용
- ◆ OptOut
  - DEP가 적용되지 않는 특정 프로그램 목록에 있는 것 외에 모든 프로그램 DEP 적용
- ◆ AlwaysOn
  - DEP제외 목록을 사용할 수 없으며 DEP 시스템 호환성 수정 프로그램이 적용되지 않는다.
- ◆ AlwaysOff
  - Hardware DEP지원 관계 없이 DEP가 시스템 전체를 보호하지 않음

ROP Mitigation(Improvement DEP)

- ROP Mitigation
  - Stack Frame의 유효성 여부 확인
    - VirtualProtect(), VirtualAlloc()등 메모리 관련 함수 포함
  - ESP Point 수정을 어렵게 하여 Stack Pivot으로 역할을 하지 못하게 함

ROP 개요

❖ ROP?

- ◆ Return-Oriented Programming
- ◆ 기본적인 개념은 ret-2-libc와 동일함
- ◆ 필요한 코드의 주소값 활용하여 Call/jmp를 반복하는 ROP chain을 생성하여 메모리 보호 기법 우회

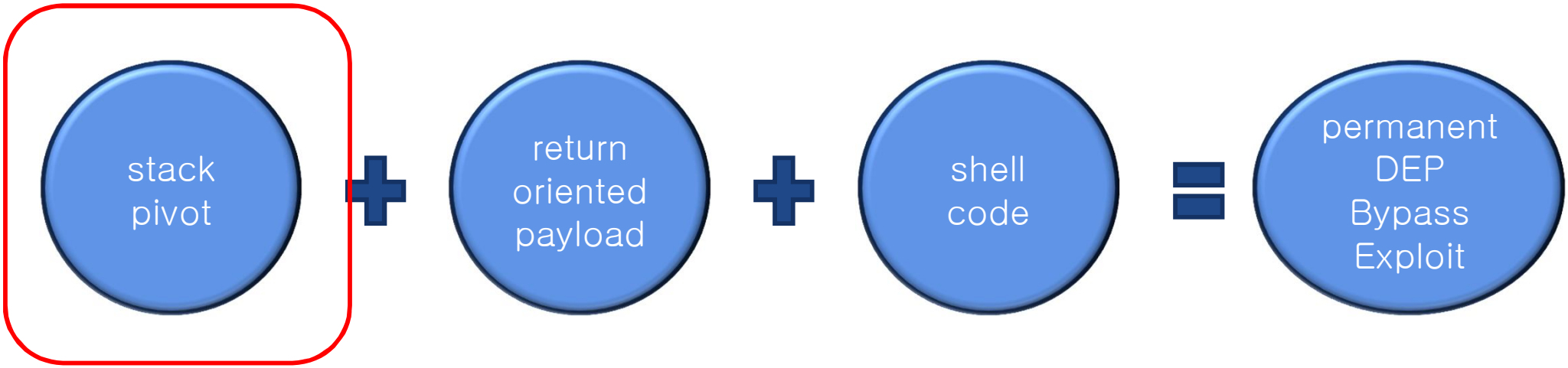
❖ ROP 요구 조건

- ◆ shellcode를 실행 가능한 지역에 복사하고, 호출 가능해야 함
- ◆ shellcode가 실행되기 전 DEP 설정을 변경 해야 한다.

