Ethical hacking

László Erdődi, PhD.

University of Agder

laszlo.erdodi@uia.no

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 Ethical Hacking (EC Council – Certified Ethical Hacker)
- Courses on exploit writing (hardcore hacking)

Schedule

- Wednesday 10.30-12.30 Ethical hacking in general, practical tricks
- Friday 10.30-12.30 Research on memory corruption

What is ethical hacking?

- Legal (contract)
- Promote the security by showing the vulnerabilities
- Find all vulnerabilities
- Without causing harm
- Document all activities
- Final presentation and report about the vulnerabilities

- Illegal
- Steal information, modify data (e.g. deface), make service unavailable
- Find only the weakest link to achieve the aim
- Do not care if the action destroys the system
- Without documentation
- Without report, delete all clues

Hiding during the process?

Ethical hacking concepts

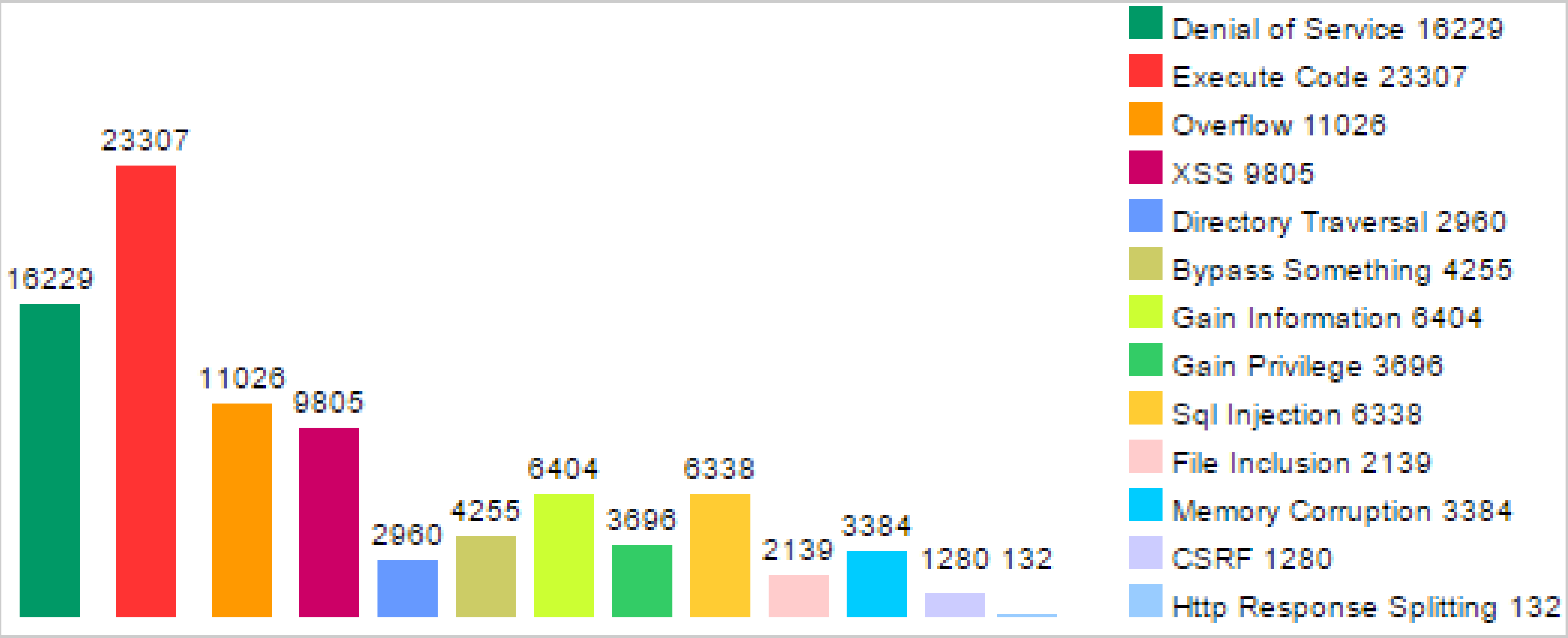
	Black box concept	Grey box concept	White box concept
Internal penetration test	X	X	X
External penetration test	X	X	X
Web hacking	X	X	X
Wireless hacking	X	X	X
Social engineering		X	

Ethical hacking steps

- General information gathering
- Technical Information gathering
- Looking for available hosts
- Looking for available services
- Manual testing
- Automatic testing
- Exploitation
- Covering tracks

Vulnerability types

Vulnerabilities By Type



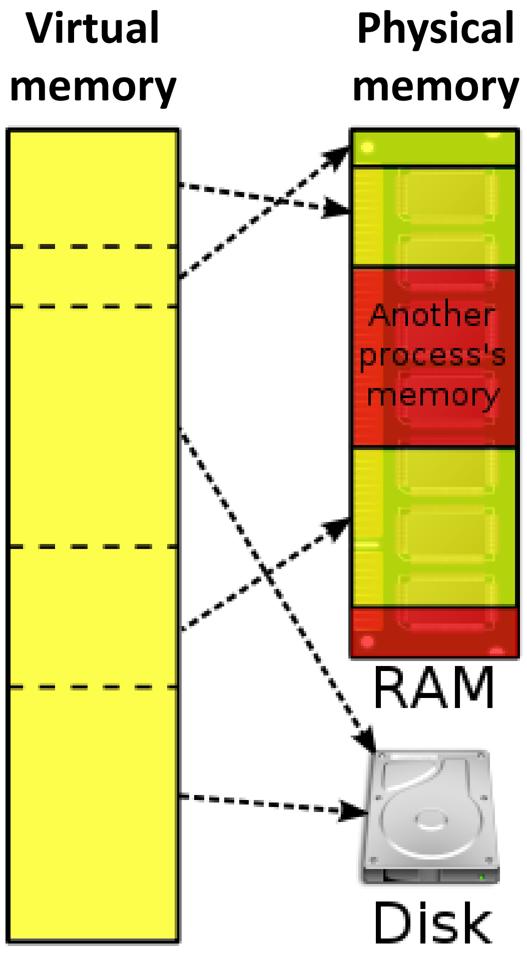
CEH topics

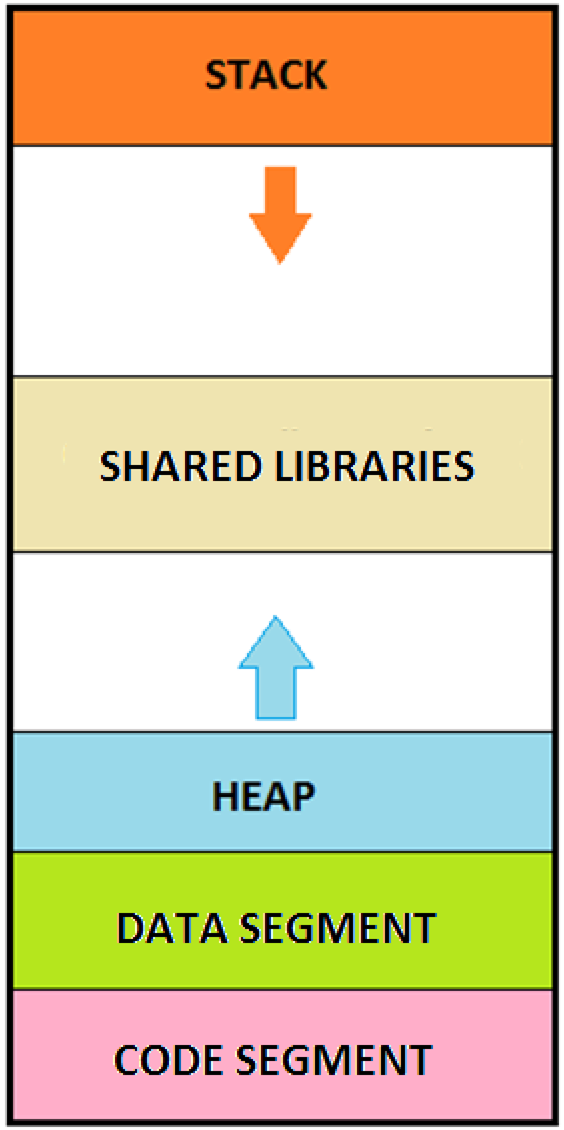
- Introduction to Ethical Hacking
- Footprinting and Reconnaissance
- Scanning Networks
- Enumeration
- System Hacking
- Malware Threats
- Sniffing
- Social Engineering
- Denial of Service
- Session Hijacking
- Hacking Webservers
- Hacking Web Applications
- SQL Injection
- Hacking Wireless Networks
- Hacking Mobile Platforms
- Evading IDS, Firewalls, and Honeypots
- Cloud Computing
- Cryptography

Ethical hacking course at UiA

http://ethical_hacking.project.uia.no

Virtual address space





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

Dynamically linked shared libraries (libc)

Dynamic variables

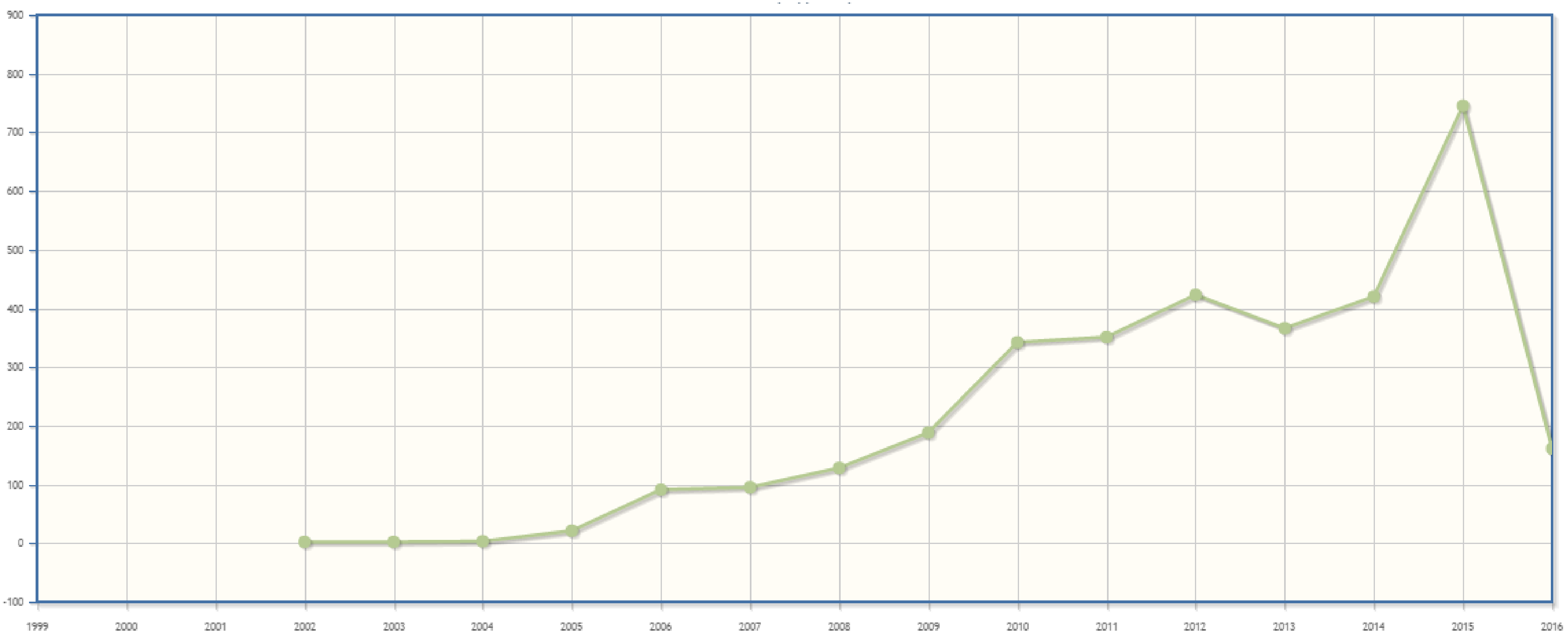
Global variables

Compiled code

Main causes and exploitation methods

- Lack of input validation within methods (strcpy, gets, etc): stack based overflow (placing harmful code to the stack, ROP, JOP)
- Dynamic memory allocation problems (use after free, double free vulnerabilities) heap overflow (function pointer overwrite + heap spray)
- Exception handling errors (SEH overwrite)
- Others

Memory corruption vulnerabilities since 2002



2002 2005 2008 2011

What's the problem with this?
(stack overflow)

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

What's the problem with this? (format string)

