Security in OS & Where to use OS Principles

Özgür Saygın Bican

MilSOFT Software Technologies

Outline

Buffer Overflow

ROP and mitigations

MemProtection & story of it

Virtualization

Where to use OS principles

Autobiography

BS 2013 METU CENG

Minor from Psychology

MS 2015 METU CENG

Thesis Topic : Static Binary Rewriting

Working at MilSOFT since graduation

Senior Software Engineer / Cyber Security Team Leader

Procedure Control Flow

Use **stack** to support procedure call and return

```
804854e:   e8 3d 06 00 00   call   8048b90 <main>
8048553:   50               pushl  %eax
```

Procedure call: `call label`

- Push **return address** on stack (`0x8048553`)
- Jump to `label`

Return address: Address of instruction beyond `call`

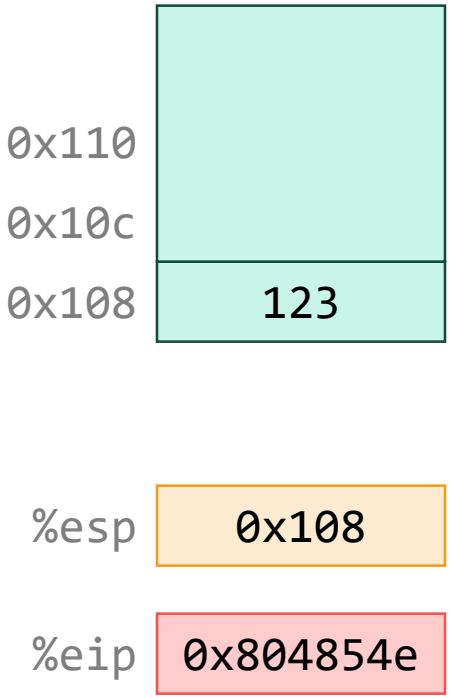
Procedure return: `ret`

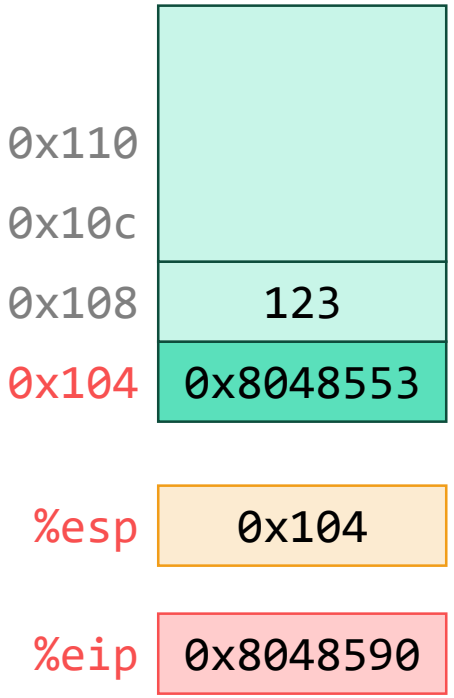
- Pop return address from stack
- Jump to address

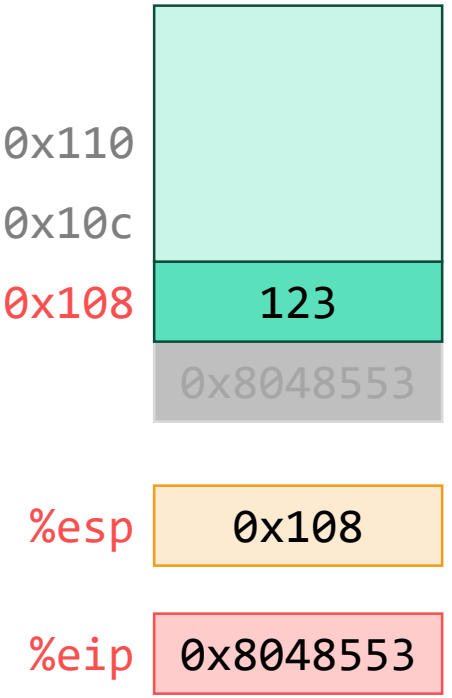
Procedure Call Example

```
804854e:  e8 3d 06 00 00    call   8048b90 <main>
8048553:  50                pushl  %eax

8048590:  ...               ...
8048591:  c3                ret
```







IA32/Linux Stack Frame

Current Stack Frame ("Top" to Bottom)

'Argument Build': Parameters for function being called

Local variables (that cannot fit in registers)