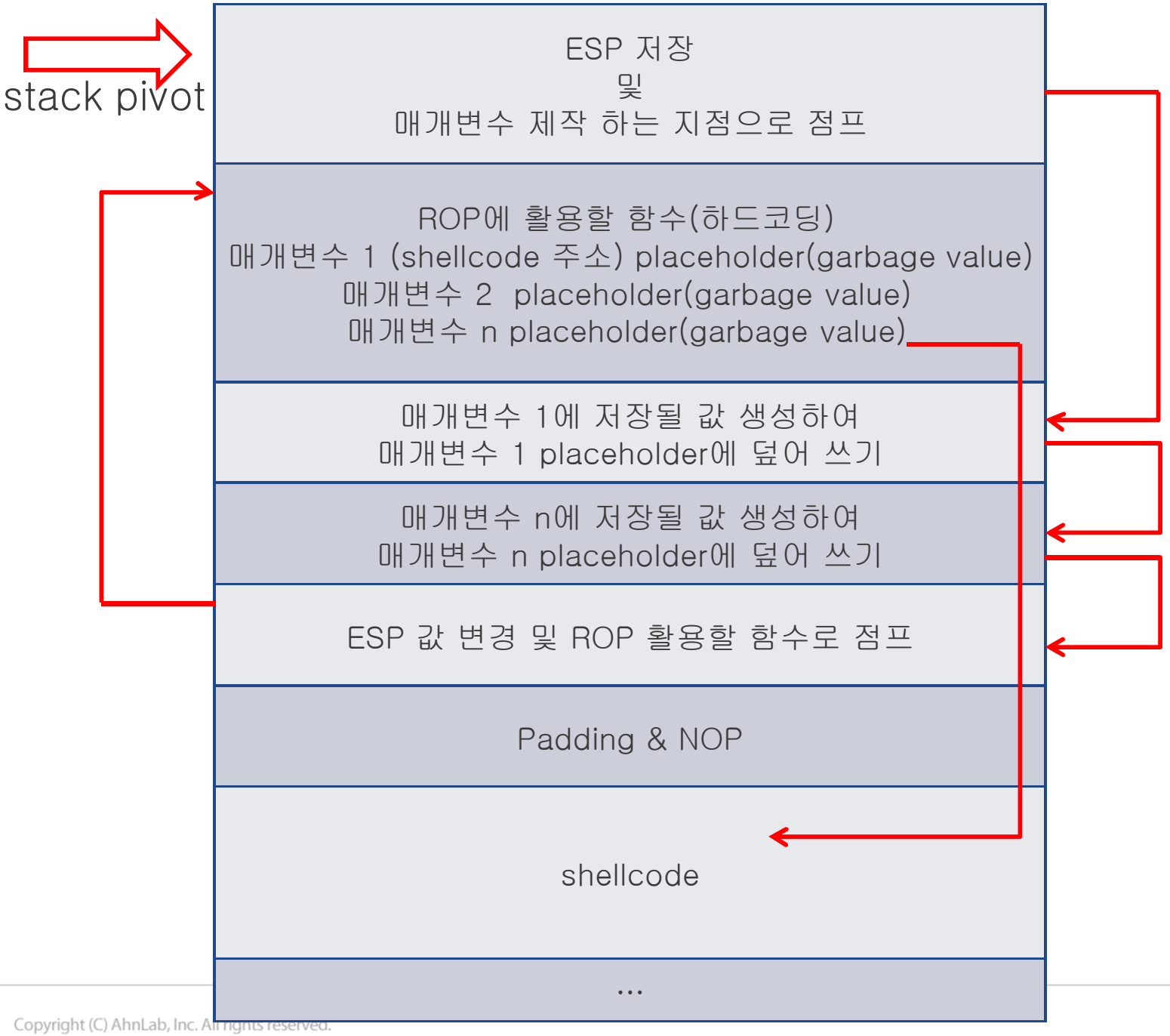
❖ gadget

- ◆ 목적을 달성하기 위해 필요한 코드들을 Call/ret를 반복 적으로 수행하는 ROP chain을 의미 함

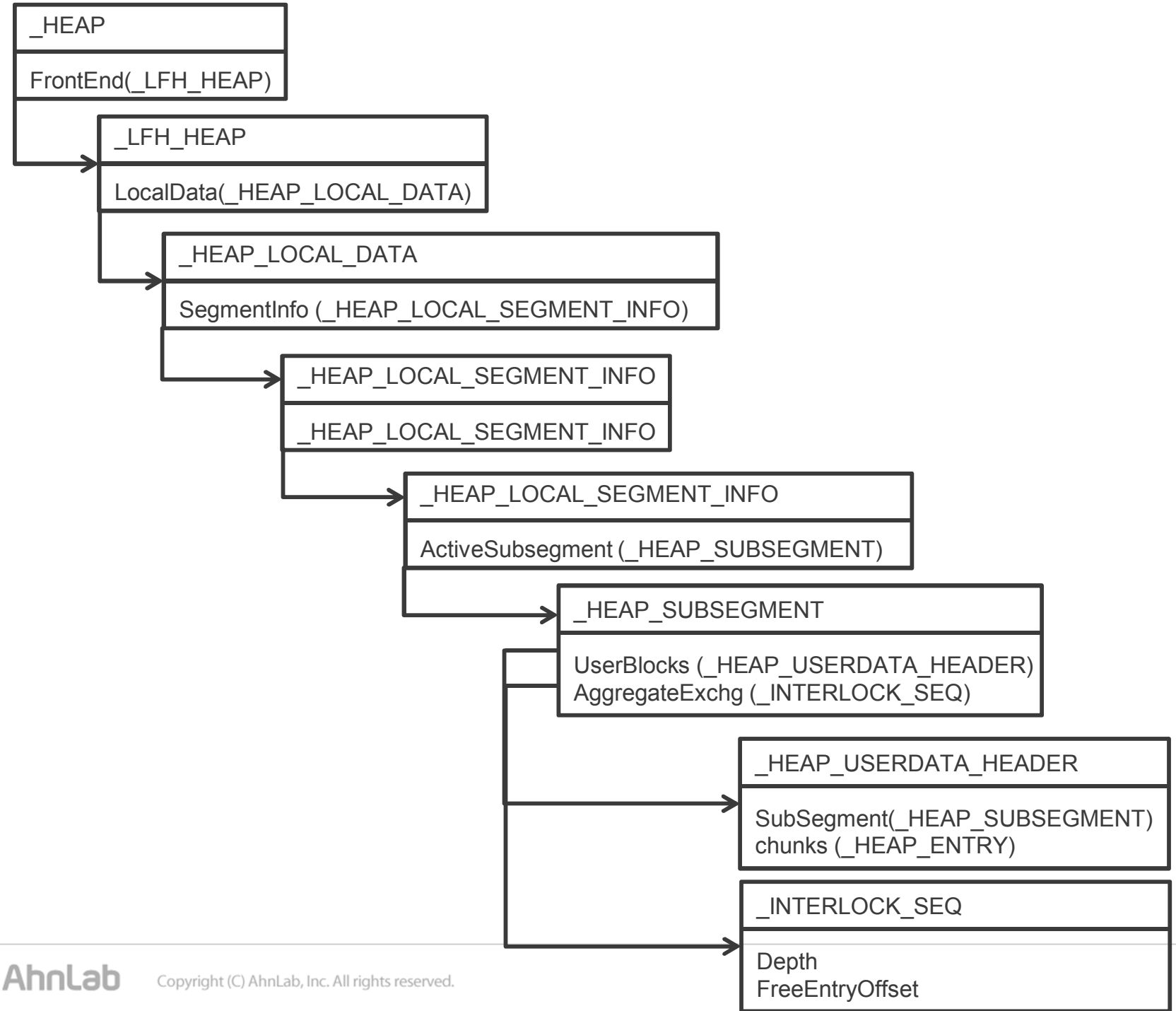
ROP 개요(cont)