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


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

What's the problem with this?
(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’;
```

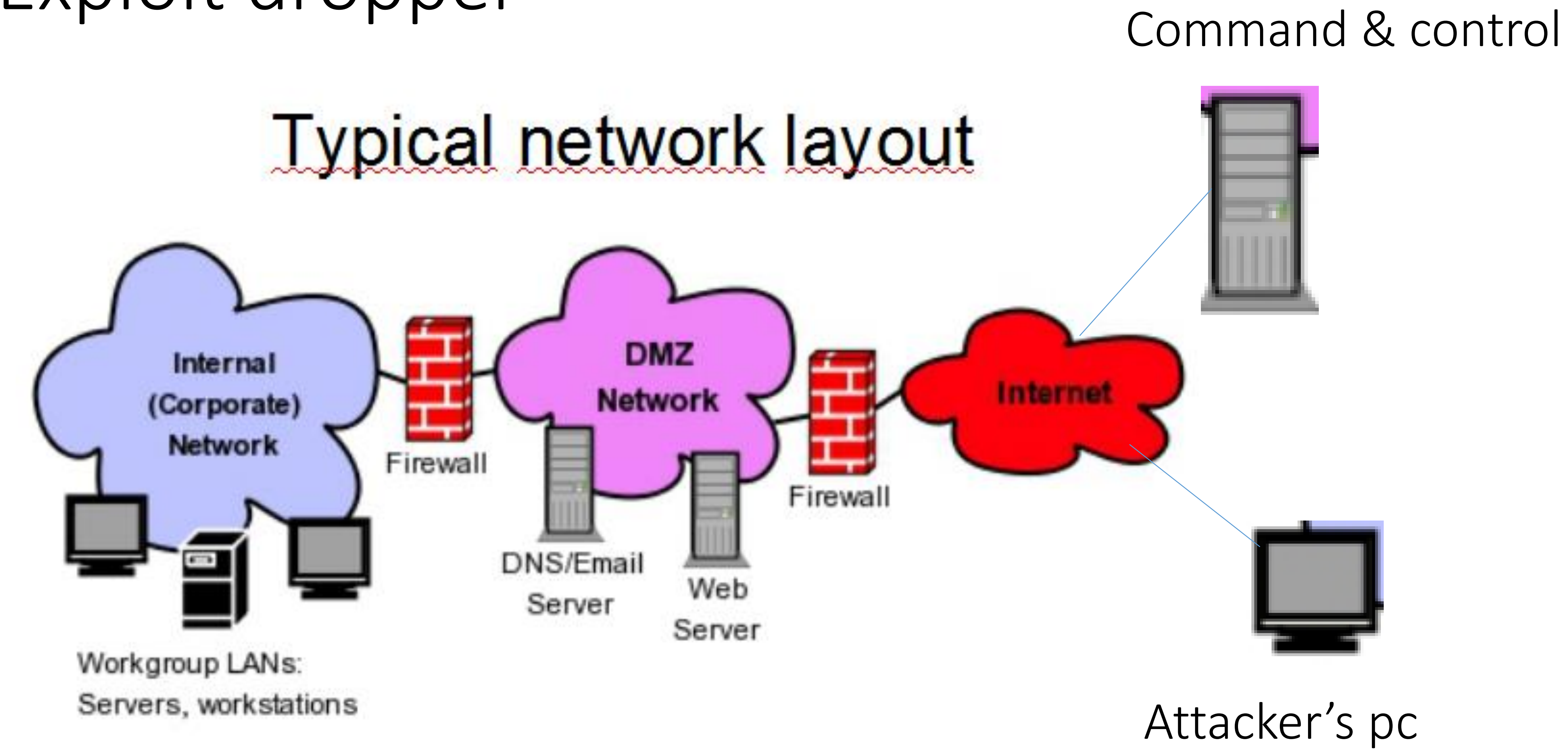
What's the problem with this?
(use after free)

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

What's the problem with this? (double free)

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

Exploit dropper



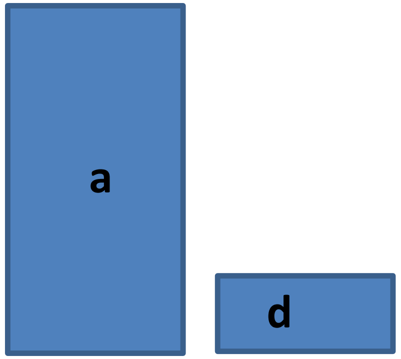
Command & control

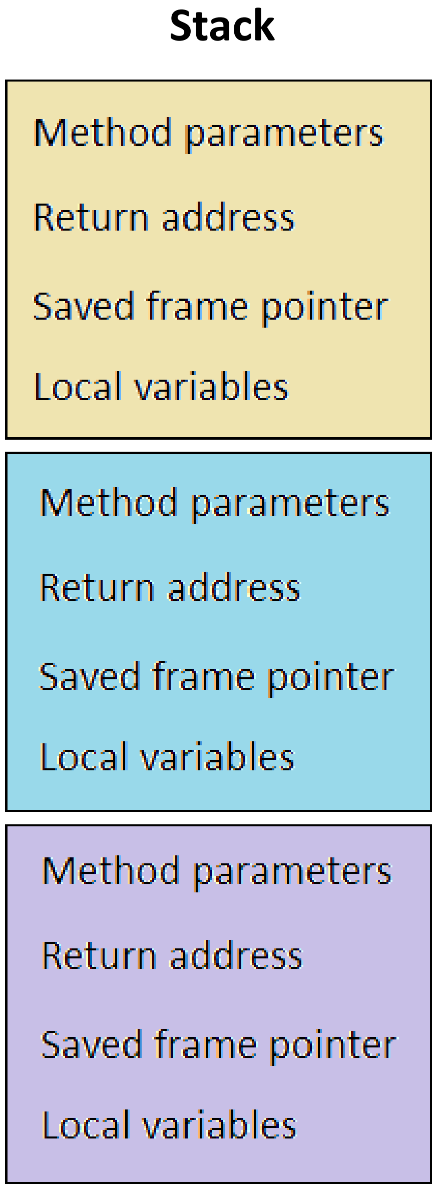
Attacker’s pc

Classic example of buffer overflow

```
…
Method1(a)
{
d : fixed size array
copy a to d
}

Method2()
{
Method1(a);
}
…
```




Stack

Method parameters

Return address

Saved frame pointer

Local variables

Method parameters

Return address

Saved frame pointer

Local variables

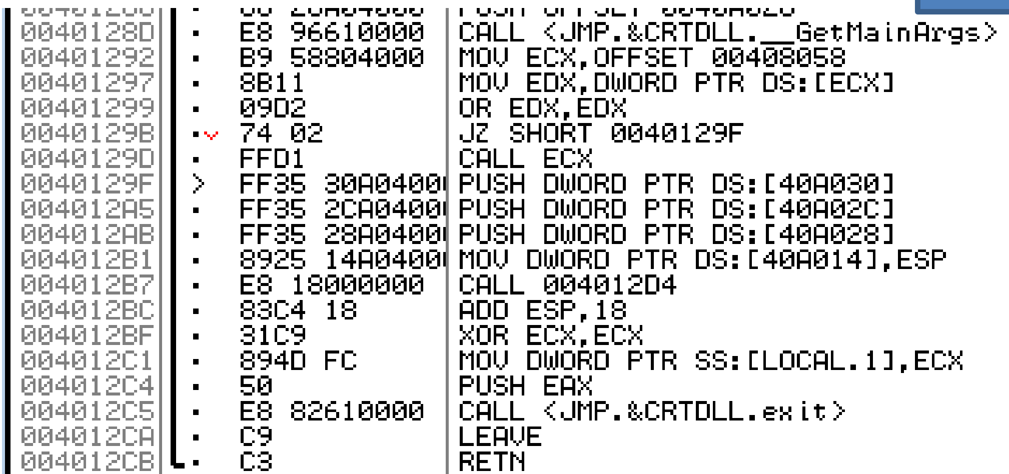
Method parameters

Return address

Saved frame pointer

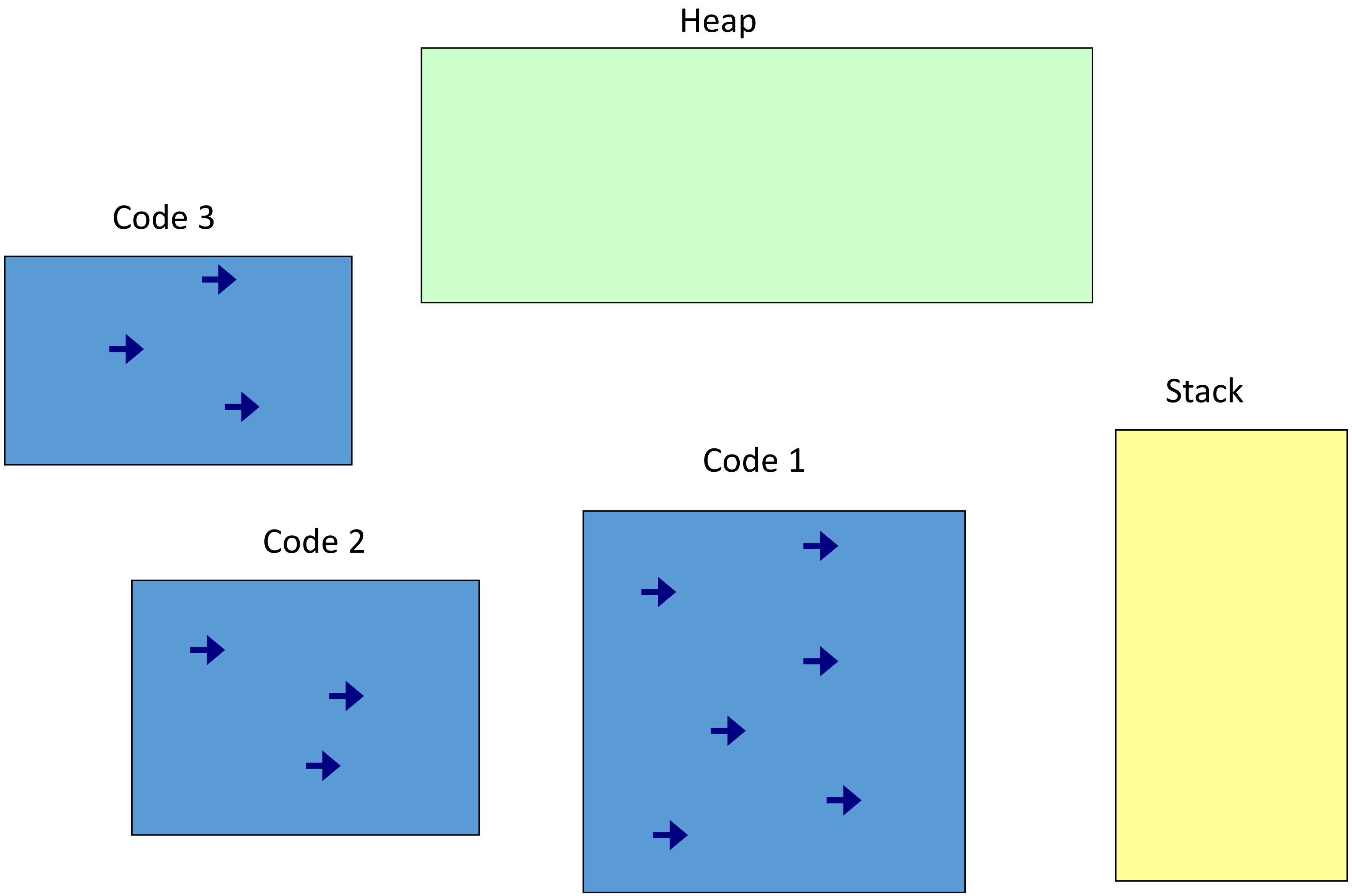
Local variables

Code segment

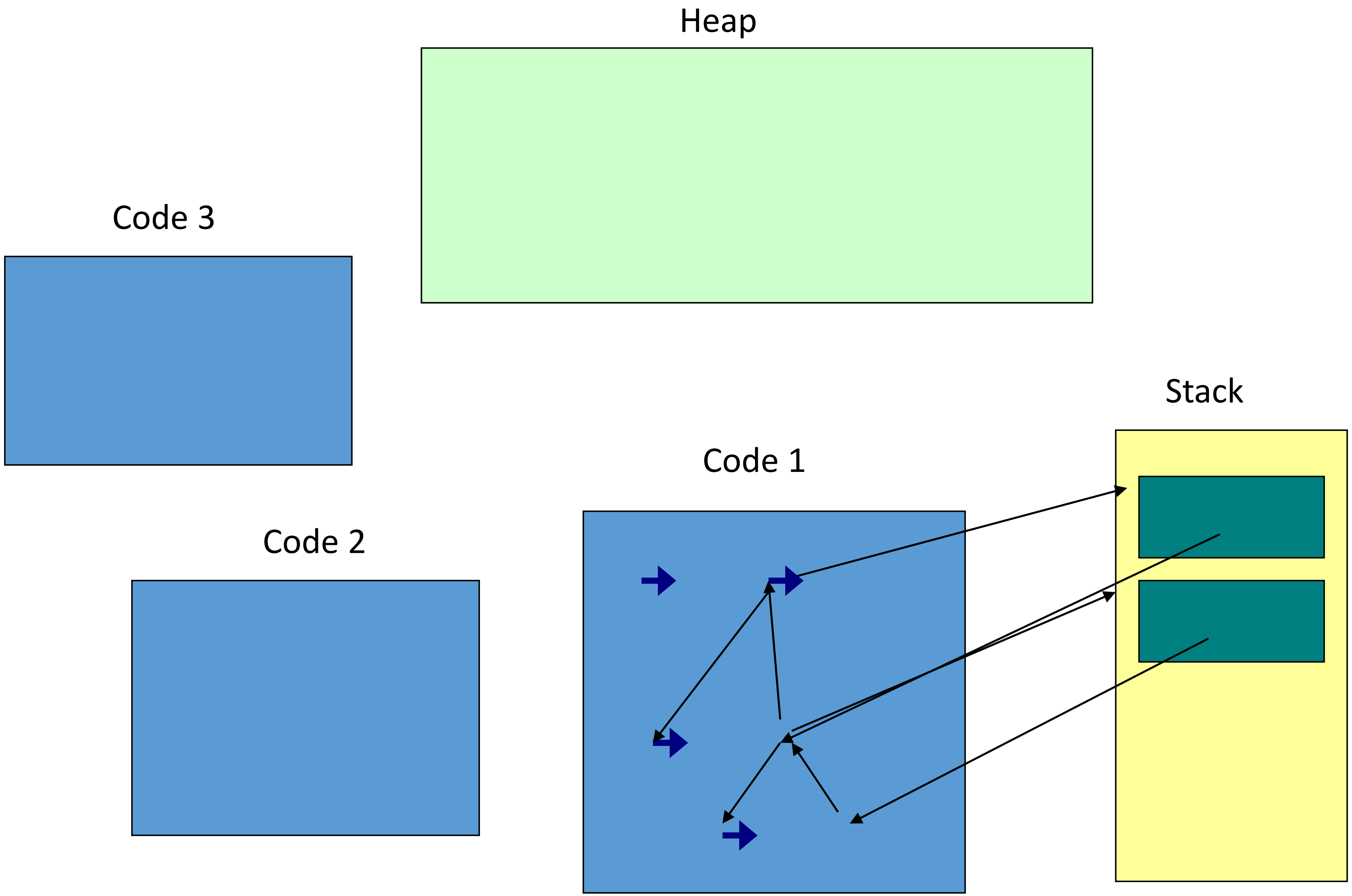