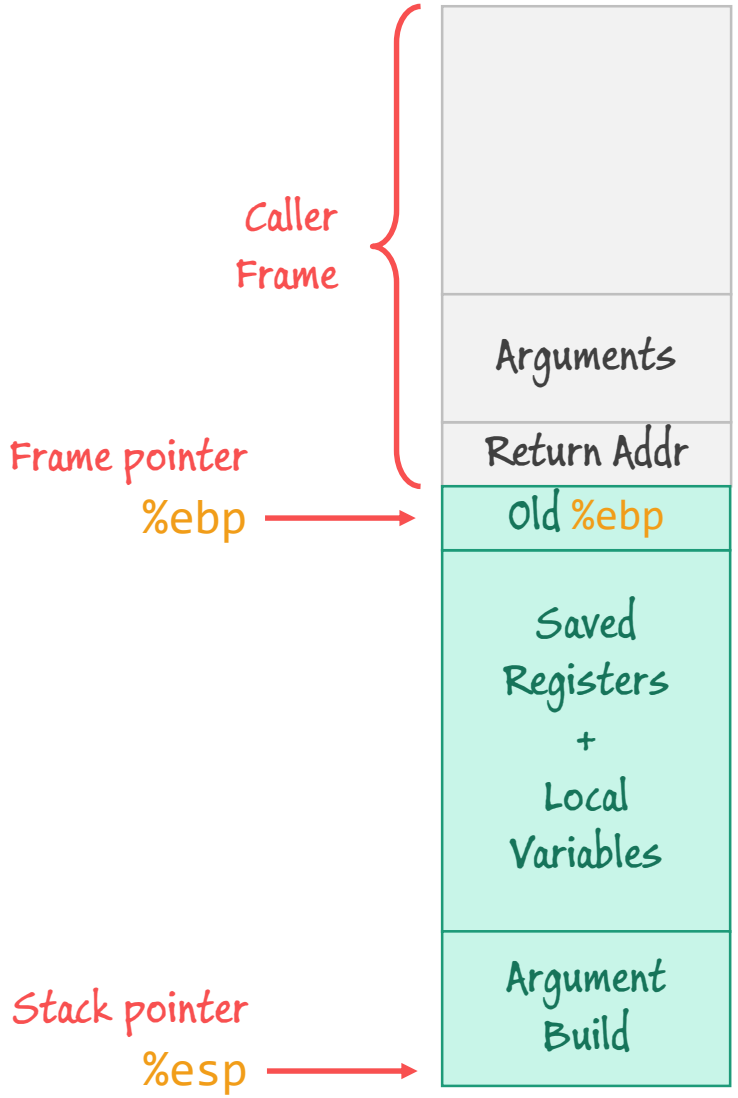
Saved register context

Old frame pointer

Caller Stack Frame

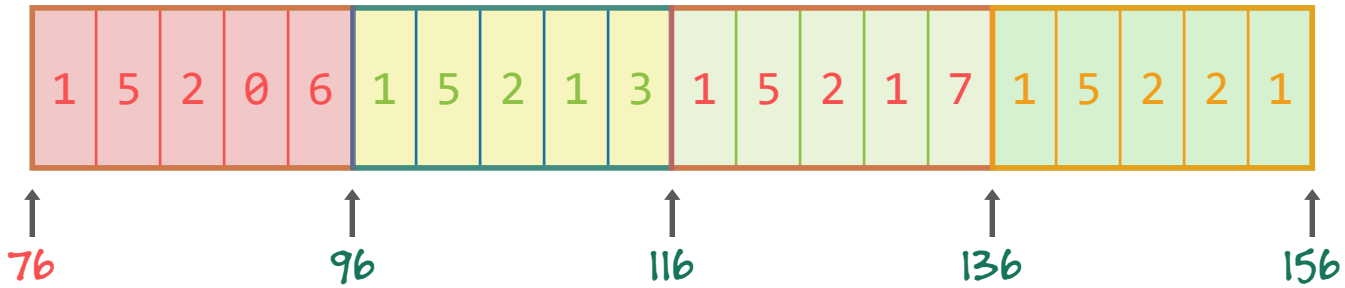
Return address (Pushed by `call` instruction)

Arguments for this call



Strange Referencing Examples

```
zip_dig pgh[4];
```



Reference	Address	Value
pgh[3][3]	76+4·(5·3+3) = 148	2
pgh[2][5]	76+4·(5·2+5) = 136	1
pgh[2][-1]	76+4·(5·2+-1) = 112	3
pgh[4][-1]	76+4·(5·4+-1) = 152	1
pgh[0][19]	76+4·(5·0+19) = 152	1
pgh[0][-1]	76+4·(5·0+-1) = 72	??

- Code **does not** do any bounds checking

- Ordering of elements **within** array guaranteed

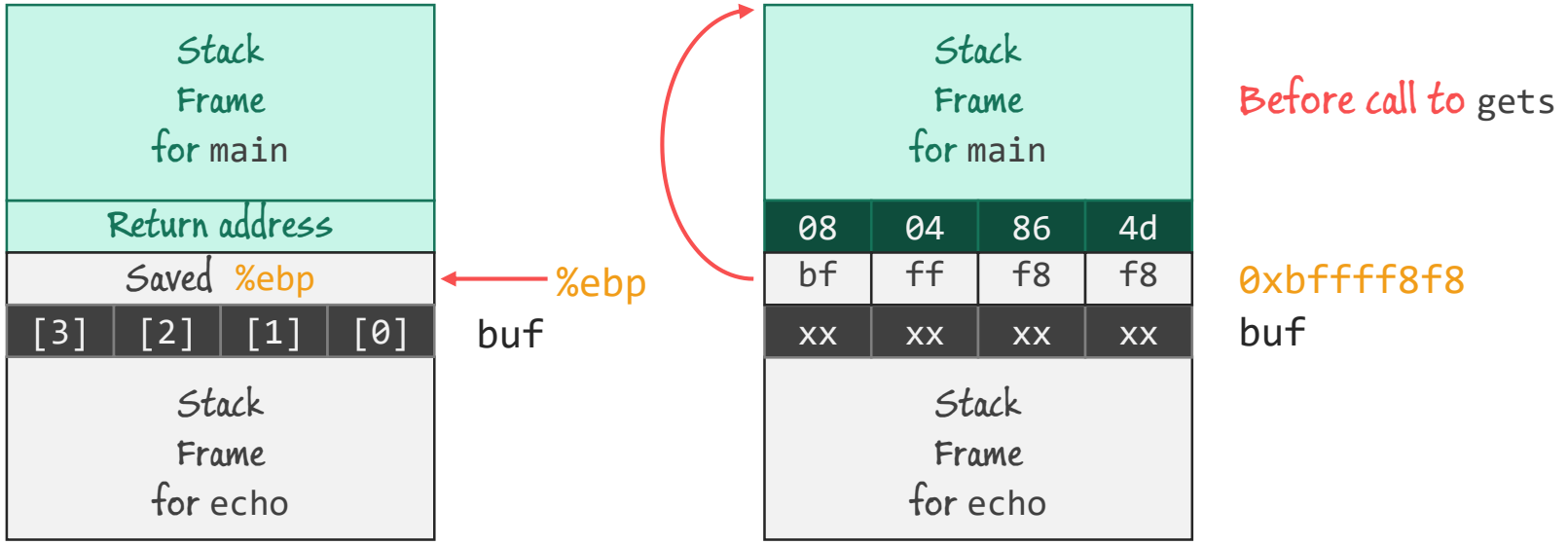
Buffer Overflow

```
/* Echo Line */
void echo () {
    char buf[4]; /* Way too small! */
    gets (buf);
    puts (buf);
}
```