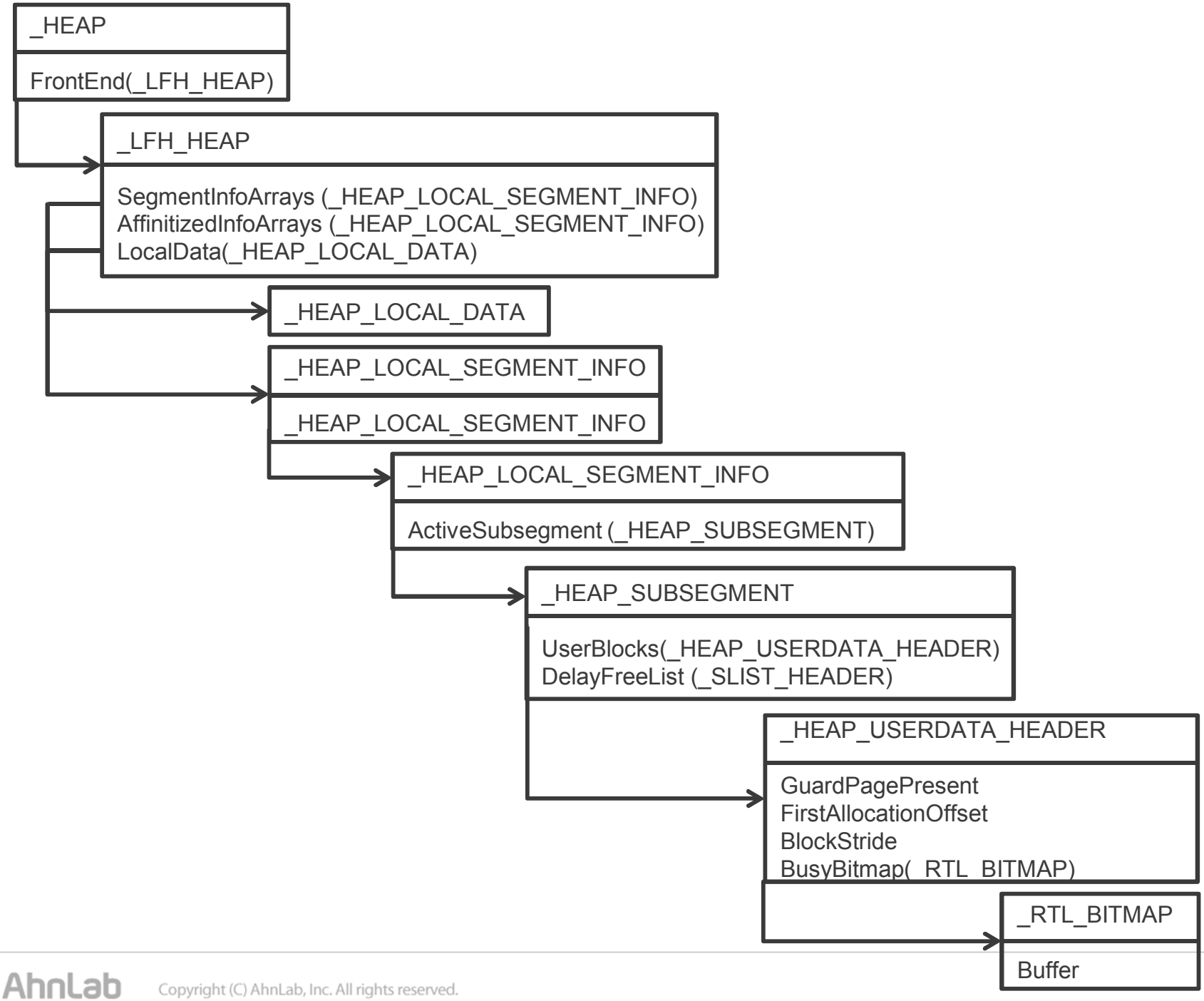
exploit code 진행 과정



Heap Structure – Windows 7



Heap Structure – Windows 8

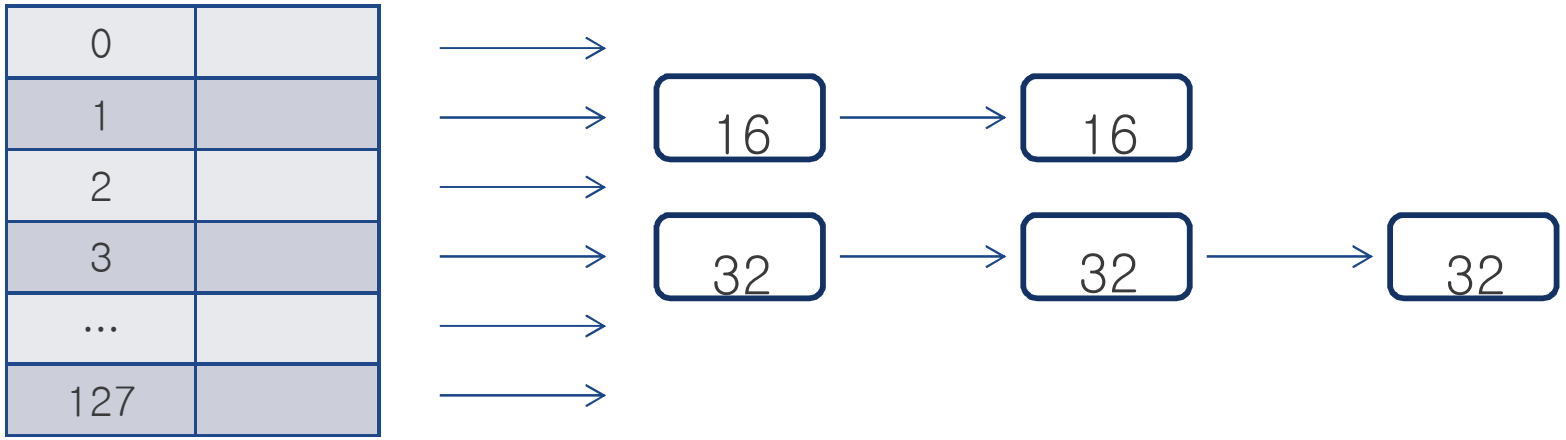


Heap Management

- Front End Manager
  - Memory 할당 요청을 효율적으로 처리하기 위해 활용
  - Free Heap블록을 사이즈 별로 관리
  - Look-aside Lists(활용되지 않음)와 Low-Fragmentation Heap으로 나눔
  - 사용자의 요청을 처리하지 못할 경우 Back End에서 관리함

❖ Front End Manager(cont)