```
00401288  .  68 20A04000   PUSH OFFSET 0040A020
0040128D  .  E8 96610000   CALL <JMP.&CRTDLL.__GetMainArgs>
00401292  .  B9 58804000   MOV ECX,OFFSET 00408058
00401297  .  8B11          MOV EDX,DWORD PTR DS:[ECX]
00401299  .  09D2          OR EDX,EDX
0040129B  .v 74 02         JZ SHORT 0040129F
0040129D  .  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
```

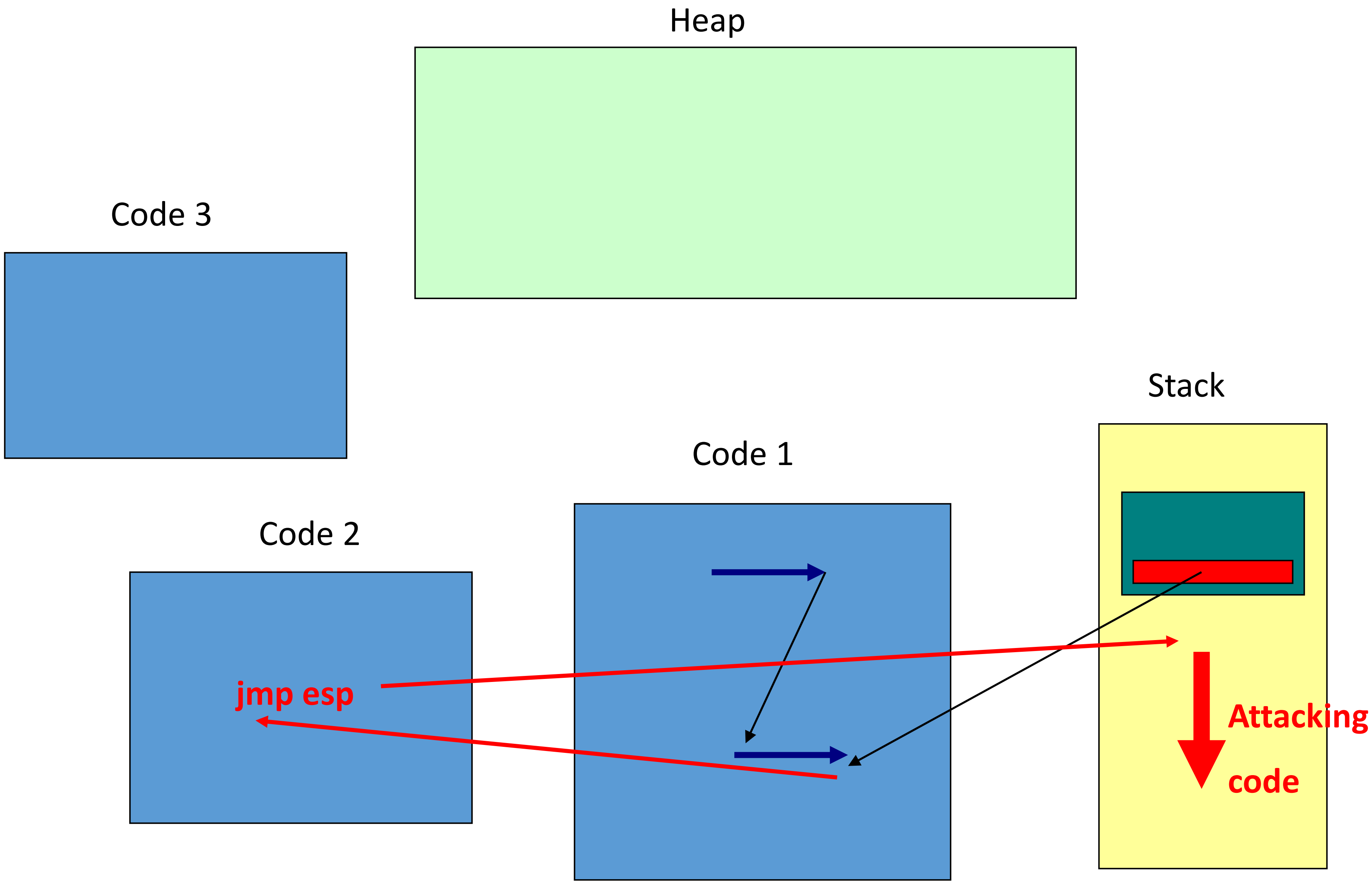
Normal operation



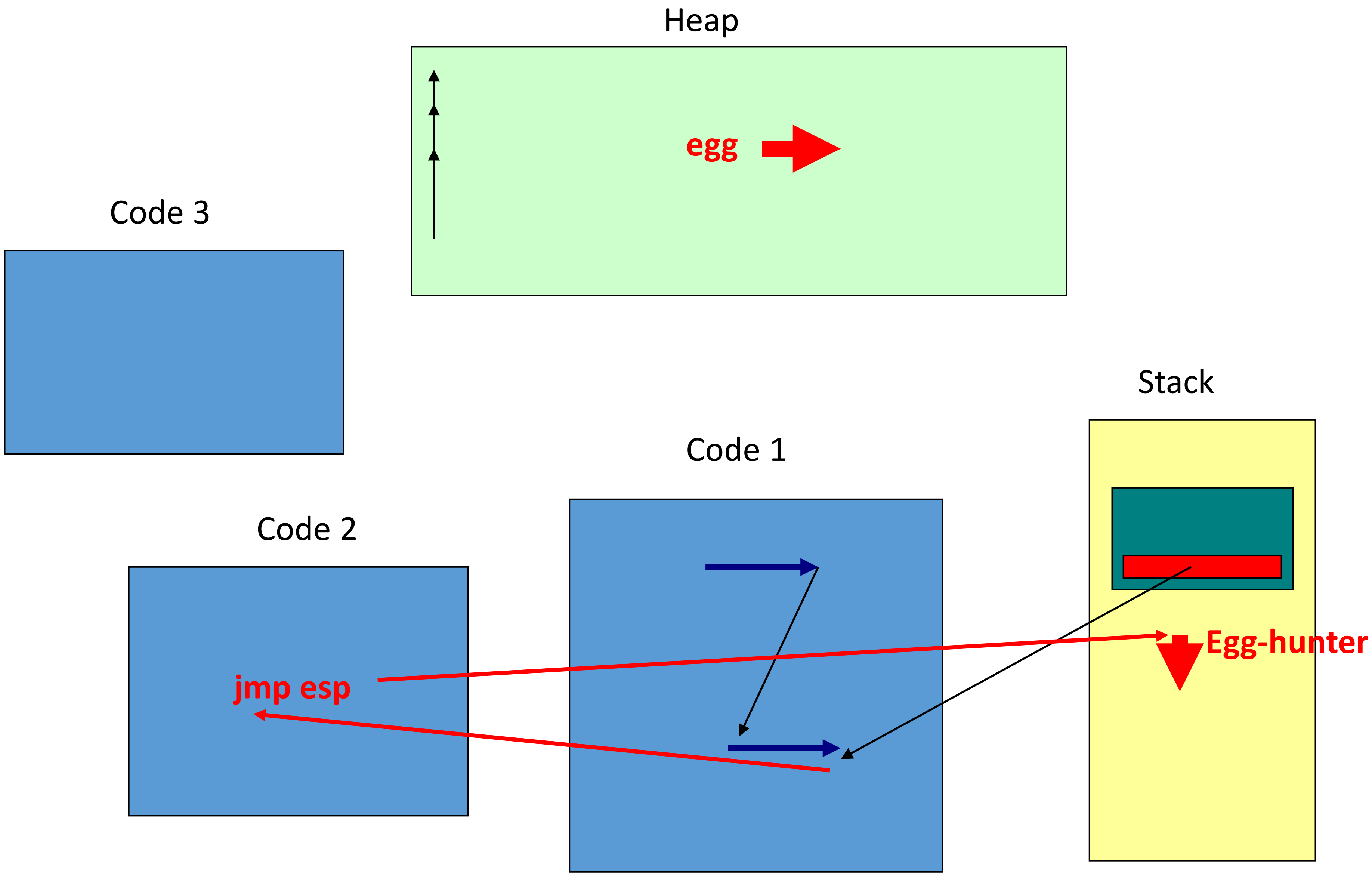
Normal operation



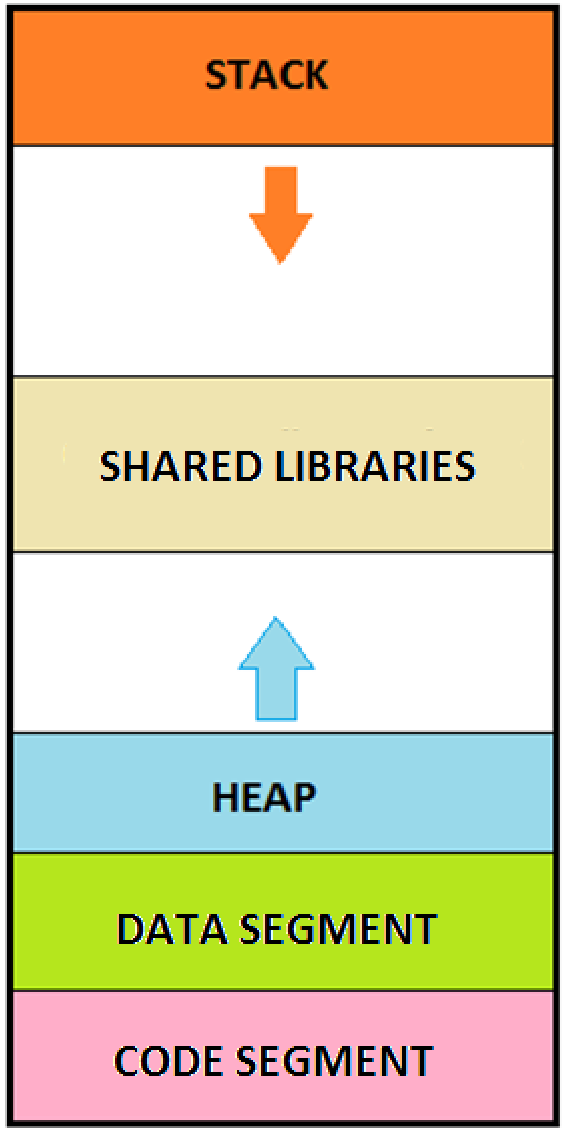
Normal operation



Egg-hunter



Data Execution Prevention



Data: read/write

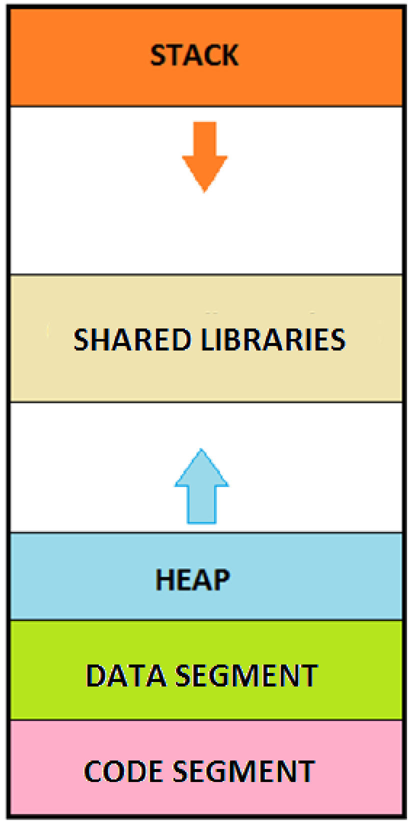
Code: read/execute

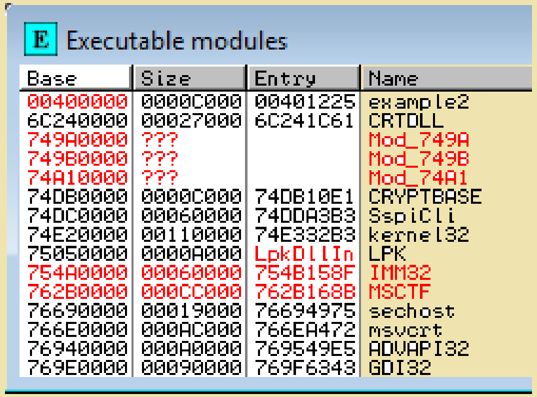
Data: read/write

Data: read/write

Code: read/execute

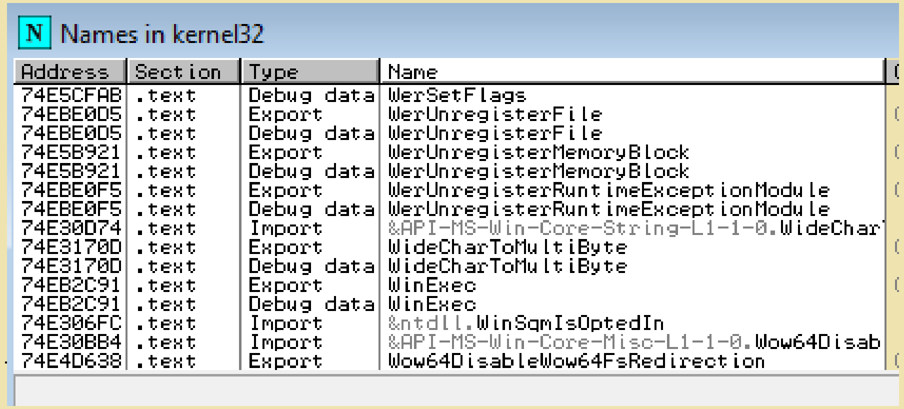
Avoiding memory execution protection (return to libc)





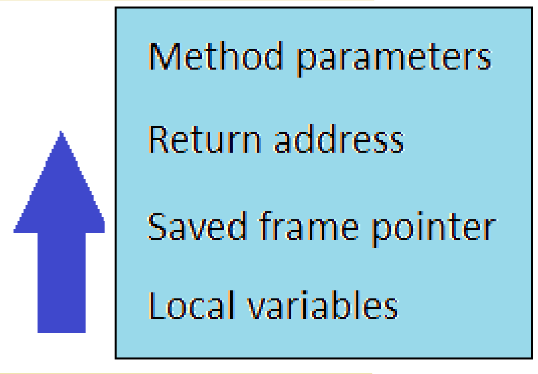
E Executable modules

Base	Size	Entry	Name
00400000	0000C000	00401225	example2
6C240000	00027000	6C241C61	CRTDLL
749A0000	???		Mod_749A
749B0000	???		Mod_749B
74A10000	???		Mod_74A1
74DB0000	0000C000	74DB10E1	CRYPTBASE
74DC0000	00060000	74DDA3B3	SspiCli