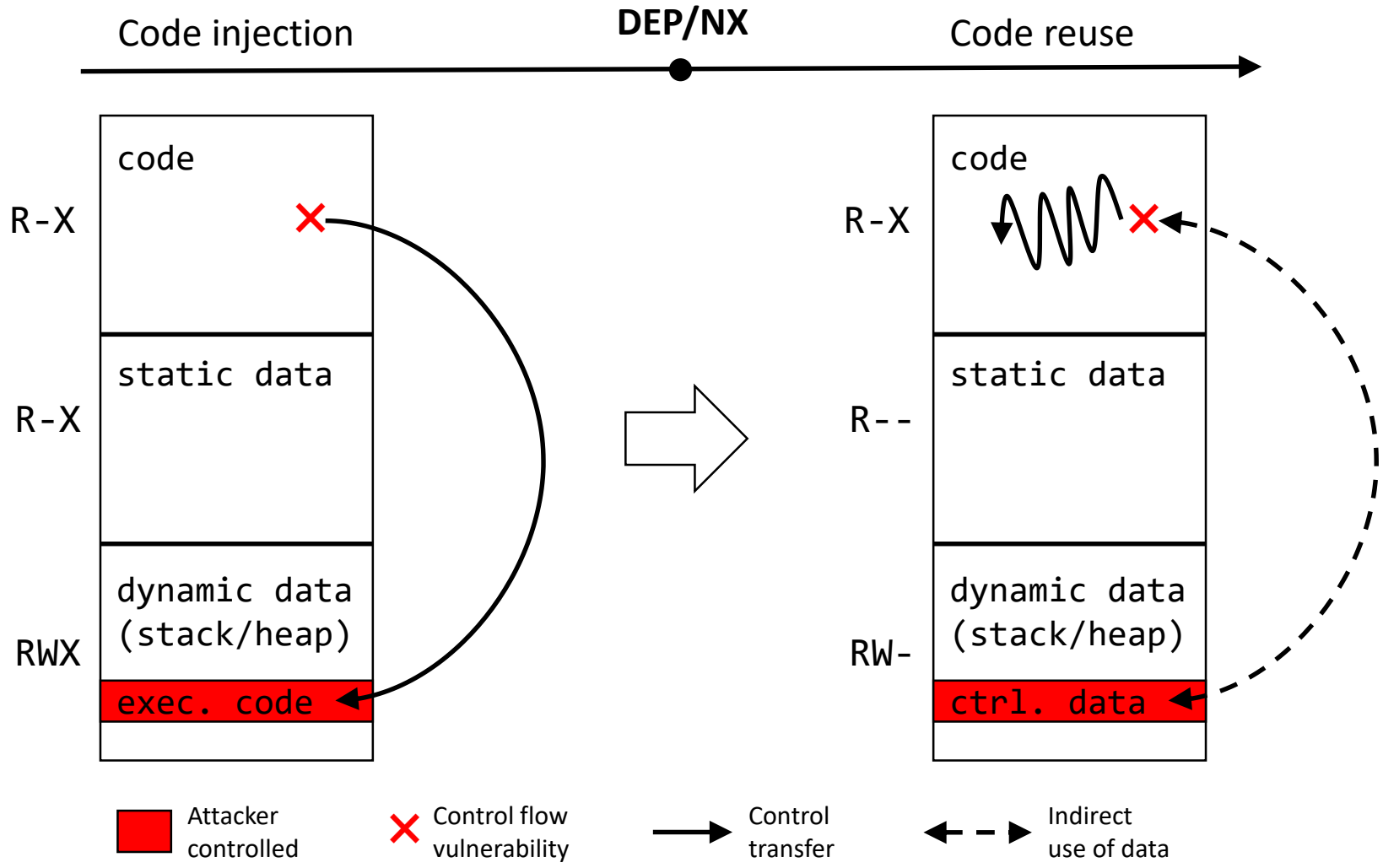
```
8048648:    call 804857c <echo>
804864d:    mov  0xfffffff8(%ebp),%ebx
```

Return point

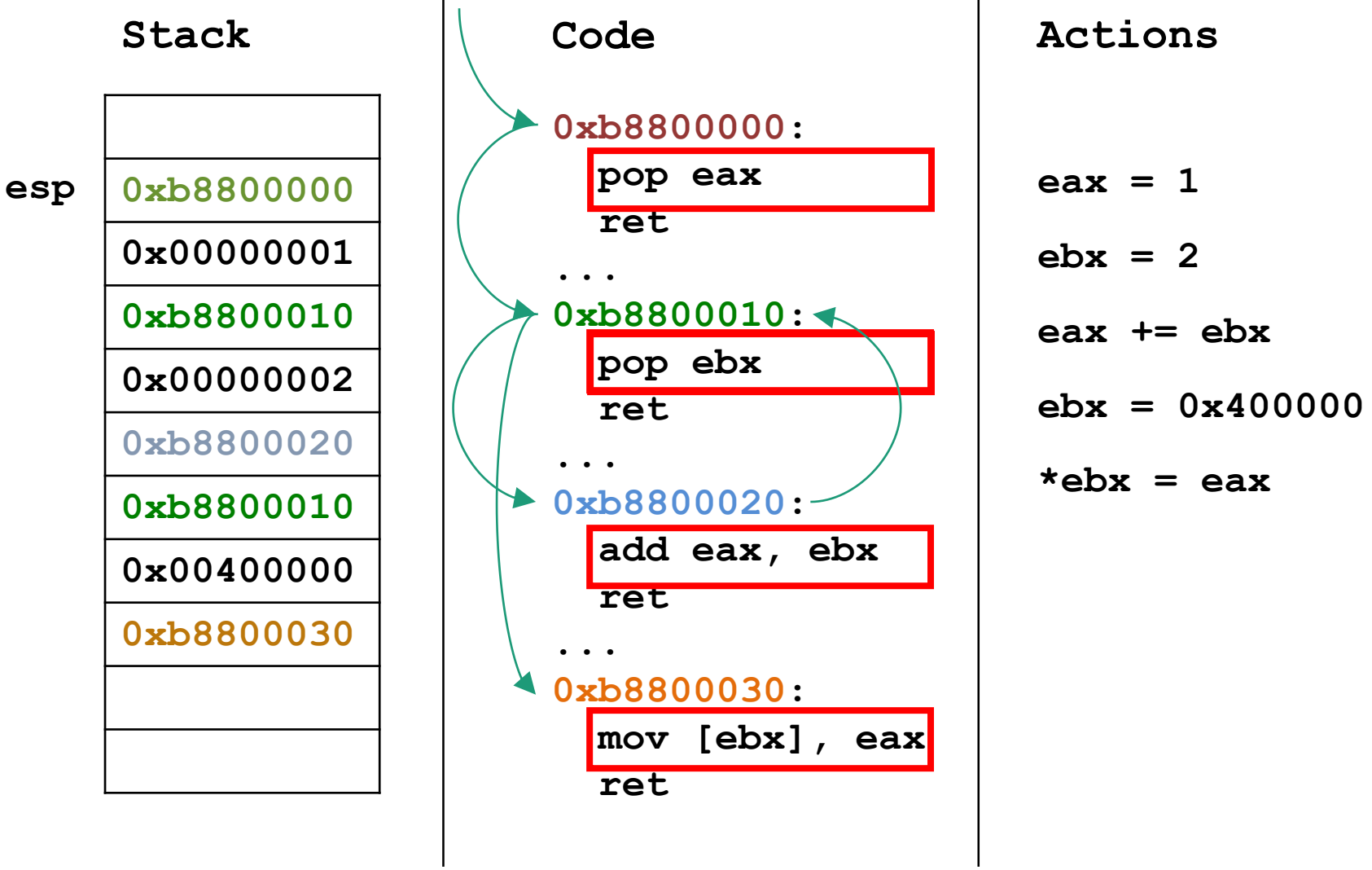
```
unix> gdb bufdemo
(gdb) break echo
Breakpoint 1 at 0x8048583
(gdb) run
Breakpoint 1, 0x8048583 in echo ()
(gdb) print /x *(unsigned *)$ebp
$1 = 0xbffff8f8
(gdb) print /x *((unsigned *)$ebp + 1)
$3 = 0x804864d
```