◆ Look-Aside List



◆ Low Fragmentation Heap

Buckets	Granularity	Range
1-32	8	1-256
33-48	16	257-512
49-64	32	513-1024
65-80	64	1025-2048
81-96	128	2049-4096
97-112	256	4097-8192
113-128	512	8193-16384

❖ Front End Manager

_HEAP_ENTRY	<User Data>		_HEAP_ENTRY	<User Data>	
_HEAP_ENTRY	FreeEntryOffset	<User Data>	_HEAP_ENTRY	FreeEntryOffset	<User Data>
_HEAP_ENTRY	FreeEntryOffset	<User Data>	_HEAP_ENTRY	FreeEntryOffset	<User Data>
_HEAP_ENTRY	FreeEntryOffset	<User Data>	_HEAP_ENTRY	FreeEntryOffset	<User Data>

Improvement Heap Management

- Front End Manager
  - Windows 8 Randomization

_HEAP_ENTRY	<User Data>	_HEAP_ENTRY	<User Data>
_HEAP_ENTRY	<User Data>	_HEAP_ENTRY	<User Data>
_HEAP_ENTRY	<User Data>	_HEAP_ENTRY	<User Data>
_HEAP_ENTRY	<User Data>	_HEAP_ENTRY	<User Data>

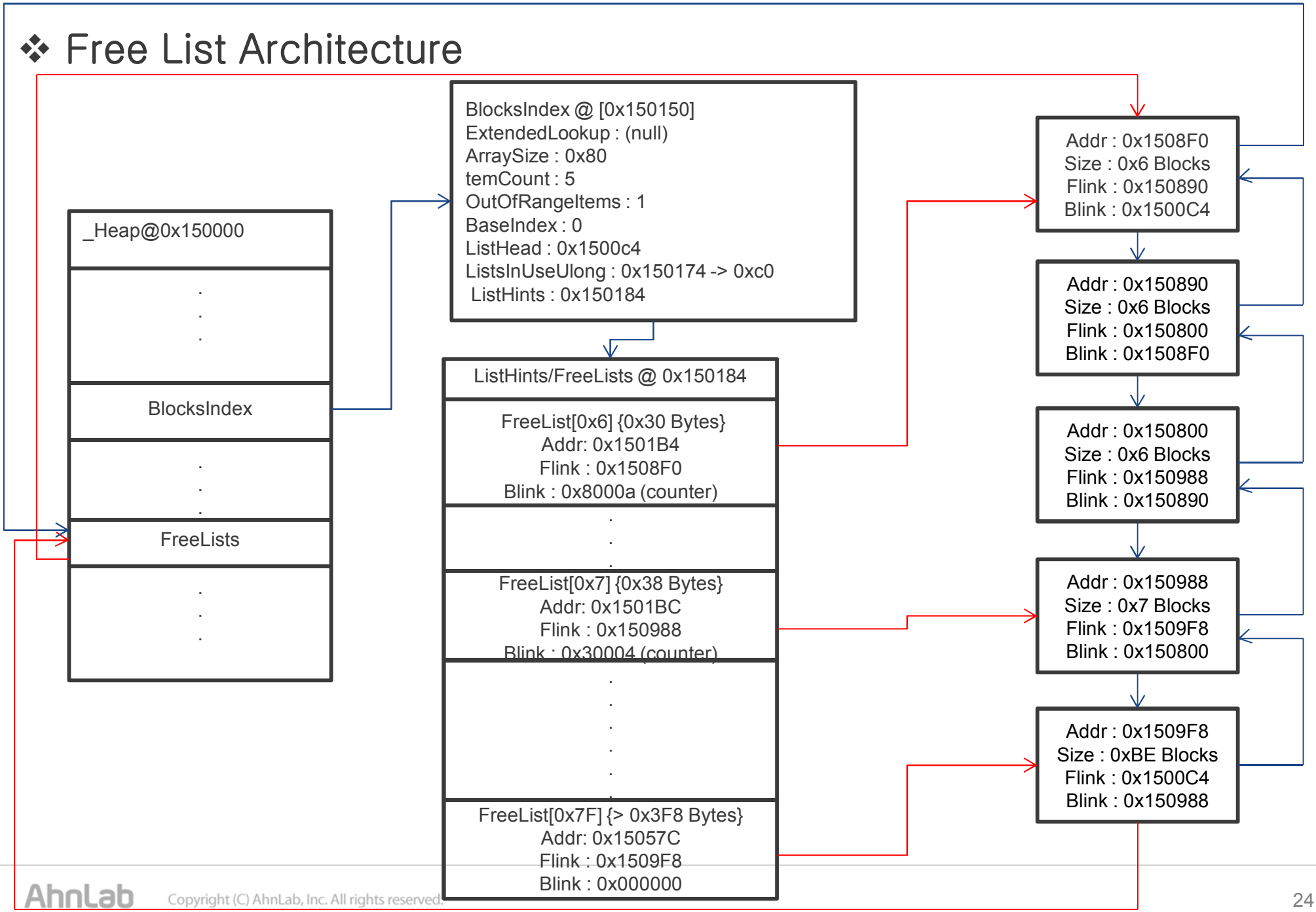
Improvement Heap Management

- Front End Manager
  - Windows 8 Guard page
    - UserBlock Overflow를 통한 공격을 방어하기 위해서 활용
    - Heap Memory의 Partition 역할을 함