74E20000	00110000	74E332B3	kernel32
75050000	0000A000	LpkDllIn	LPK
754A0000	00060000	754B158F	IMM32
762B0000	000CC000	762B168B	MSCTF
76690000	00019000	76694975	sechost
766E0000	000AC000	766EA472	msvcrt
76940000	000A0000	769549E5	ADVAPI32
769E0000	00090000	769F6343	GDI32



N Names in kernel32

Address	Section	Type	Name
74E5CFAB	.text	Debug data	WerSetFlags
74EBE0D5	.text	Export	WerUnregisterFile
74EBE0D5	.text	Debug data	WerUnregisterFile
74E5B921	.text	Export	WerUnregisterMemoryBlock
74E5B921	.text	Debug data	WerUnregisterMemoryBlock
74EBE0F5	.text	Export	WerUnregisterRuntimeExceptionModule
74EBE0F5	.text	Debug data	WerUnregisterRuntimeExceptionModule
74E30D74	.text	Import	&API-MS-Win-Core-String-L1-1-0.WideChar
74E3170D	.text	Export	WideCharToMultiByte
74E3170D	.text	Debug data	WideCharToMultiByte
74EB2C91	.text	Export	WinExec
74EB2C91	.text	Debug data	WinExec
74E306FC	.text	Import	&ntdll.WinSqmIsOptedIn
74E30BB4	.text	Import	&API-MS-Win-Core-Misc-L1-1-0.Wow64Disab
74E4D638	.text	Export	Wow64DisableWow64FsRedirection



Method parameters

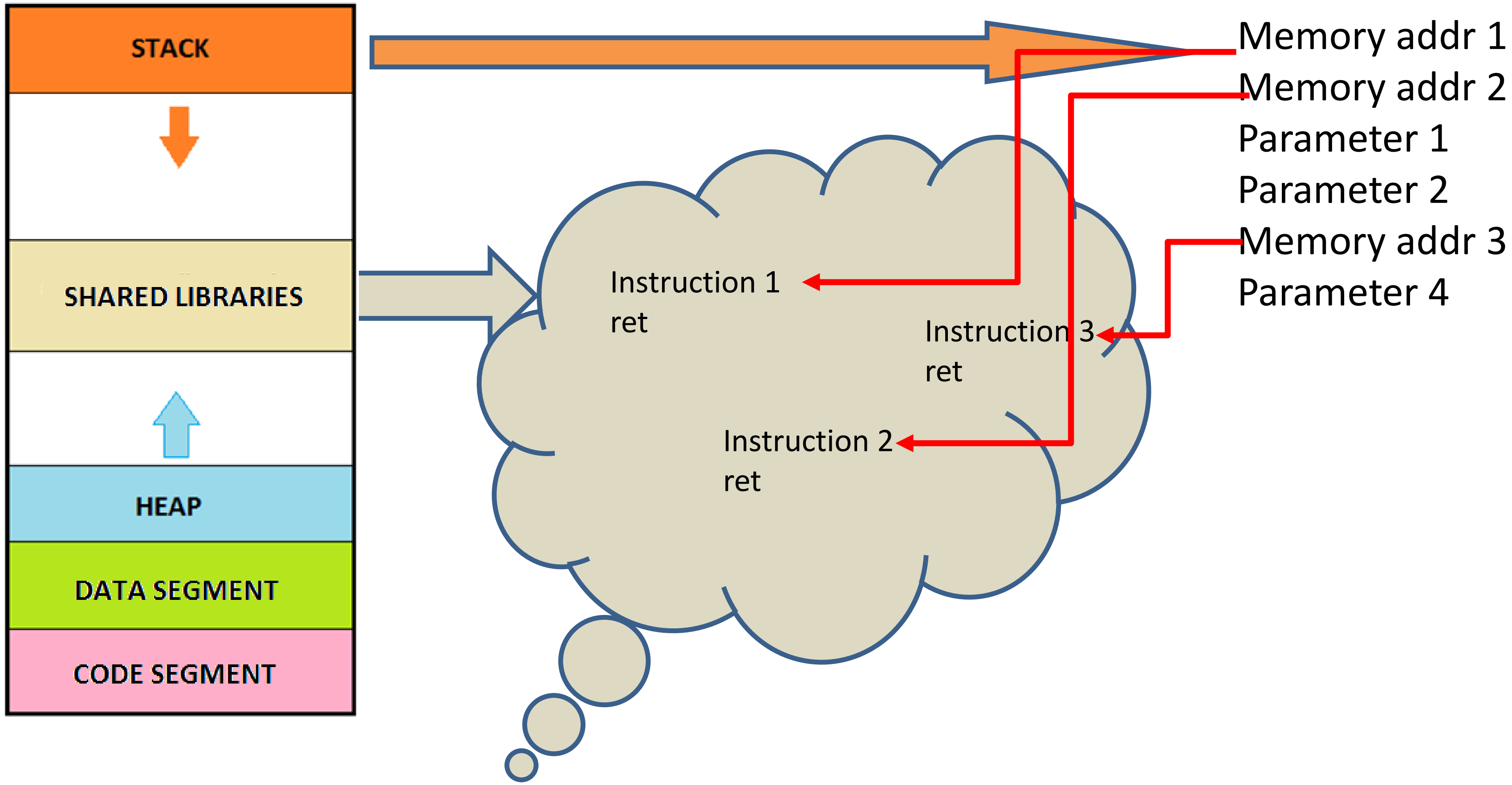
Return address

Saved frame pointer

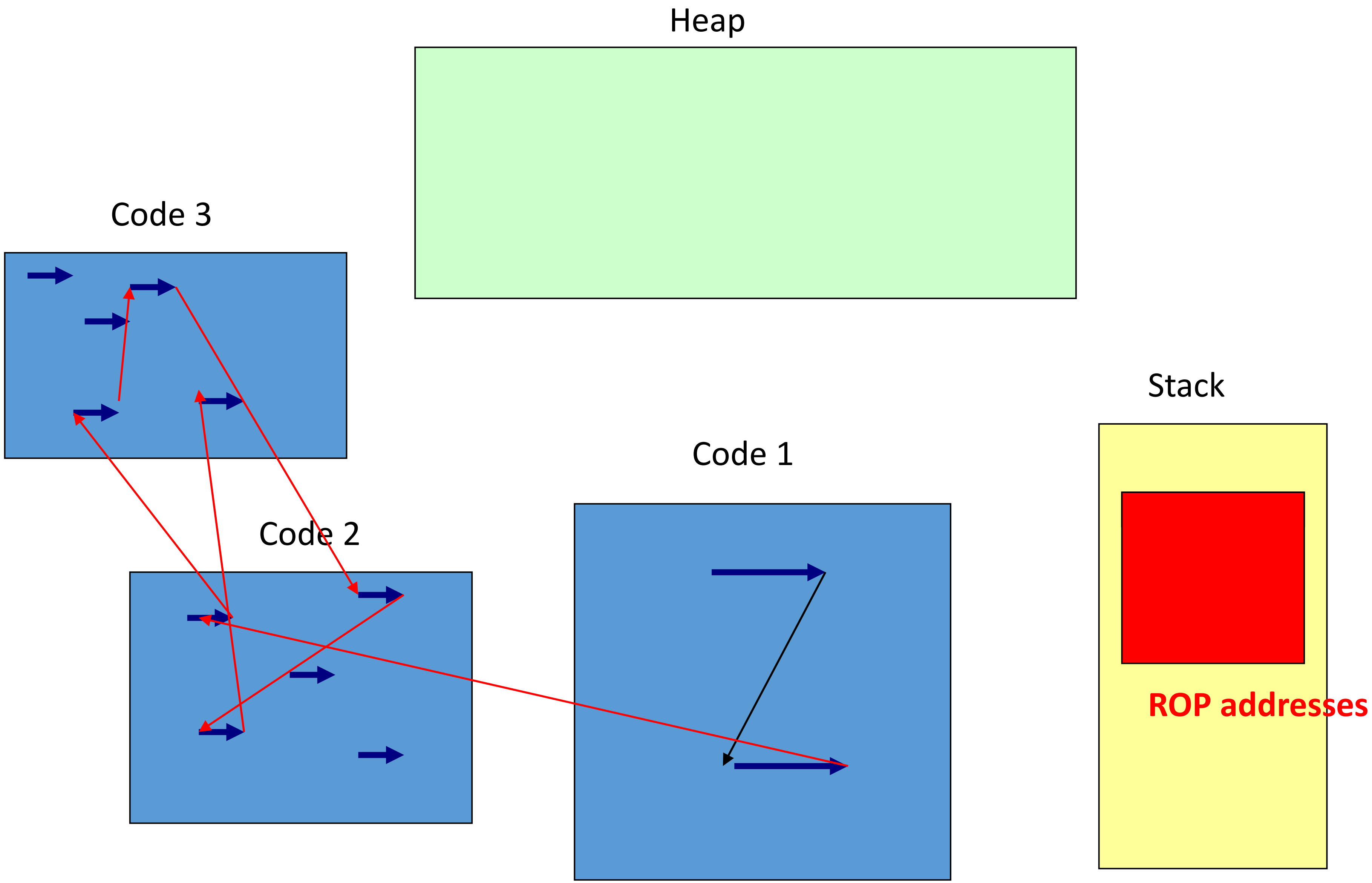