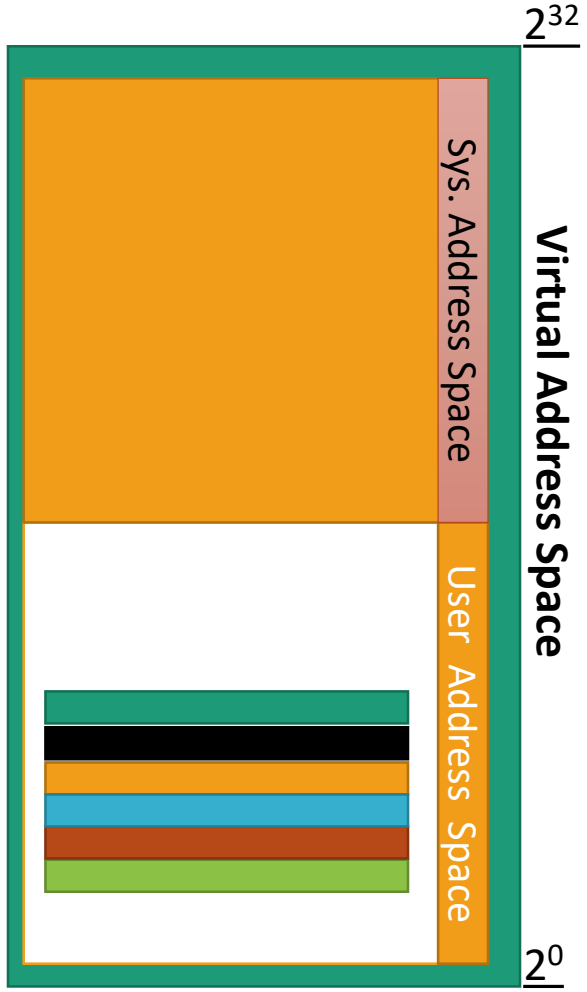
DEP and ROP


Code injection
DEP/NX
Code reuse
code
R-X
static data
R-X
dynamic data
(stack/heap)
RWX
exec. code
code
R-X
static data
R--
dynamic data
(stack/heap)
RW-
ctrl. data
Attacker controlled
Control flow vulnerability
Control transfer
Indirect use of data

Return-Oriented Programming

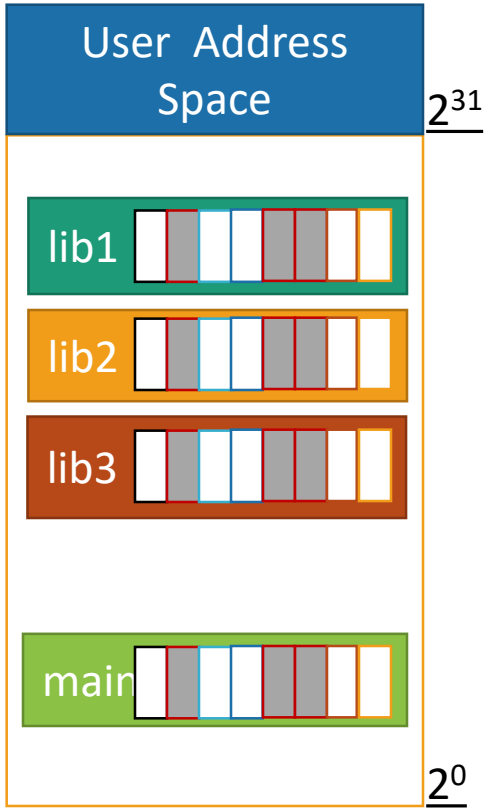


RoP Defenses: ASLR



- ASLR randomizes the image base of each library
  - Gadgets hard to predict
  - Brute force attacks still possible

RoP Defenses: IPR / ILR



- Instruction Location Randomization (ILR)
  - Randomize each instruction address using a virtual machine
  - Increases search space
  - Cannot randomize all instructions
  - High overhead due to VM (13%)
- In-place Randomization (IPR)
  - Modify assembly to break known gadgets
  - Breaks 80% of gadgets on average
  - Cannot remove all gadgets
  - Preserves gadget semantics
  - Deployment issues

Bug Classes

Stack Overflow

Integer Overflow

Heap Overflow

Use After `free()`

Unitialized Variables

Race Conditions

Patchguard

Critical Data Structures in kernel are Tamper Proofed

Check data multiple times from different data structures

Space utilization decreases

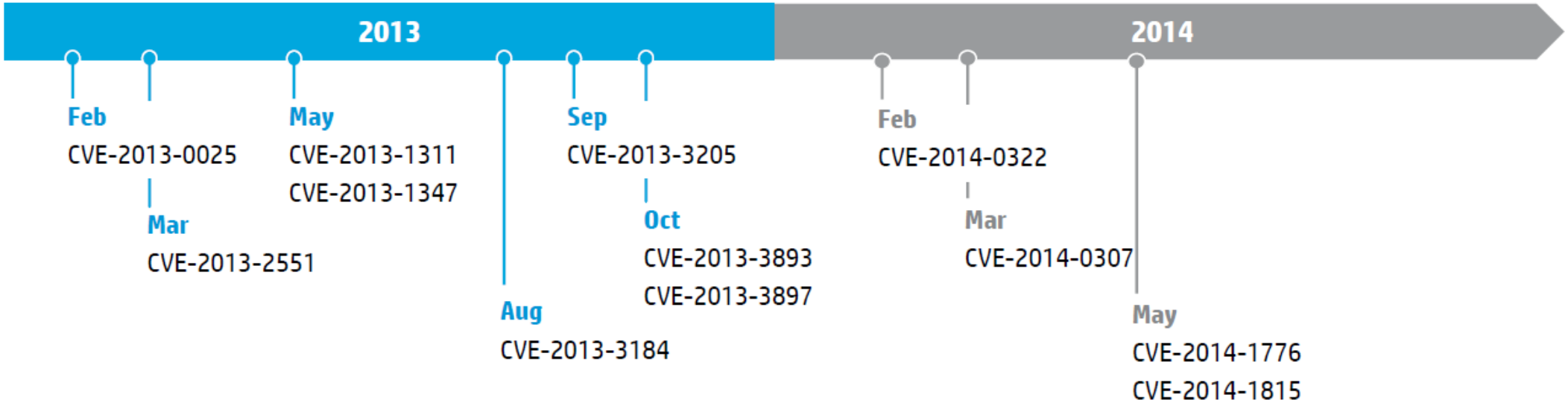
Internals were supposed to be Top Secret

Reverse Engineered

Reversers are hired afterwards, improving patchguard

MemProtection

Use-After-Free Vulnerabilities



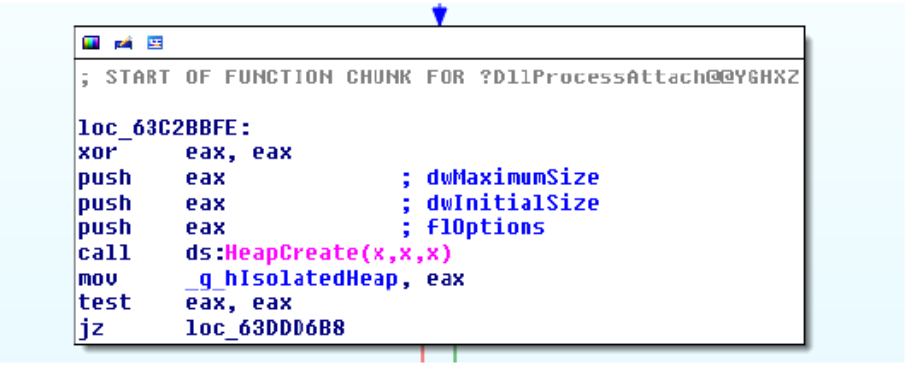
MemProtection

What is next?