Heap Management

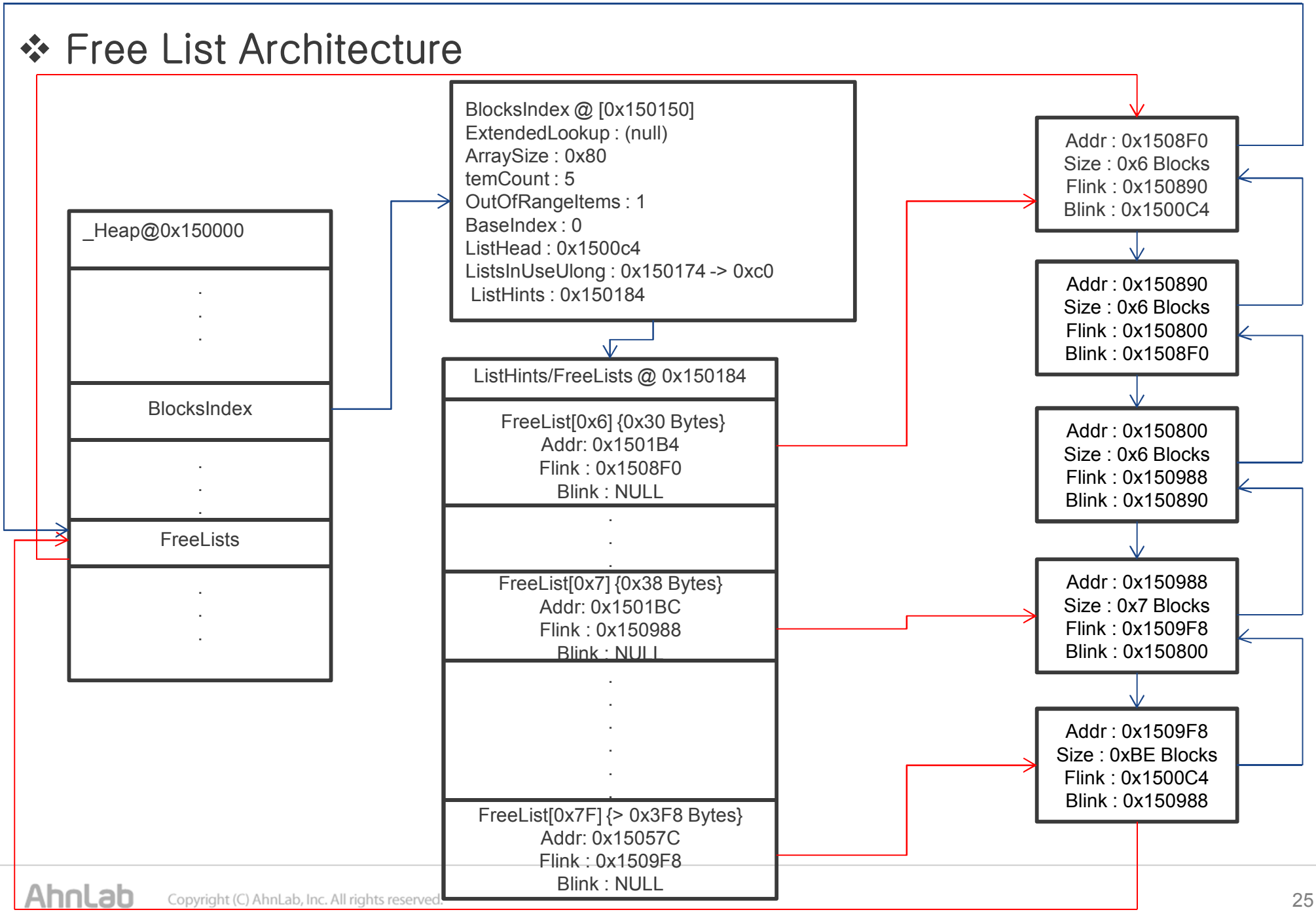
- Back End Manager
  - Front-End와 비슷하게 Free List라는 테이블을 통해서 Free Heap 관리
  - double linked list로 Free Heap Block을 관리
  - 특정 크기의 Free Heap Block을 포함하는 128개의 배열로 구성
  - 알맞은 크기의 Heap이 없을 경우 Block Splitting활용