Local variables

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



Heap

Code 3

Stack

Code 1

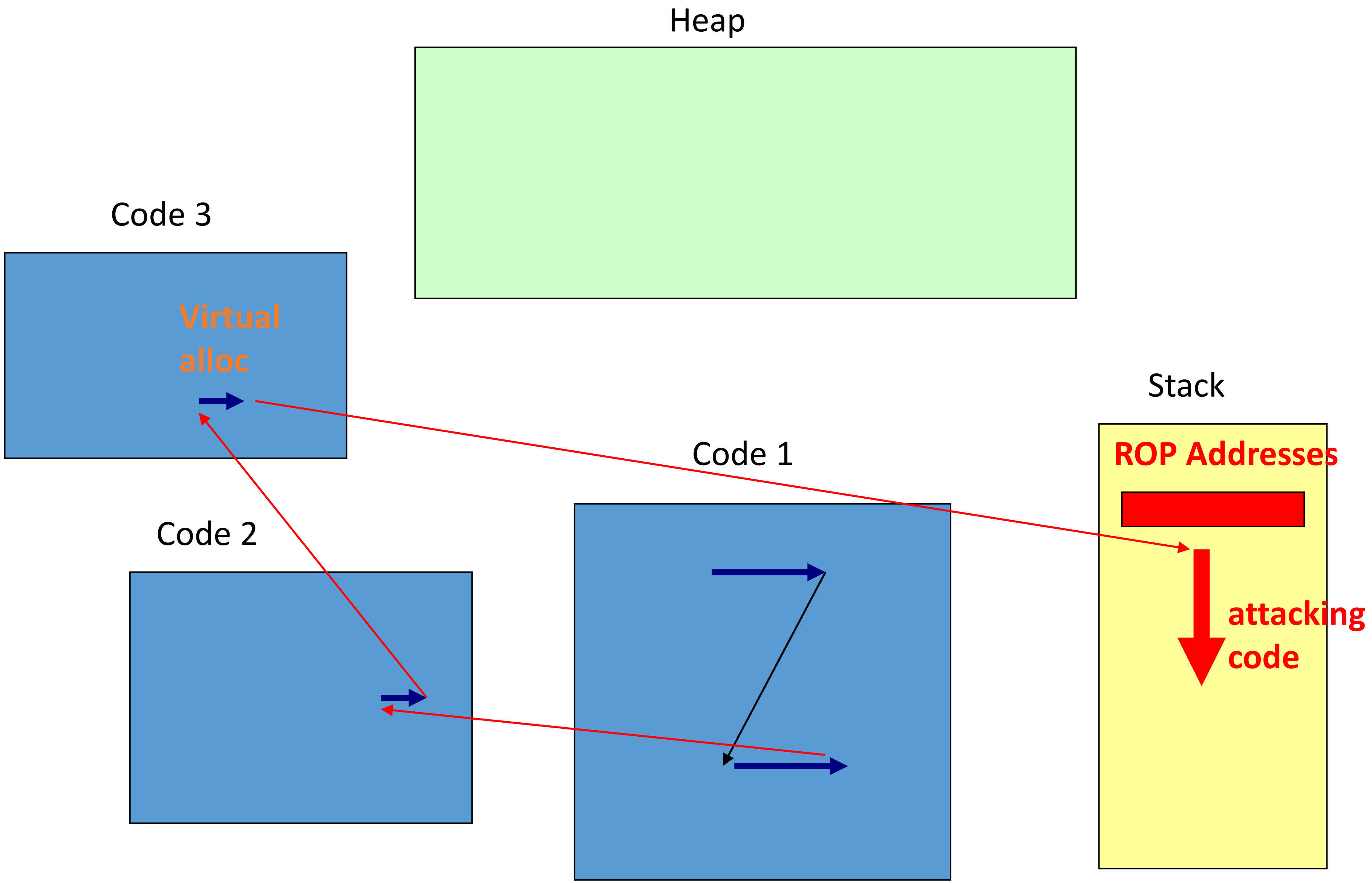
Code 2

ROP addresses

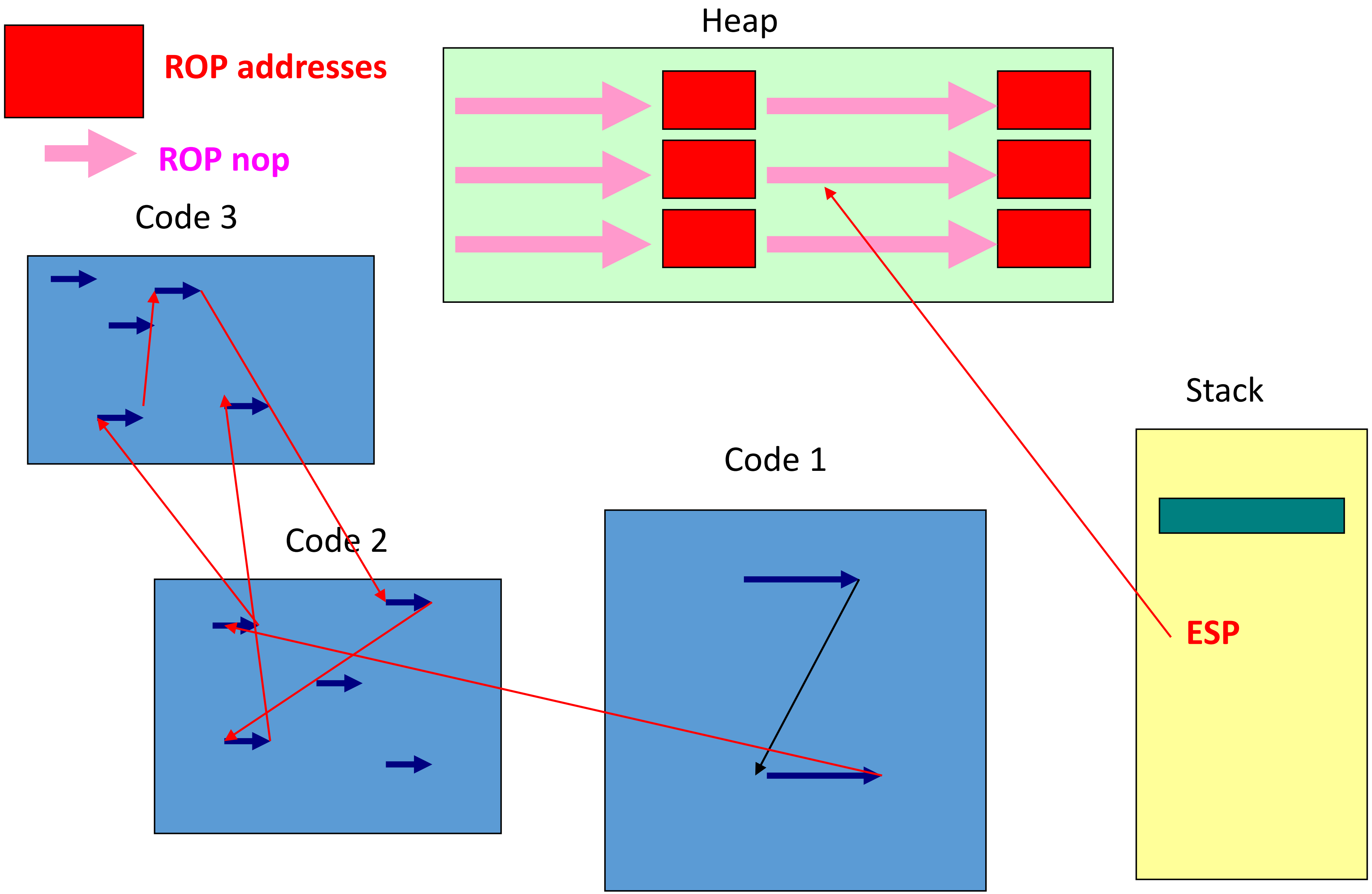
ROP – Turing completeness

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

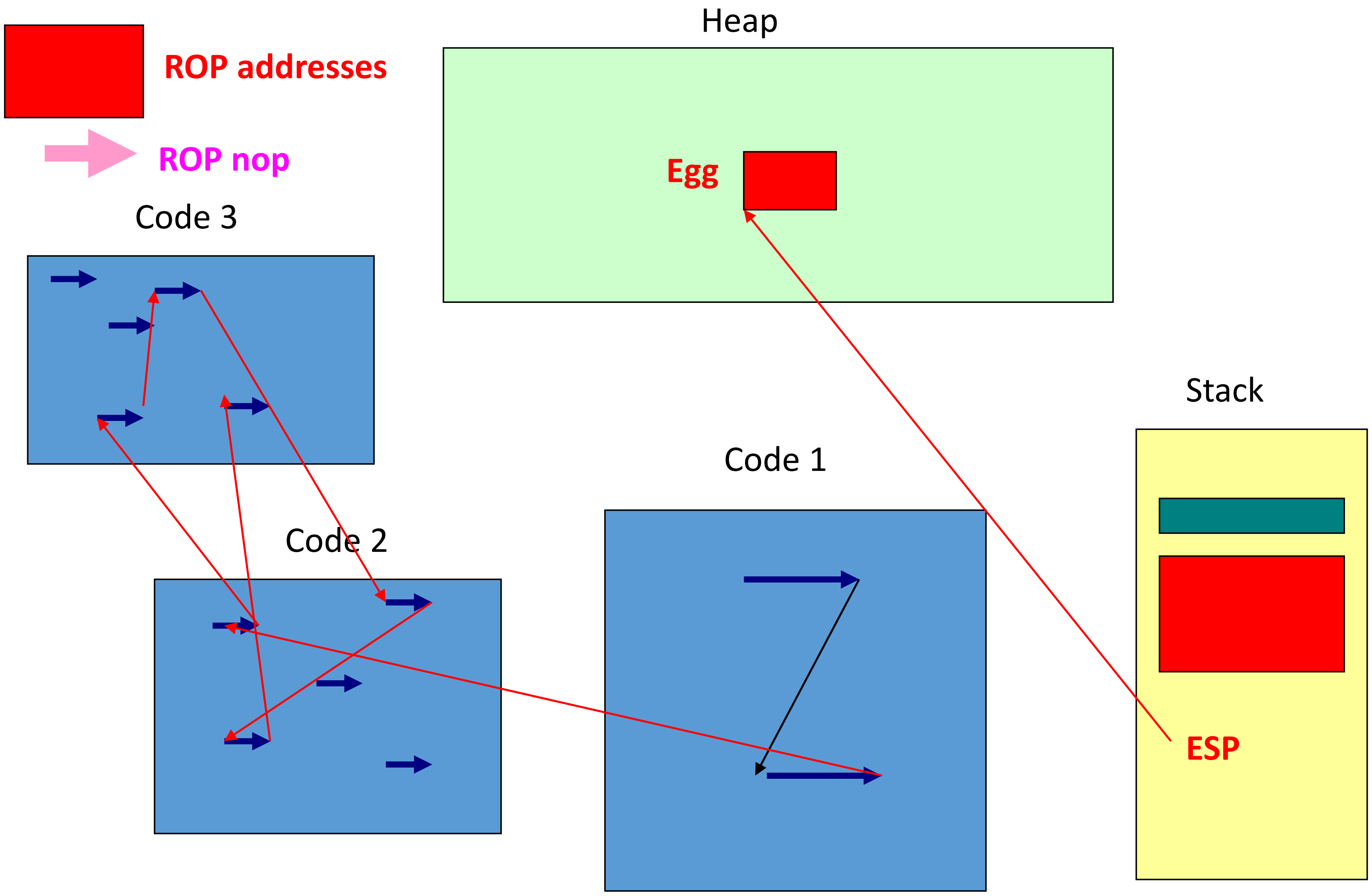
ROP + turn off DEP



ROP + Heap spray

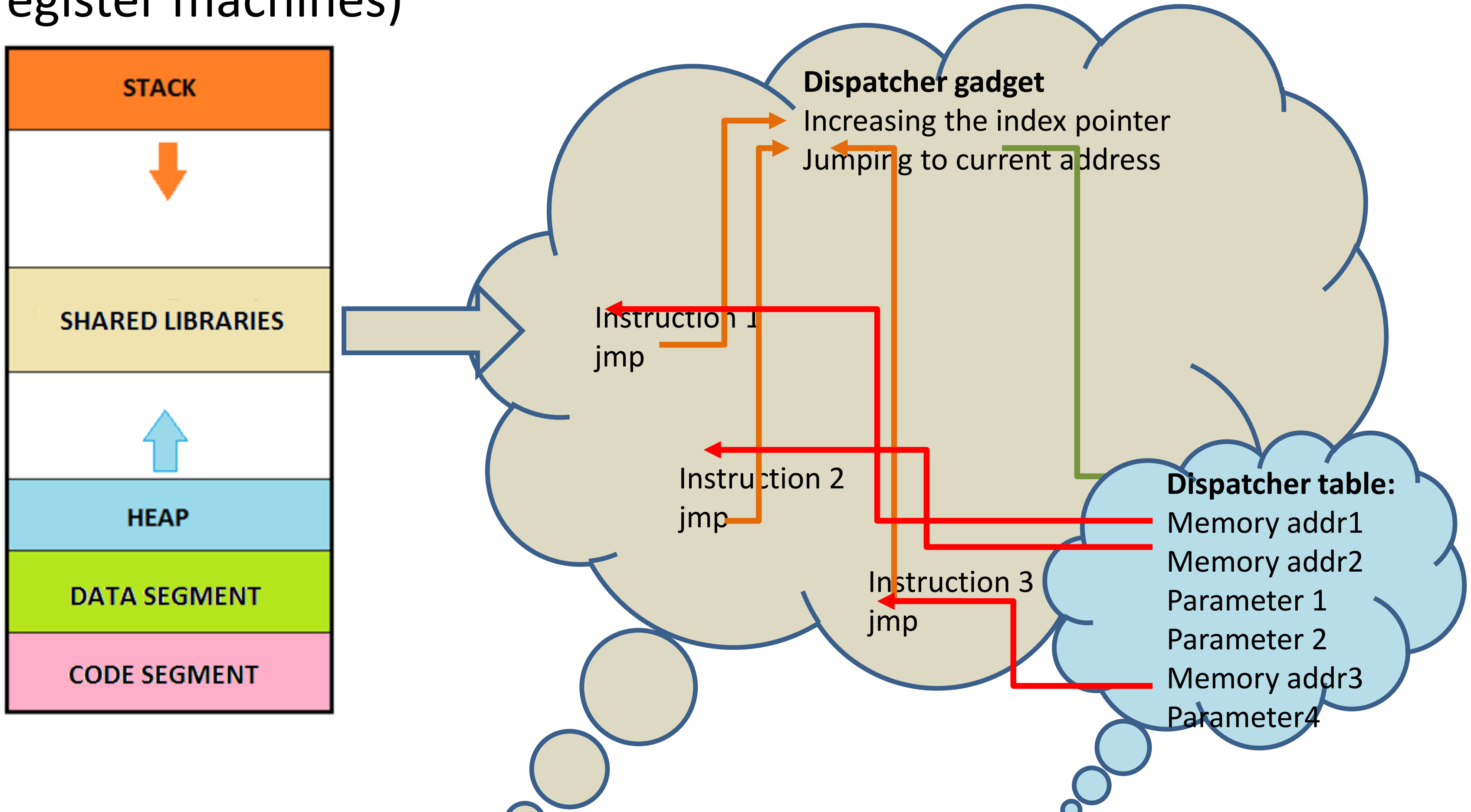


ROP + Egg-hunter

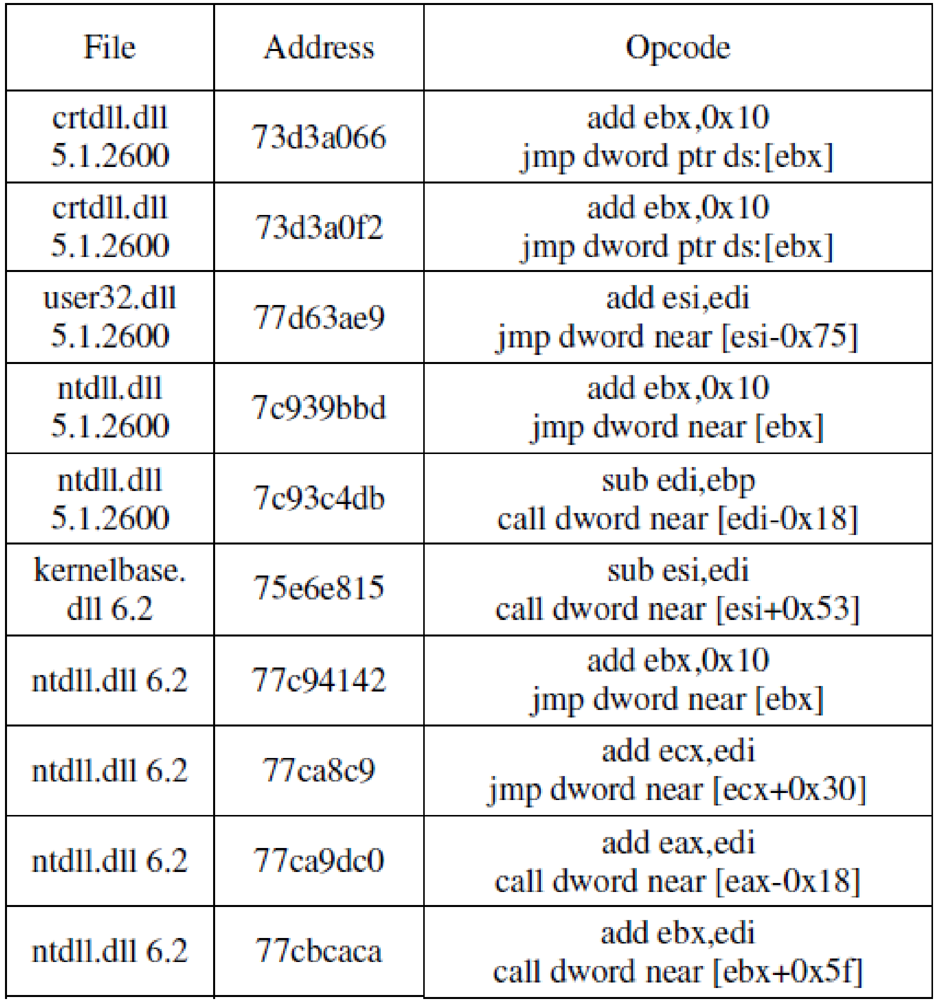


Jump oriented programming (JOP)
Bletsch, Jiang, Freeh 2011

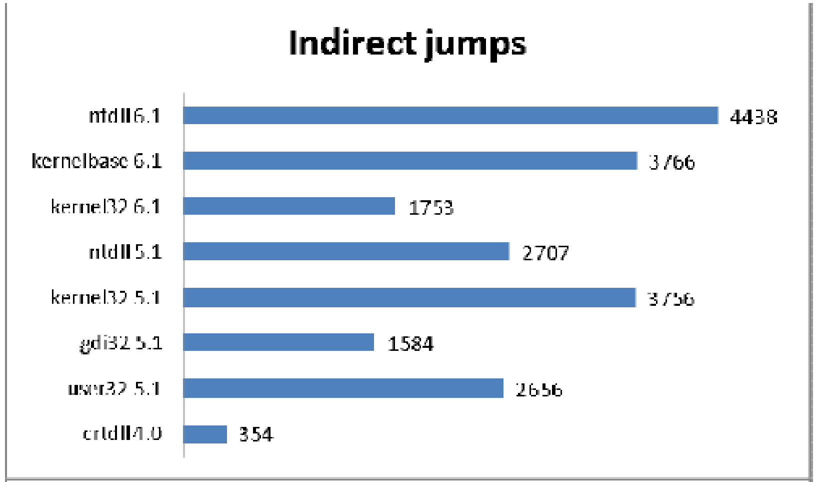
- Attack execution without using stack (not sensible for stack cookie and returnless kernel, it can be used in the case of register machines)



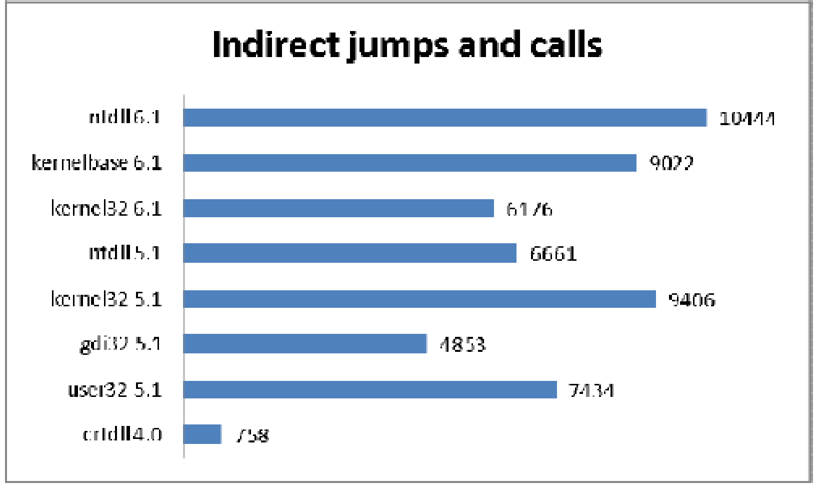
Jump Oriented Programming – dispatcher gadgets in shared libraries (**Erdődi**, 2013)



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]

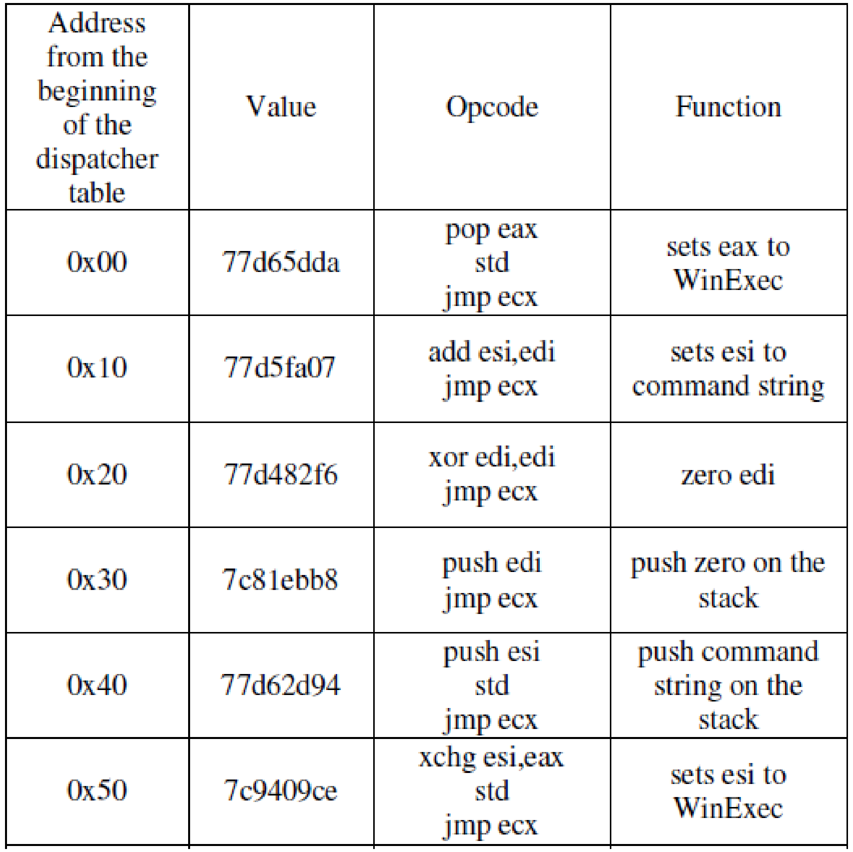


Indirect jumps

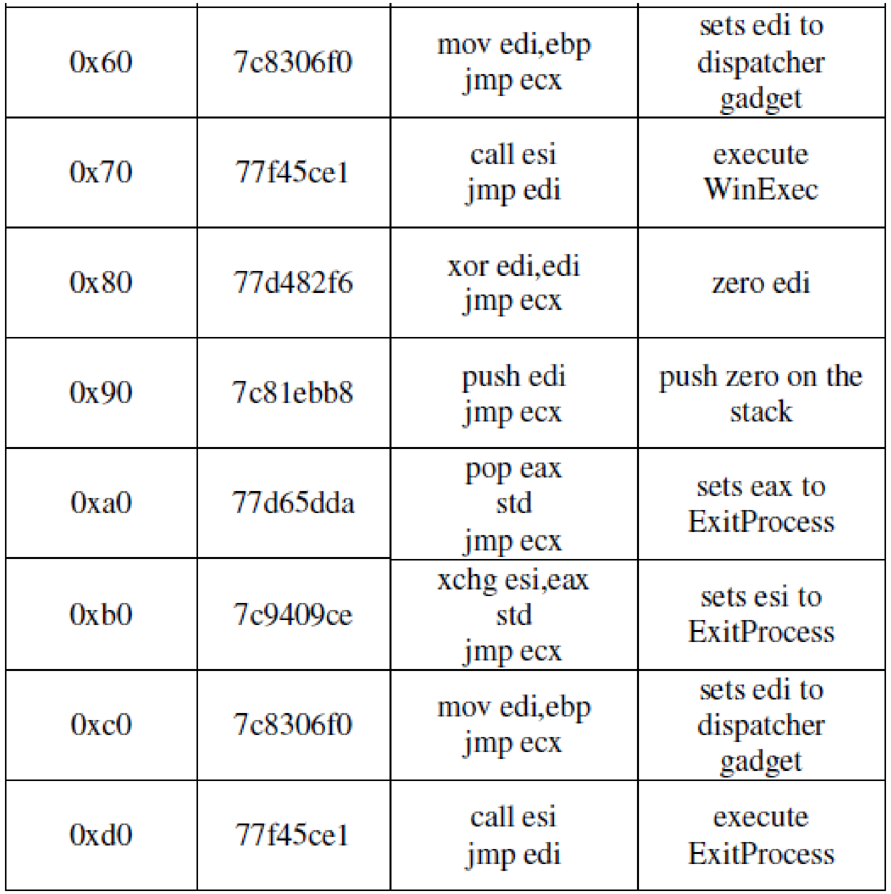


Indirect jumps and calls

Jump Oriented Programming – WinExec example for Win32 X86

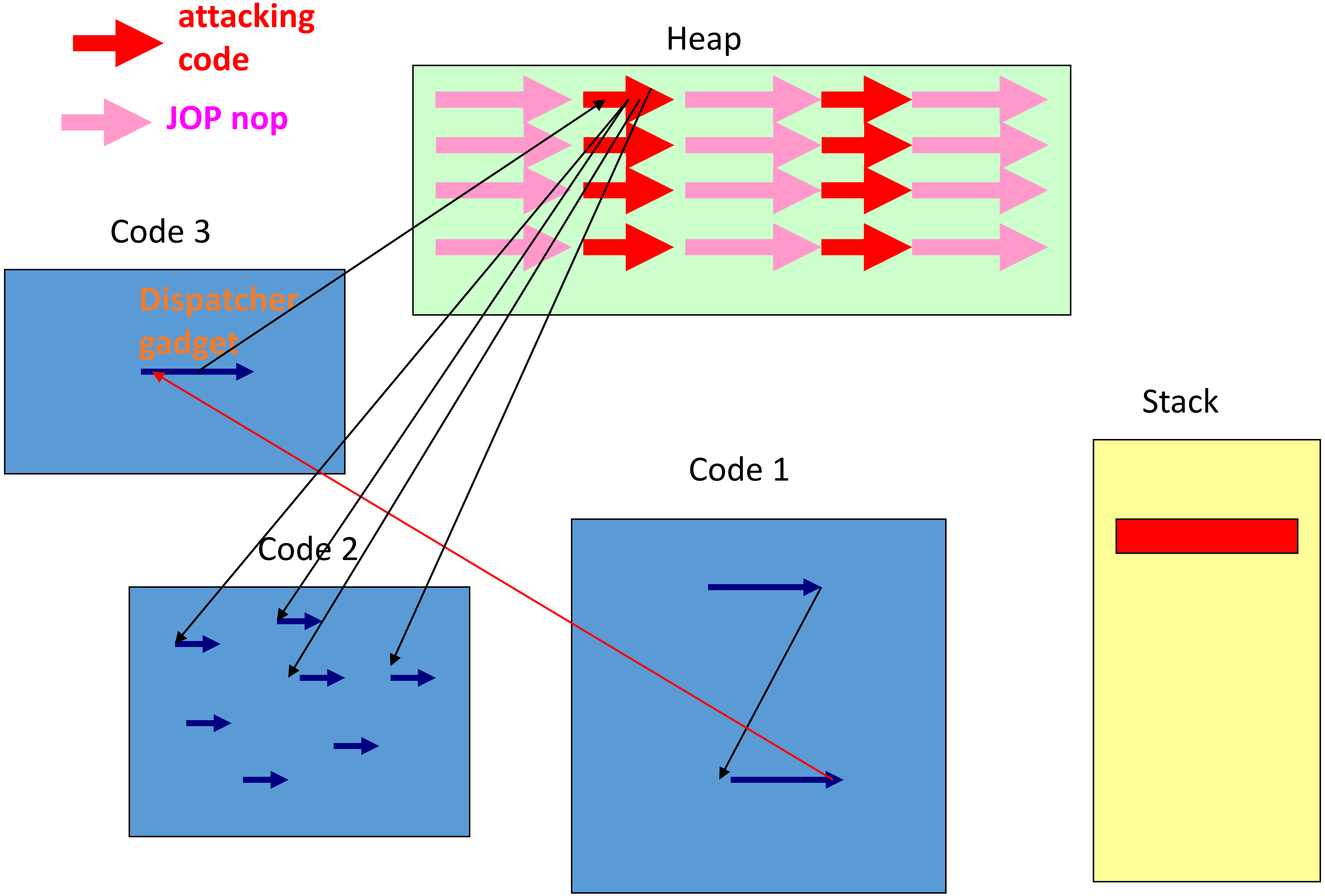