It is finally getting harder!

Microsoft Security Bulletin MS14-035 – Critical

Cumulative Security Update for Internet Explorer (2969262)

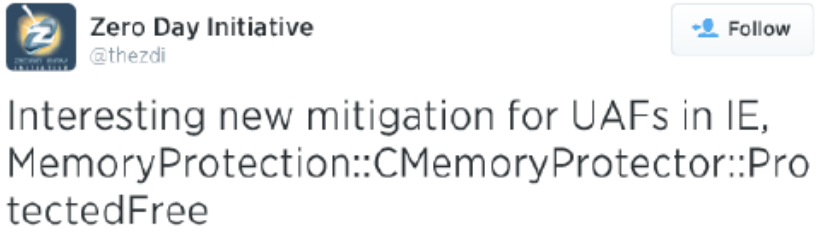


```
; START OF FUNCTION CHUNK FOR ?DllProcessAttach@@YGHXZ

loc_63C2BBFE:
xor     eax, eax
push    eax             ; dwMaximumSize
push    eax             ; dwInitialSize
push    eax             ; flOptions
call    ds:HeapCreate(x,x,x)
mov     _g_hIsolatedHeap, eax
test    eax, eax
jz      loc_63DDD6B8
```

Microsoft Security Bulletin MS14-037 – Critical

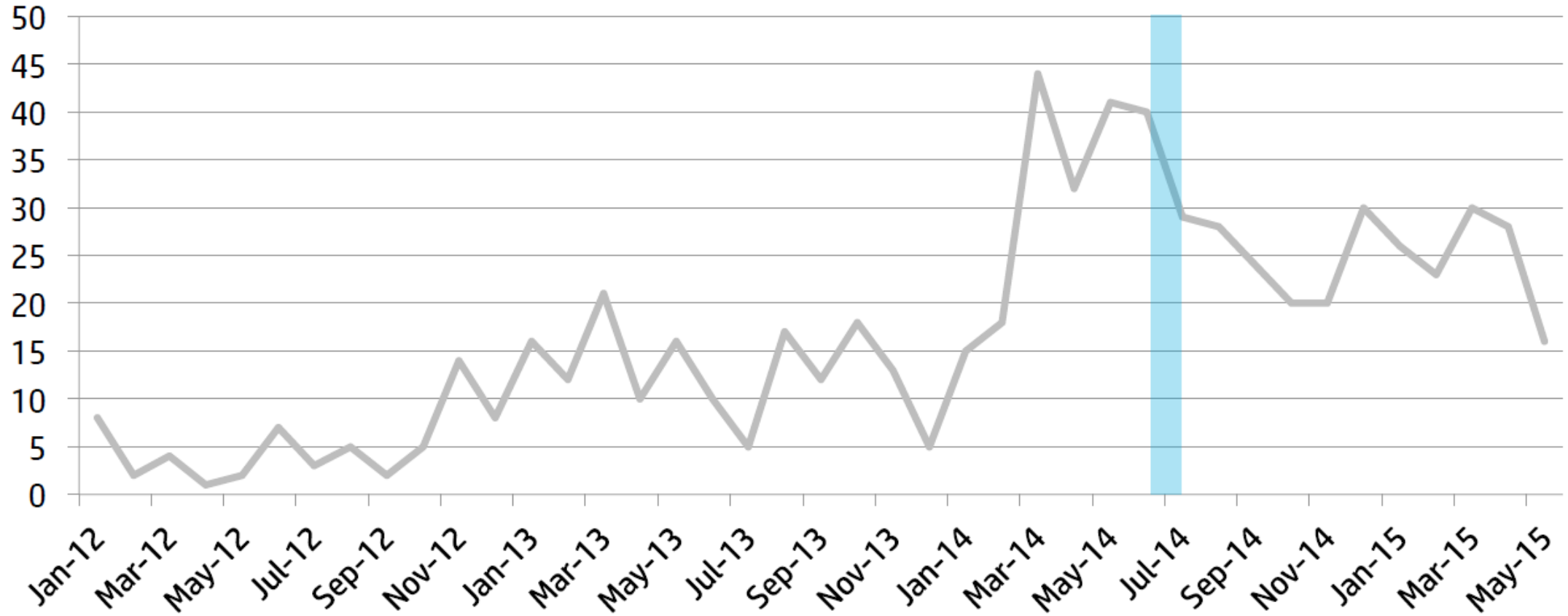
Cumulative Security Update for Internet Explorer (2975687)



MemProtection

ZDI Internet Explorer Submission Trends

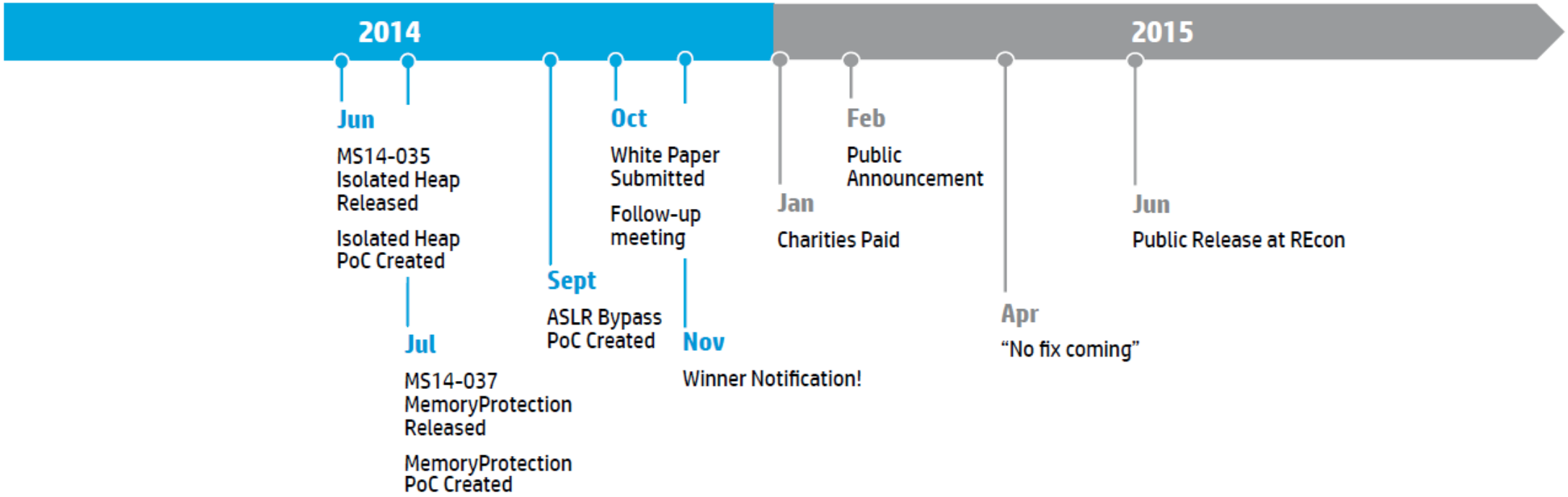
Impact of Microsoft's Mitigations



MemProtection

Research Timeline

From Mitigation Release to Public Release



MemProtection

Weaknesses and attack scenarios

Isolated Heap does not keep track the object types

Type confusion possible

Attacker can overwrite an isolated freed object with smaller / bigger objects

Make use of the the confusion/size weaknesses

Highly dependent on the offset being dereferenced from the freed object

MemProtection

What is MemProtection?