❖ Free List Architecture



Improvement Heap Management

❖ Free List Architecture



Memory Allocation & Free Demo

AhnLab

03 Exploit Writing Technique

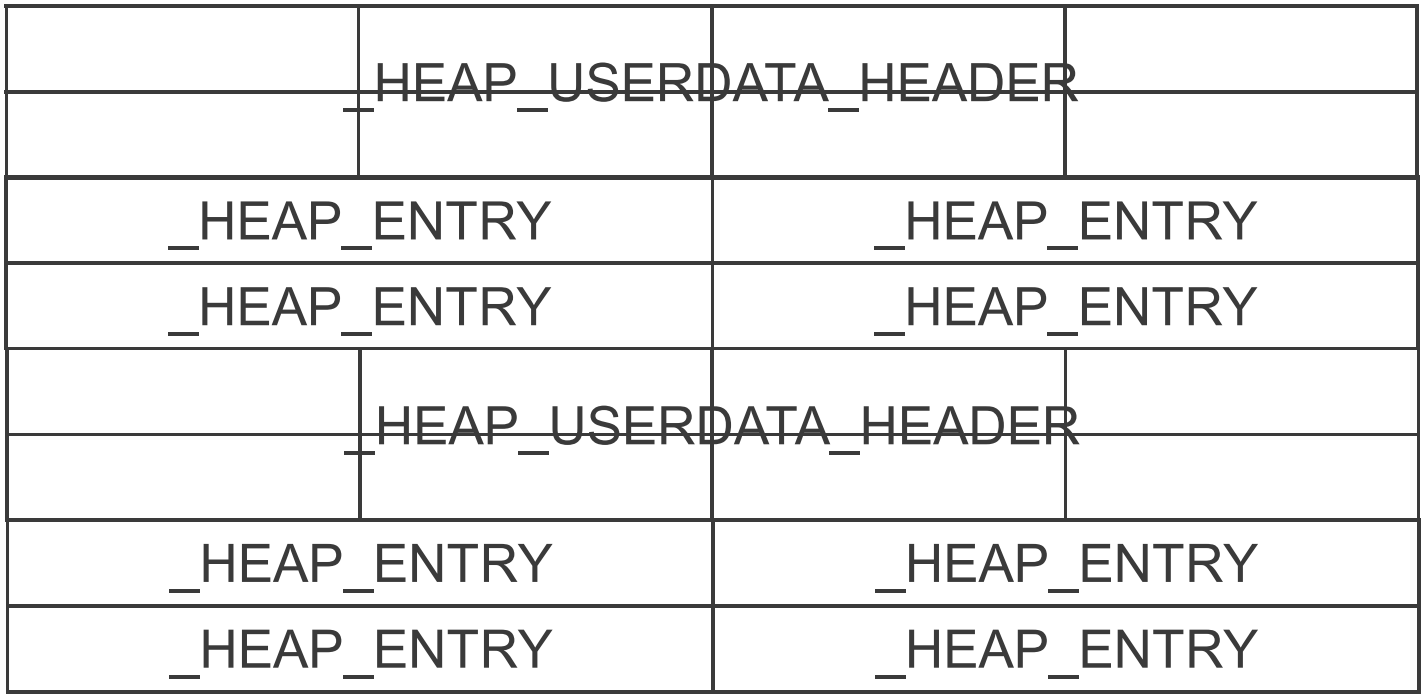
AhnLab

_HEAP_USERDATA_HEADER Attack

- UserBlocks Overwriting
  - FreeEntryOffset Attack을 더 이상 활용할 수 없음
  - _HEAP_USERDATA_HEADER Structure 공격을 목적으로 함
  - BlockStride는 각각의 Chunk 사이의 간격을 의미 함
  - FreeEntryOffset overwrite와 유사함

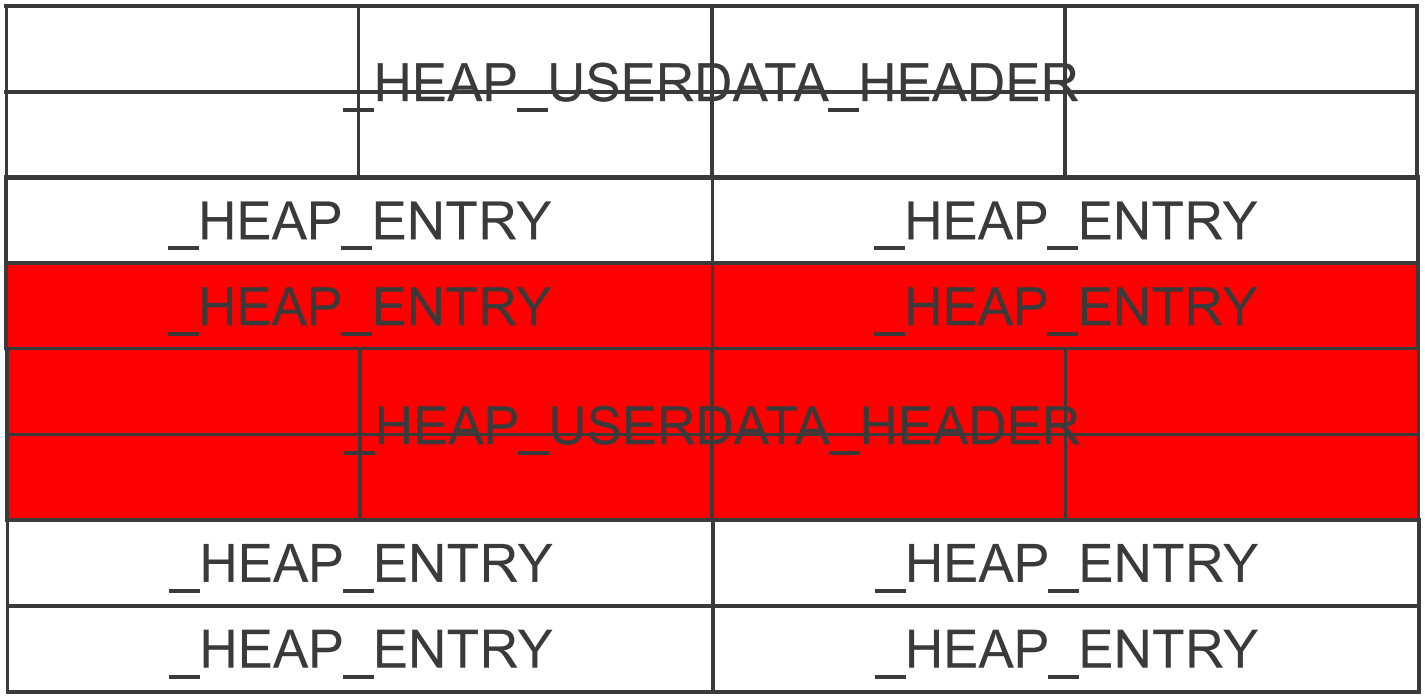
_HEAP_USERDATA_HEADER Attack

❖ UserBlocks Overwriting



_HEAP_USERDATA_HEADER Attack

❖ UserBlocks Overwriting



_HEAP_USERDATA_HEADER Attack

- Bitmap Flipping 2.0
  - _HEAP_ENTRY.PreviousSize member를 통하여 Bitmap값을 업데이트 함
  - UserBlocks->BusyBitmap->Buffer, Header->PreviousSize
  - PreviousSize값을 변조 하게 될 경우 영향을 받을 수 있음