Address from the beginning of the dispatcher table	Value	Opcode	Function
0x00	77d65dda	pop eax std jmp ecx	sets eax to WinExec
0x10	77d5fa07	add esi,edi jmp ecx	sets esi to command string
0x20	77d482f6	xor edi,edi jmp ecx	zero edi
0x30	7c81ebb8	push edi jmp ecx	push zero on the stack
0x40	77d62d94	push esi std jmp ecx	push command string on the stack
0x50	7c9409ce	xchg esi,eax std jmp ecx	sets esi to WinExec

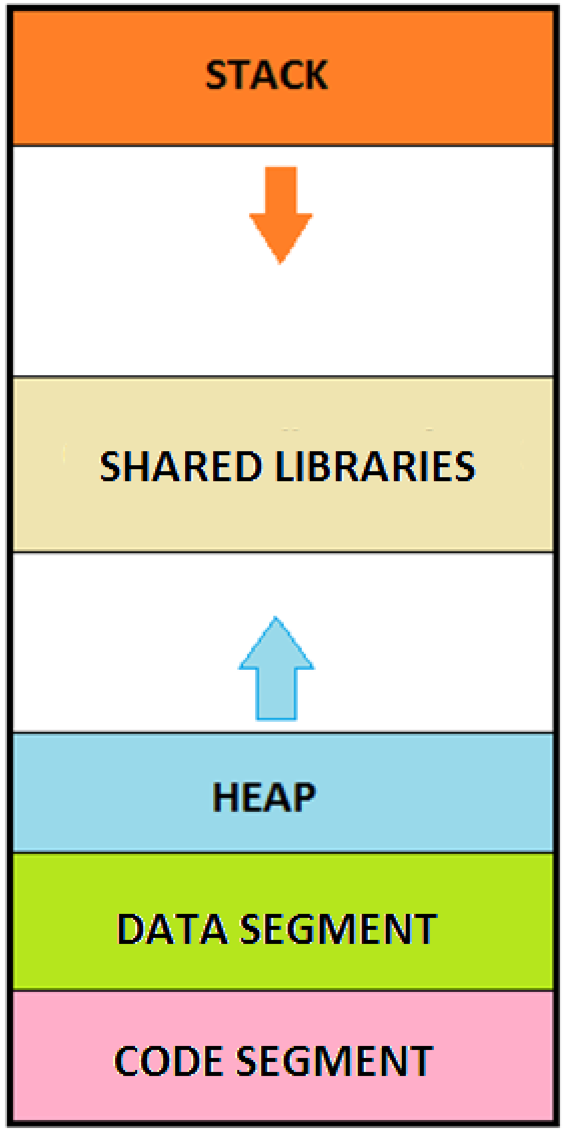


0x60	7c8306f0	mov edi,ebp jmp ecx	sets edi to dispatcher gadget
0x70	77f45ce1	call esi jmp edi	execute WinExec
0x80	77d482f6	xor edi,edi jmp ecx	zero edi
0x90	7c81ebb8	push edi jmp ecx	push zero on the stack
0xa0	77d65dda	pop eax std jmp ecx	sets eax to ExitProcess
0xb0	7c9409ce	xchg esi,eax std jmp ecx	sets esi to ExitProcess
0xc0	7c8306f0	mov edi,ebp jmp ecx	sets edi to dispatcher gadget
0xd0	77f45ce1	call esi jmp edi	execute ExitProcess

JOP + Heap spray



Address Space Layout Randomization (ASLR)



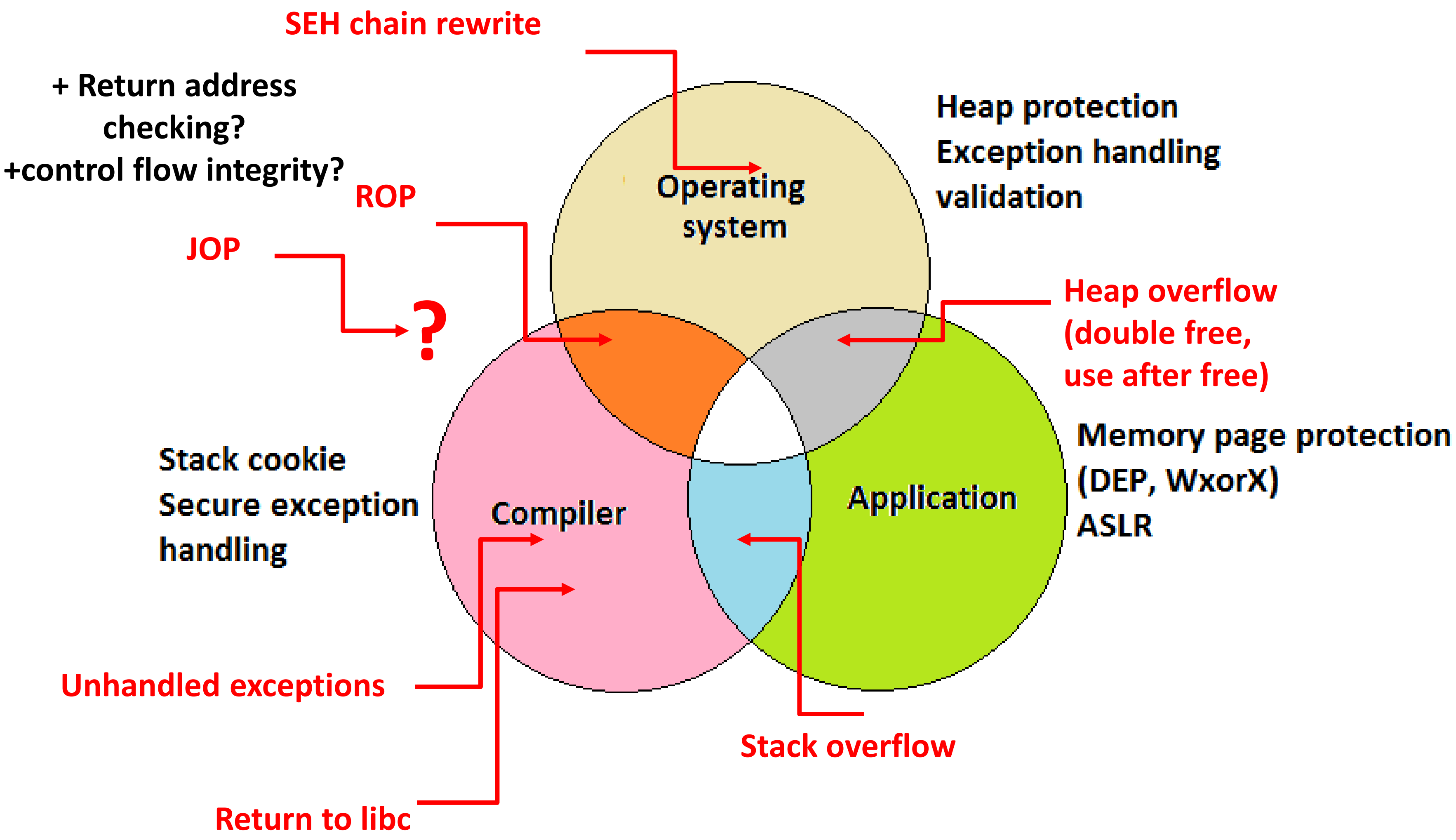
Bypassing ASLR

- Non Position Independent code segments
- Guessing the ASLR offset
- Information leakage
- JIT-ROP
- Blind ROP

Additional protections

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