Prevent memory blocks from being deallocated while being referenced

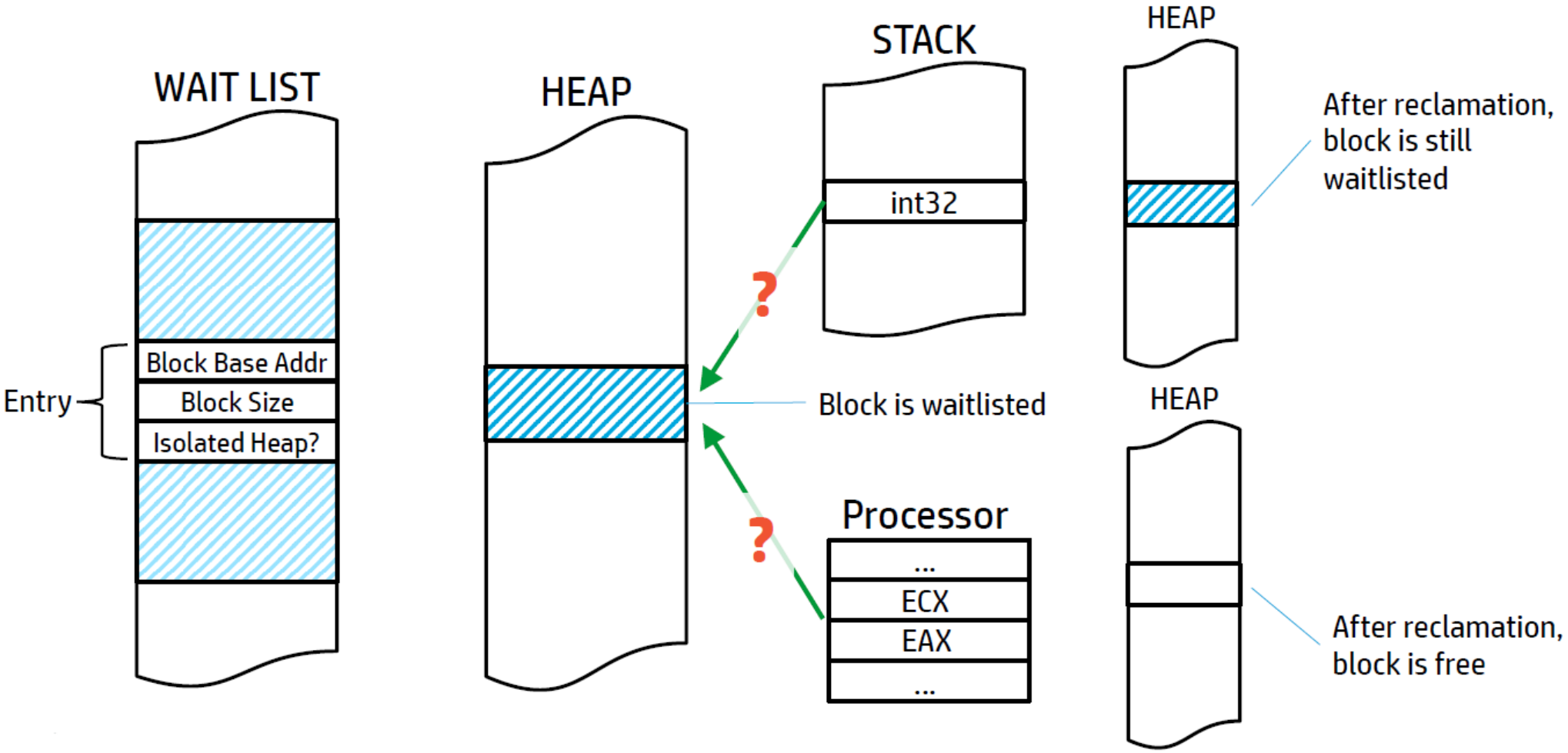
First Release: Checks for references on the stack