_HEAP_USERDATA_HEADER Attack

❖ Bitmap Flipping 2.0

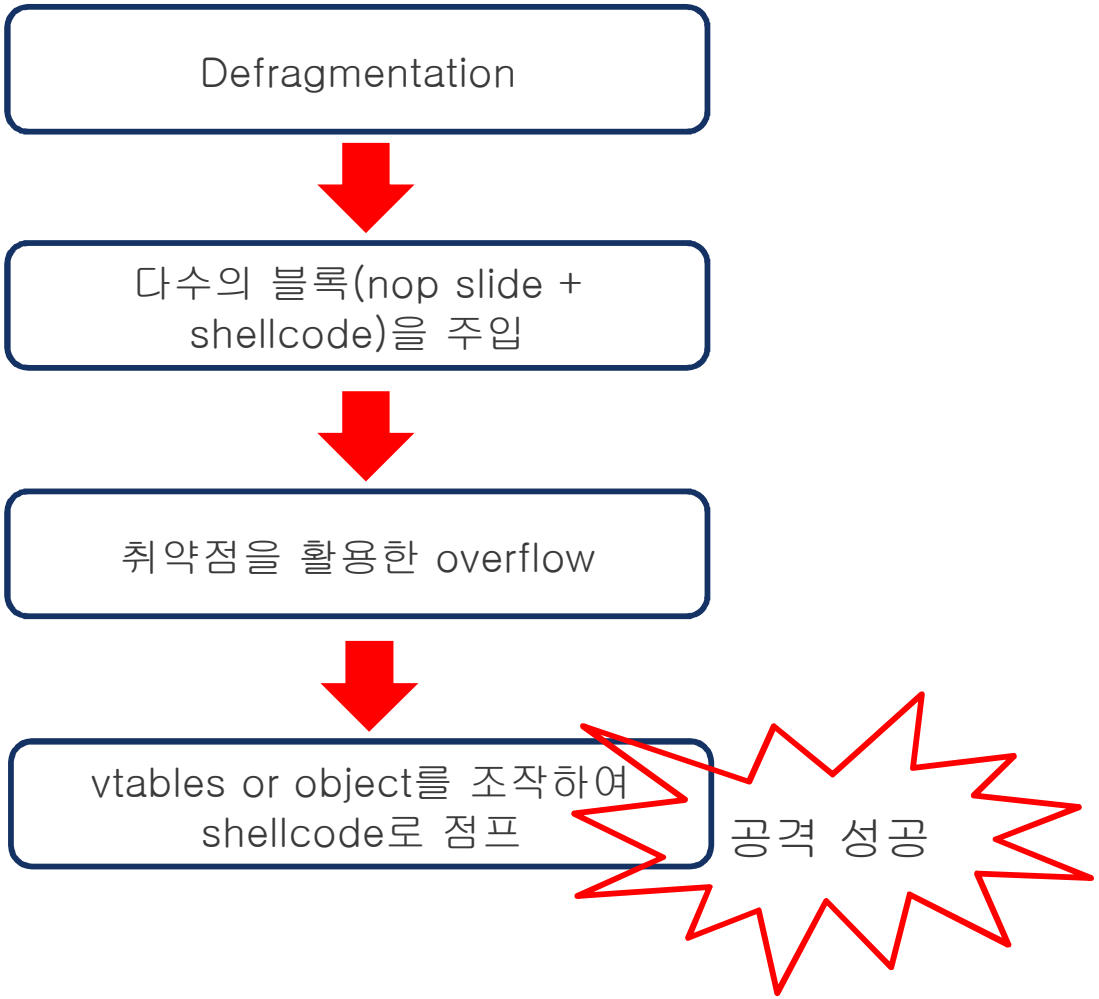
_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY
_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY
_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY

Exploit Strategy

- Heap Spray
  - 연속된 Heap 영역에 exploit code를 뿌리는 방법
  - 지속적으로 Heap을 할당 받아, 특정 메모리 주소 영역까지 데이터를 덮음
  - 대량의 nop slide와 shellcode로 제작된 block 생성 하여 heap에 저장

Exploit Strategy(cont)

❖ Heap Spray



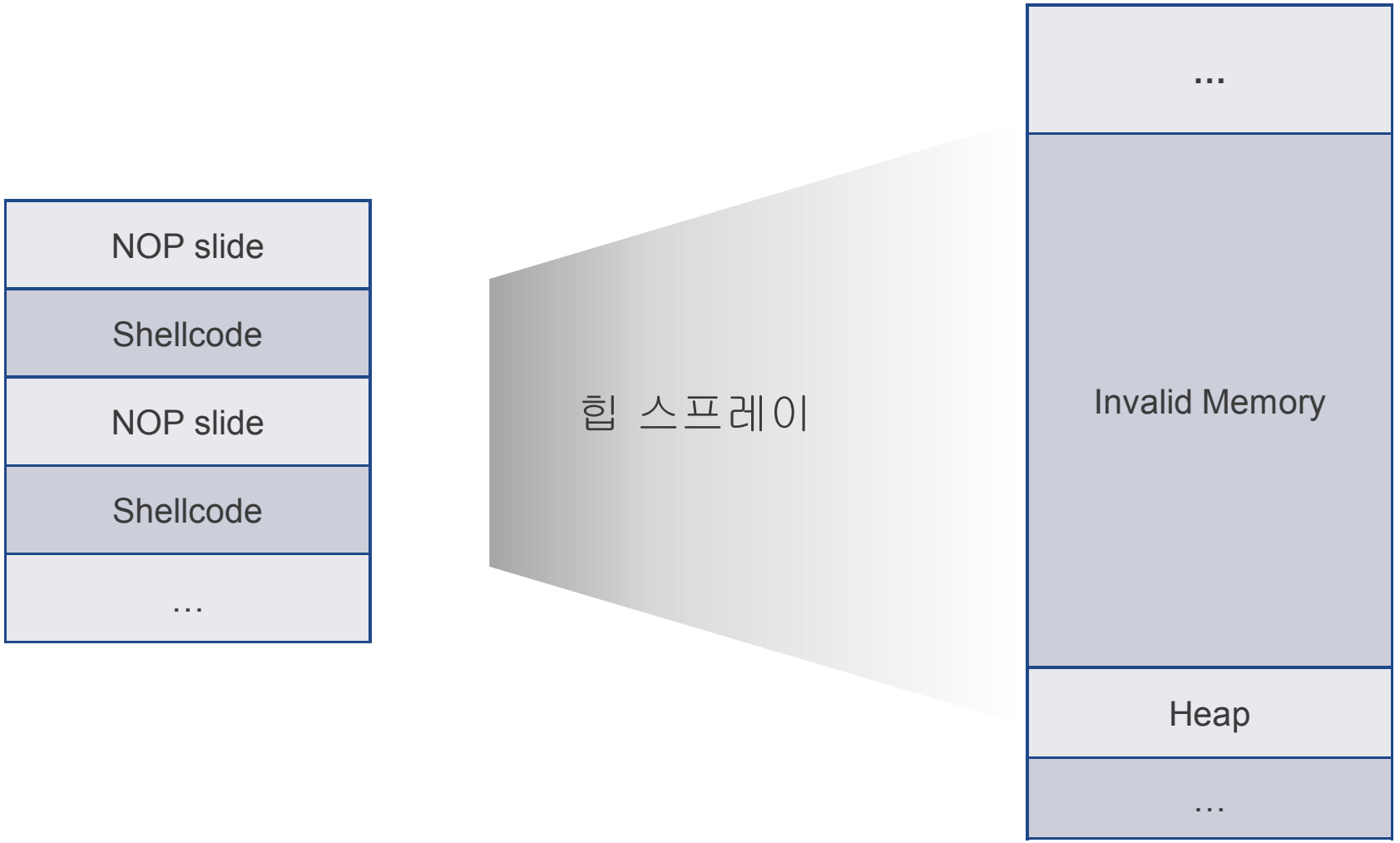
Exploit Strategy(cont)