Second Release: Added checks for references in processor registers

`ProtectedFree` instead of `HeapFree`

Adds block to waiting list to be feed

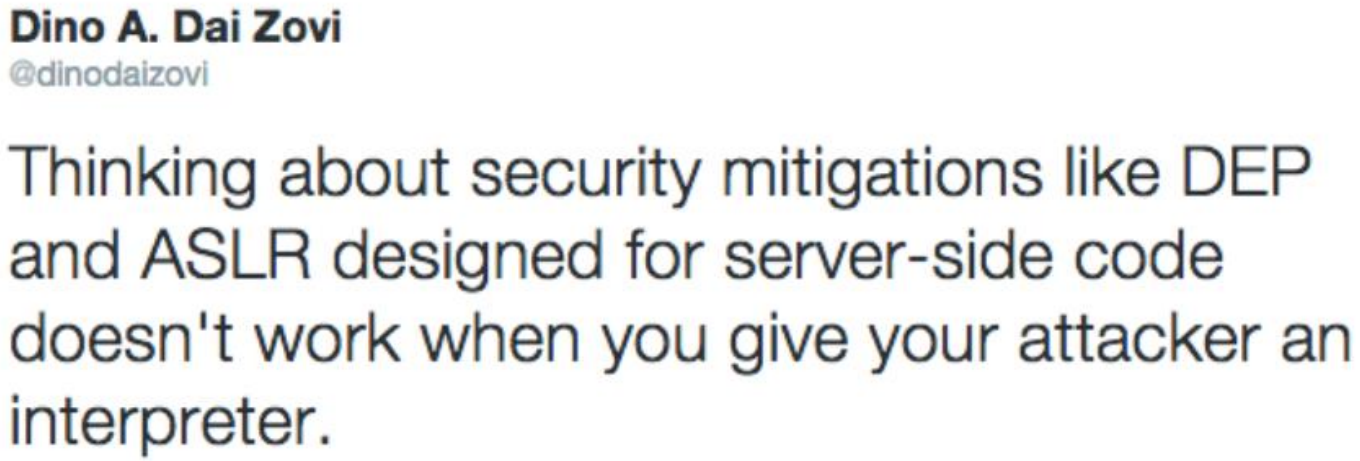
MemProtection



Recommendations from Attackers

Remove MemoryProtection from array and buffer allocations

Strenghten ASLR by performing an entropy check at module load time



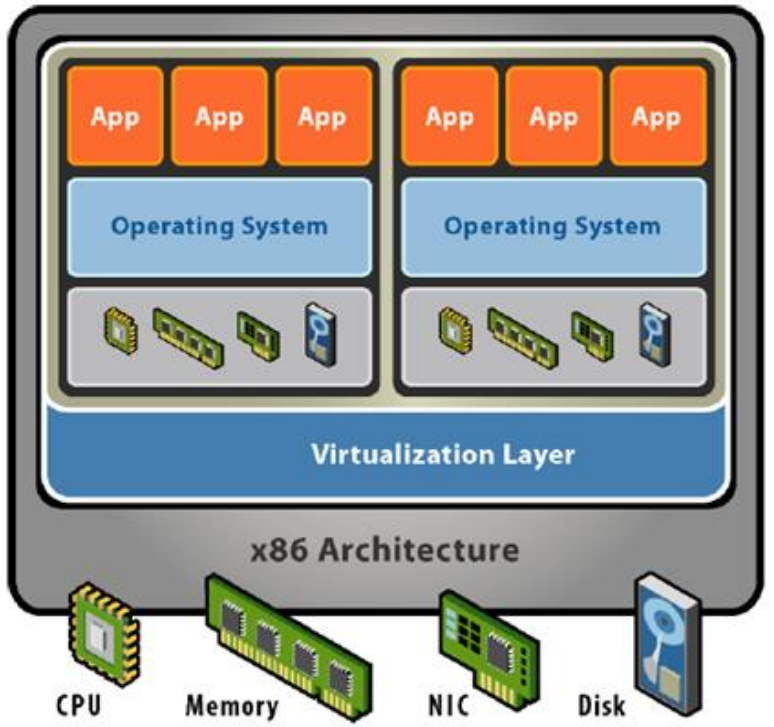
More info @ Blackhat 2015, '*Abusing Silent Mitigations*' presentation

More Isolation

Virtualization

Separation from physical layer

Terms: Hypervisor/Virtual Machine Monitor



Virtualization

Advantages

Security

Still not a silver bullet!

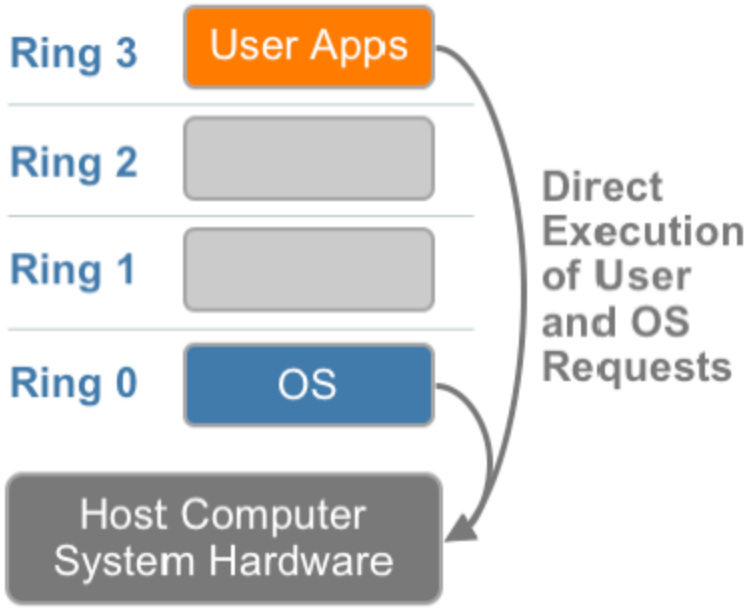
Portability

Scalability

Utilization (ex: cloud providers)

CPU Virtualization

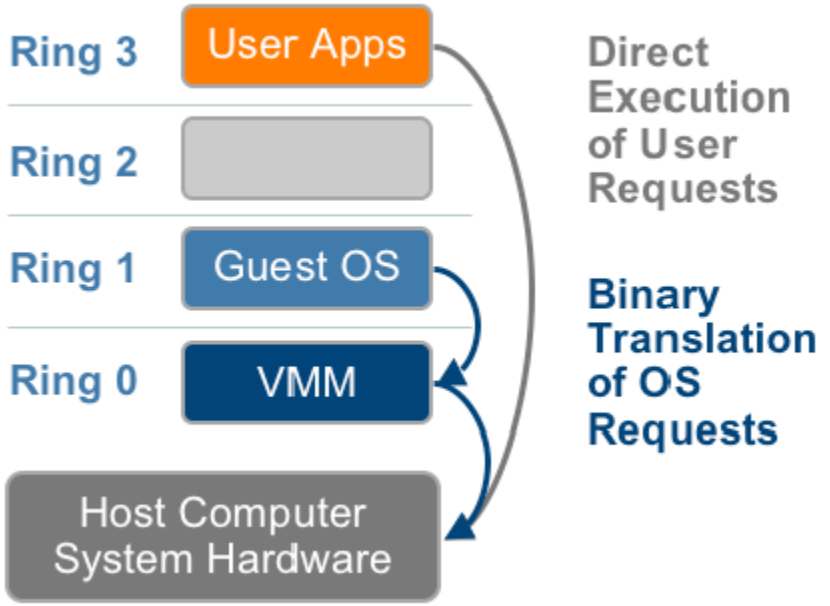
Challenges and Types



CPU Virtualization

Challenges and Types

Full Virtualization using Binary Translation

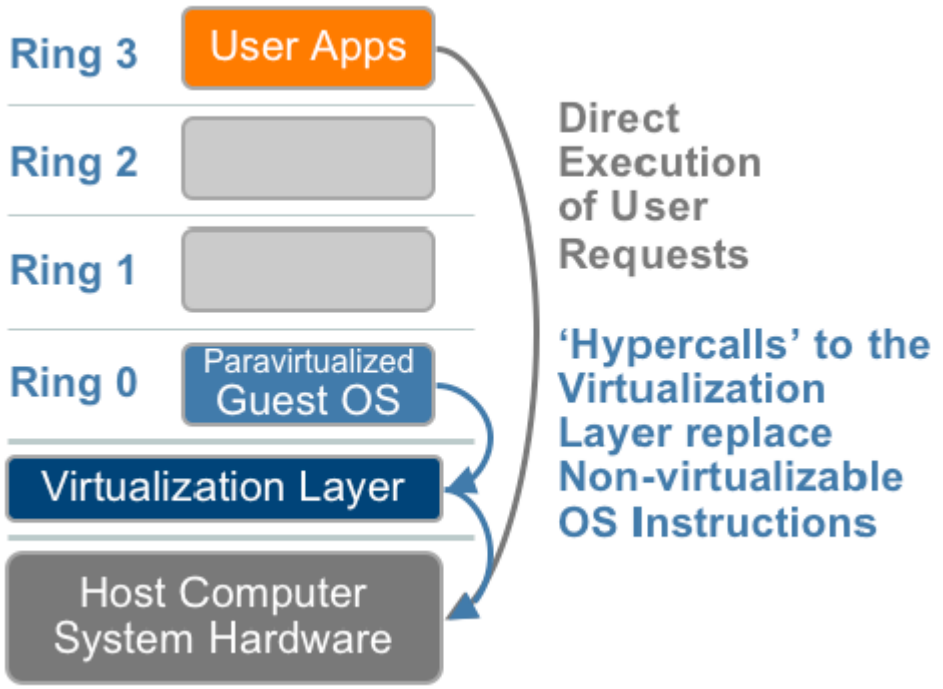


CPU Virtualization

Challenges and Types

Full Virtualization using Binary Translation

Paravirtualization (OS-Assisted)



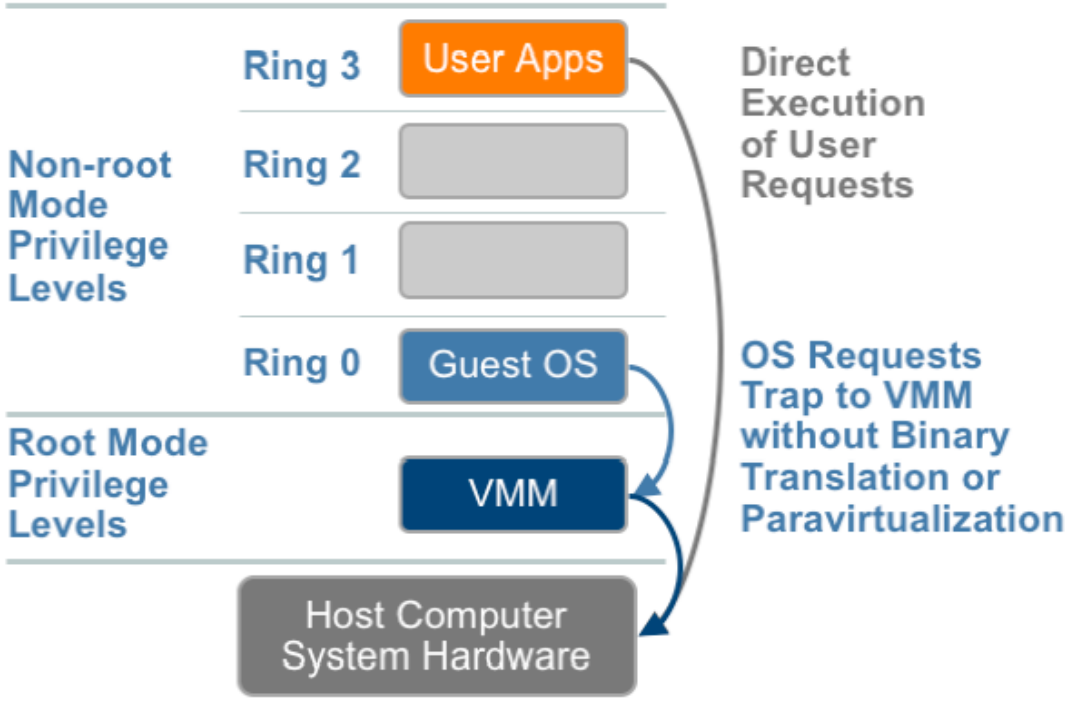
CPU Virtualization

Challenges and Types

Full Virtualization using Binary Translation

Paravirtualization (OS-Assisted)

Hardware-assisted

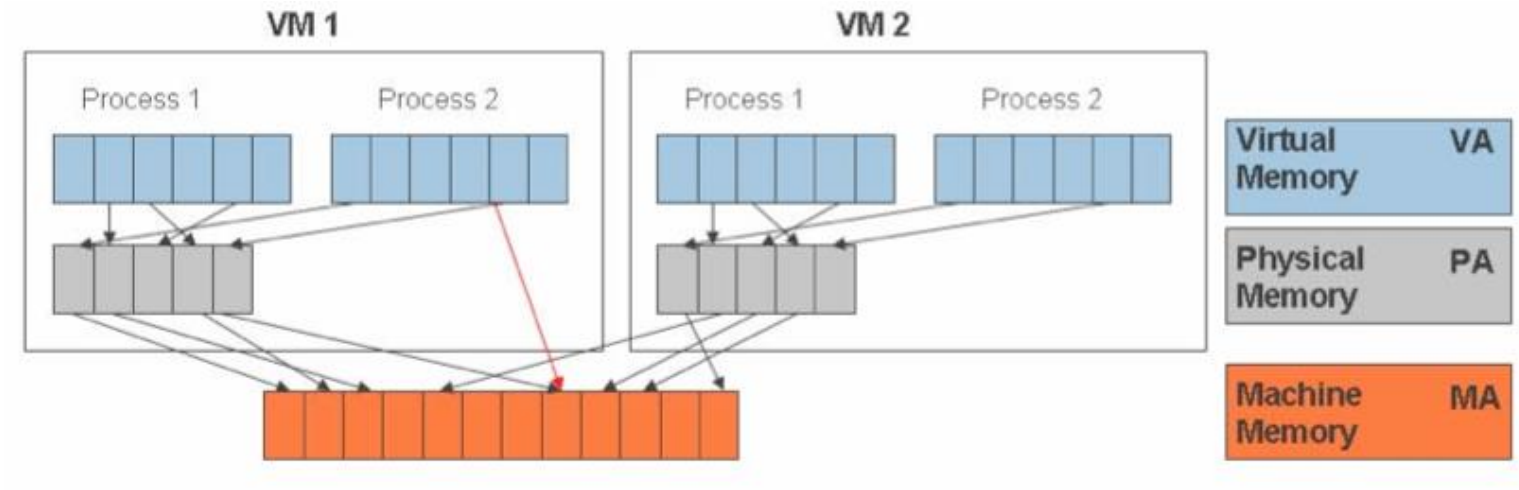


Memory Virtualization

Multiple VMs on a single system

- MMU is not enough
- Another level of memory virtualization: **VMM**

  Virtualize the MMU



Where to use OS principles

Multi-threaded applications

Bugs are hard to reproduce: you may not catch them during tests but the customer probably will

Performance: you should minimize the lock usage

Virtual Memory & Memory Hierarchy

Performance: register vs memory |cache mechanism, TLB | page in/out etc.

Debugging

Interprocess communication and synchronization

Single process software is unlikely

User space & kernel space differences

Case Study

Turkish Air Force and NATO Communication

Data Loss Prevention

Content Filtering

Must be transparent in network connection

Performance, performance, performance (go beyond algorithmic complexity)

Used skills so far (in 2 months) (1/2)

Turkish Air Force and NATO Communication

Network stack know-how

TCP, UDP, Application Layer Protocols

Kernel module development

No complex data structures, memory restrictions, performance

Inter process communication

Sockets, memory-map, shared memory etc.

Used skills so far (in 2 months) (2/2)

Turkish Air Force and NATO Communication

Kernel – user space synchronization

Watch out for deadlocks, locking is dangerous

Other OS principles

Multi-threading, synchronization, OS architecture

Performance optimization

Cache friendly code, profiling

C and Java Development

No STL, implement your own fast & lightweight data structures, algorithms