- Heap Spray
  - Defragment Heap
    - Heap은 alloc/free를 반복하면서 단편화가 생기고, 이 때 다음 할당되는 Heap의 위치를 예상하는 것이 어려움
    - 연속적인 공간이 아닐 경우 EIP조작을 통해 점프 하더라도 공격 성공율이 낮음
    - NOP slide의 역할을 할 수 있으면서 주소값 으로 사용되었을 경우 조작된 Heap블록을 가리킬 수 있어야 하므로 다수의 Heap 블록 할당

```
var dummy = new Array(1000);

for(i=0; i<1000; i++){

dummy[i]= new Array(size);

}
```

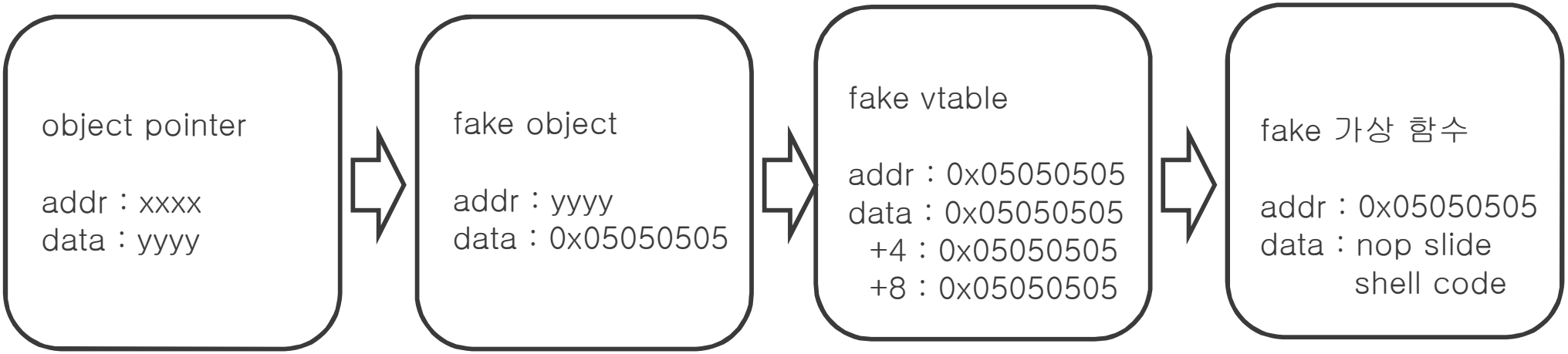
Exploit Strategy(cont)

- Heap Spray
  - NOP slide + shellcode 주입



Exploit Strategy(cont)

- Heap Spray
  - vtables or object 조작



❖ Heap Spray

◆ Heap Spray 공격 패턴

```
var shellcode = unescape("shellcode");

var block = unescape("%u0505%0505");

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

var x = new Array();

for (var i = 0; i < 200; i++) {

        x[i] = block + shellcode;

}
```

NOP slide
+
shellcode

특정 주소
지점까지
힙을 할당

Corelan Team – Heap Spray

- 기존 Use-After-Free 시나리오의 한계점
- DOM Element를 기반으로 한 Heap Spray 기법 제안
- 진행 과정
  - 페이지에 div 엘리먼트 삽입
  - 다수의 buttern 생성
  - 페이로드와 함께 title 속성 설정 및 div에 buttern 추가

Corelan Team – Heap Spray

❖ Sample Code

```
<!-- Corelan "DEPS" - Precise heap spray for FF/IE8/IE9/IE10 - corelanc0d3r www.corelan.be 2013 -->
<html>
<head></head>
<body>
<div id="blah"></div>
<script language = 'javascript'>
                var div_container = document.getElementById("blah");
                div_container.style.cssText = "display:none";
                var data;
                offset = 0x104;
                junk = unescape("%u2020%u2020");
                while (junk.length < 0x1000) junk += junk;

                rop =unescape("%u4141%u4141%u4242%u4242%u4343%u4343%u4444%u4444%u4545%u4545%u4646%u4646%u4747%u4747");
                shellcode = unescape("%ucccc%ucccc%ucccc%ucccc%ucccc%ucccc%ucccc%ucccc");
                data = junk.substring(0,offset) + rop + shellcode
                data += junk.substring(0,0x800-offset-rop.length-shellcode.length);

                while (data.length < 0x80000) data += data;
                // Targets:
                // FireFox : 0x20302210
                // IE 8, 9 and 10 : 0x20302228
                for (var i = 0; i < 0x500; i++)
                {
                                var obj = document.createElement("button");
                                obj.title = data.substring(0,0x40000-0x58);
                                div_container.appendChild(obj);
                }
                alert("spray done");
</script>
</body>
</html>
```

Heap Spray In Windows 8 Demo

AhnLab

?

AhnLab

thank you.

CodeEngn

www.CodeEngn.com

2013 CodeEngn Conference 08

AhnLab