Protection against memory corruption



SEH chain rewrite

+ Return address checking?
+control flow integrity?

ROP

JOP

Heap protection
Exception handling validation

Heap overflow (double free, use after free)

Memory page protection (DEP, WxorX)
ASLR

Stack cookie
Secure exception handling

Unhandled exceptions

Stack overflow

Return to libc

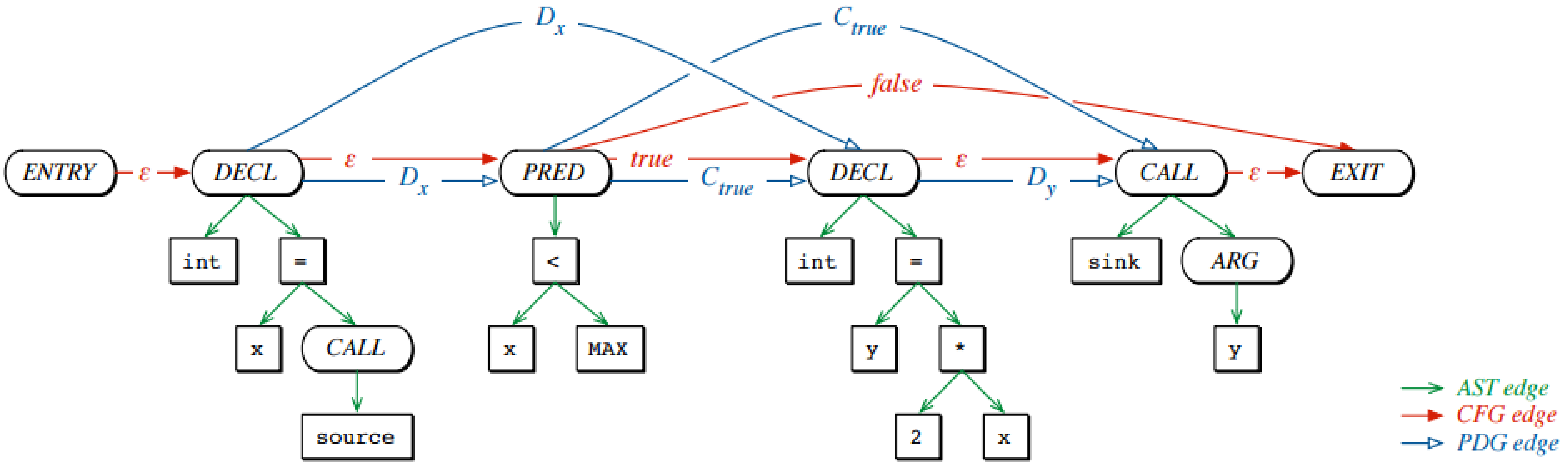
Vulnerability searching

- Static Analysis (source validators, Interactive Dissasembler (IDA))
- Dynamic Analyis (Fuzzing)
- Finding vulnerability accidently
- AV softwares by behaviour analysis (for already discovered non-public 0days)

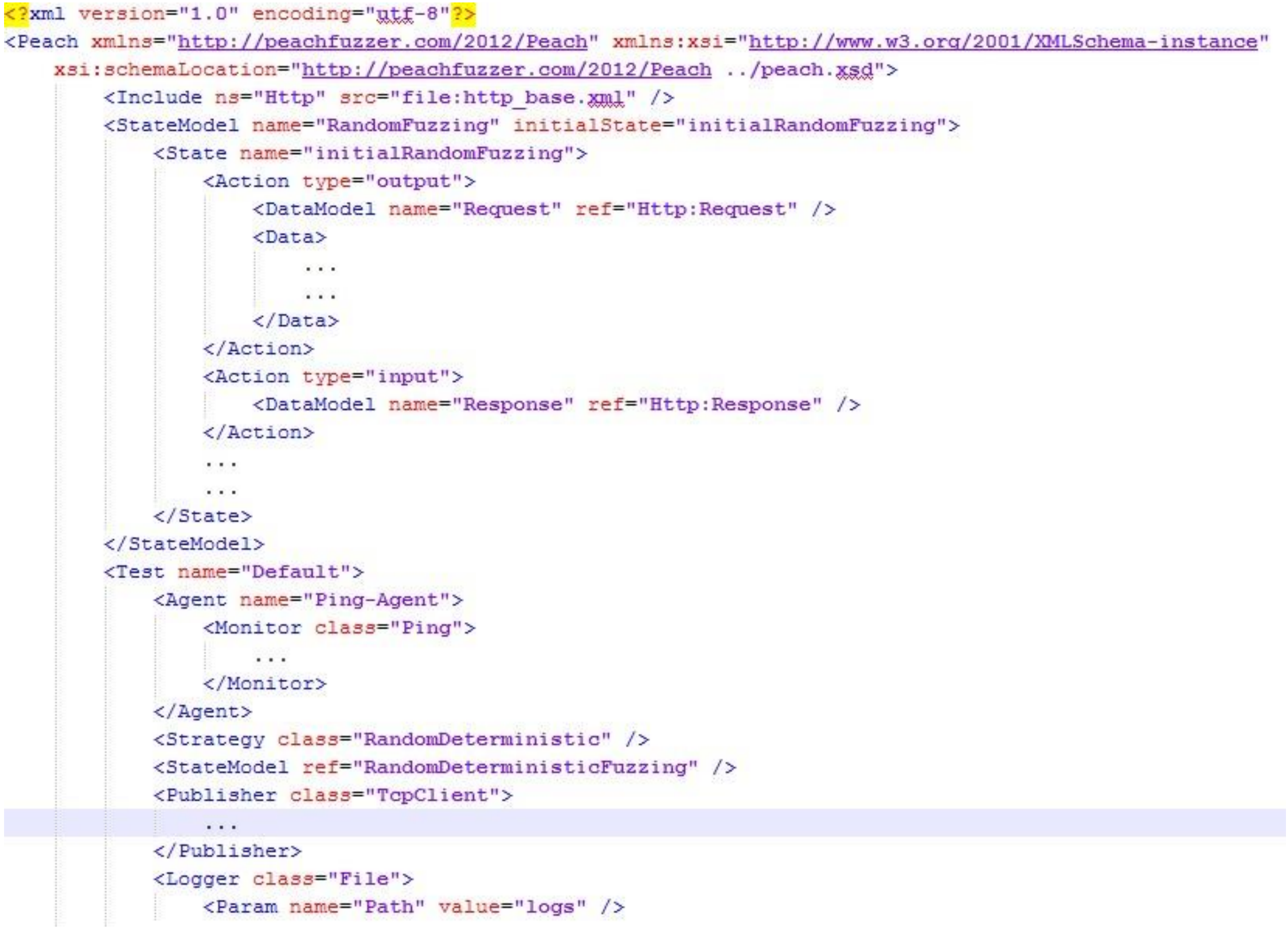
Static code analyzers

- Unreachable codes
- Code duplicates
- Inappropriate memory management
- Lack of validation
- Etc.

Code Property Graph (Yamaguchi et al, 2014)

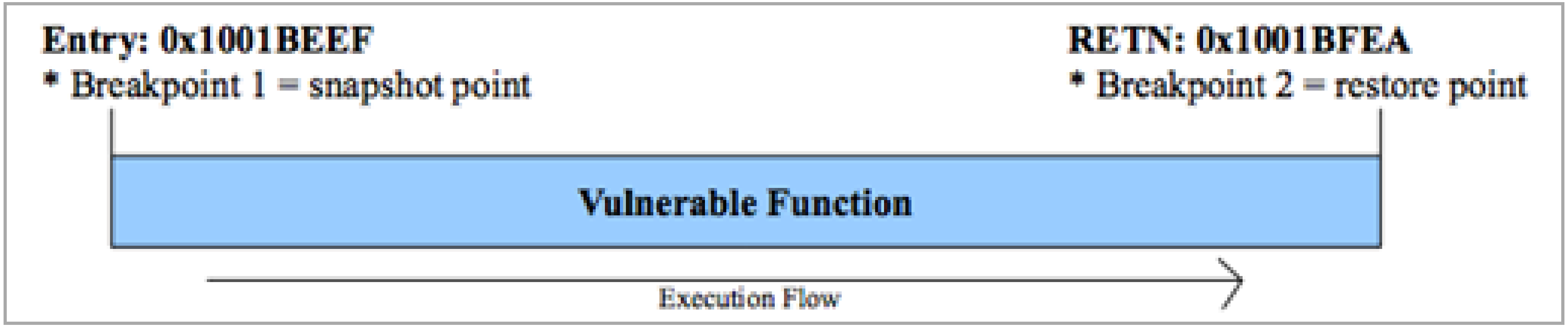


Input parameter / file format fuzzing

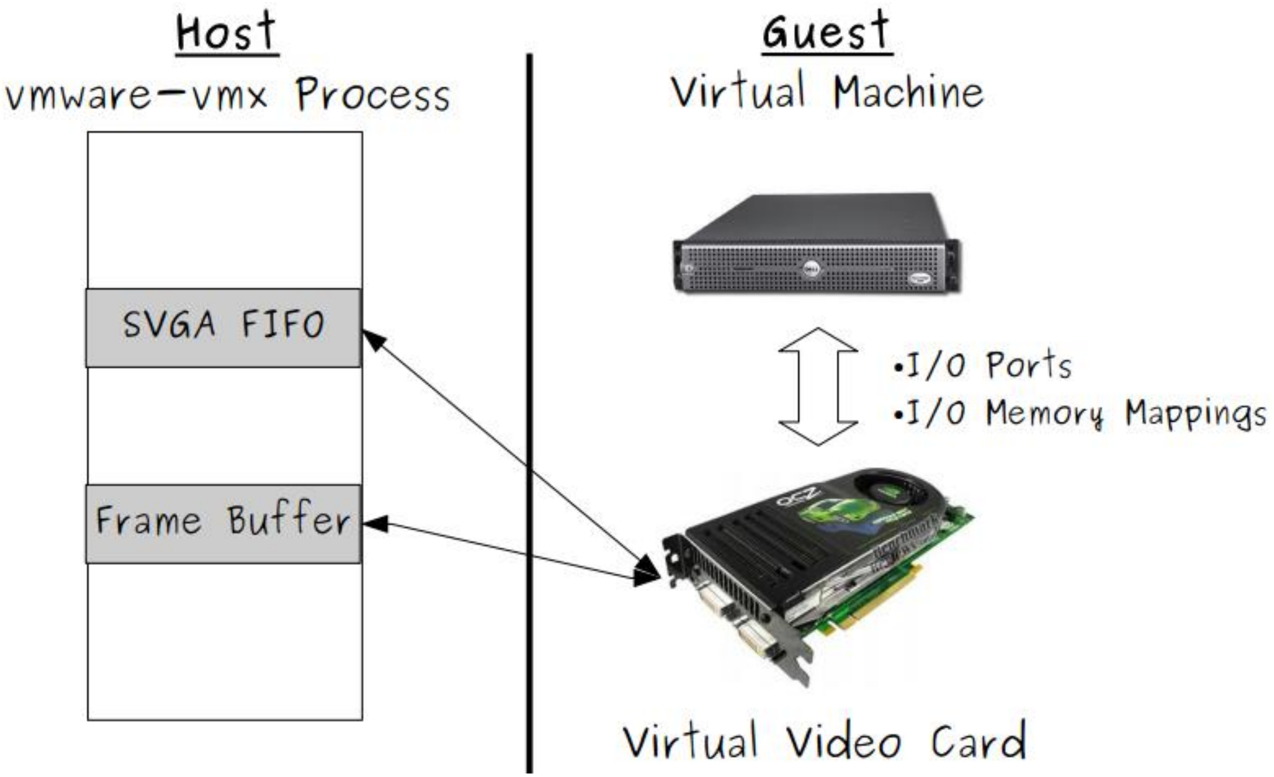


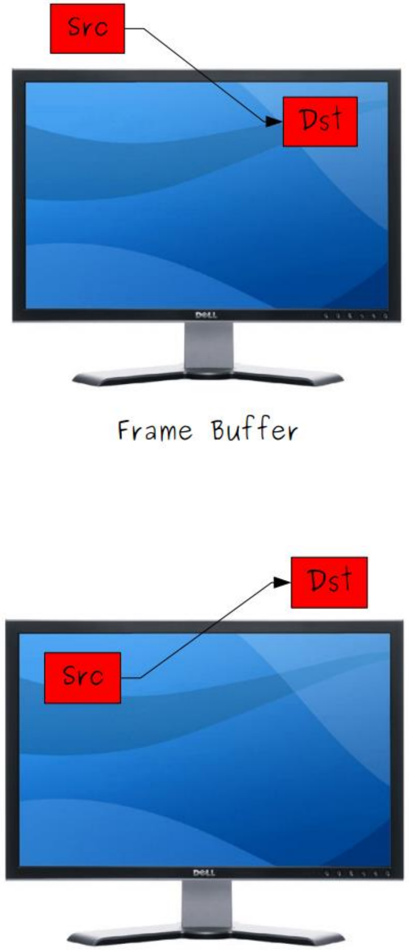
```
<?xml version="1.0" encoding="utf-8"?>
<Peach xmlns="http://peachfuzzer.com/2012/Peach" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://peachfuzzer.com/2012/Peach ../peach.xsd">
        <Include ns="Http" src="file:http_base.xml" />
        <StateModel name="RandomFuzzing" initialState="initialRandomFuzzing">
            <State name="initialRandomFuzzing">
                <Action type="output">
                    <DataModel name="Request" ref="Http:Request" />
                    <Data>
                        ...
                        ...
                    </Data>
                </Action>
                <Action type="input">
                    <DataModel name="Response" ref="Http:Response" />
                </Action>
                ...
                ...
            </State>
        </StateModel>
        <Test name="Default">
            <Agent name="Ping-Agent">
                <Monitor class="Ping">
                    ...
                </Monitor>
            </Agent>
            <Strategy class="RandomDeterministic" />
            <StateModel ref="RandomDeterministicFuzzing" />
            <Publisher class="TcpClient">
                ...
            </Publisher>
            <Logger class="File">
                <Param name="Path" value="logs" />
```

In memory fuzzing

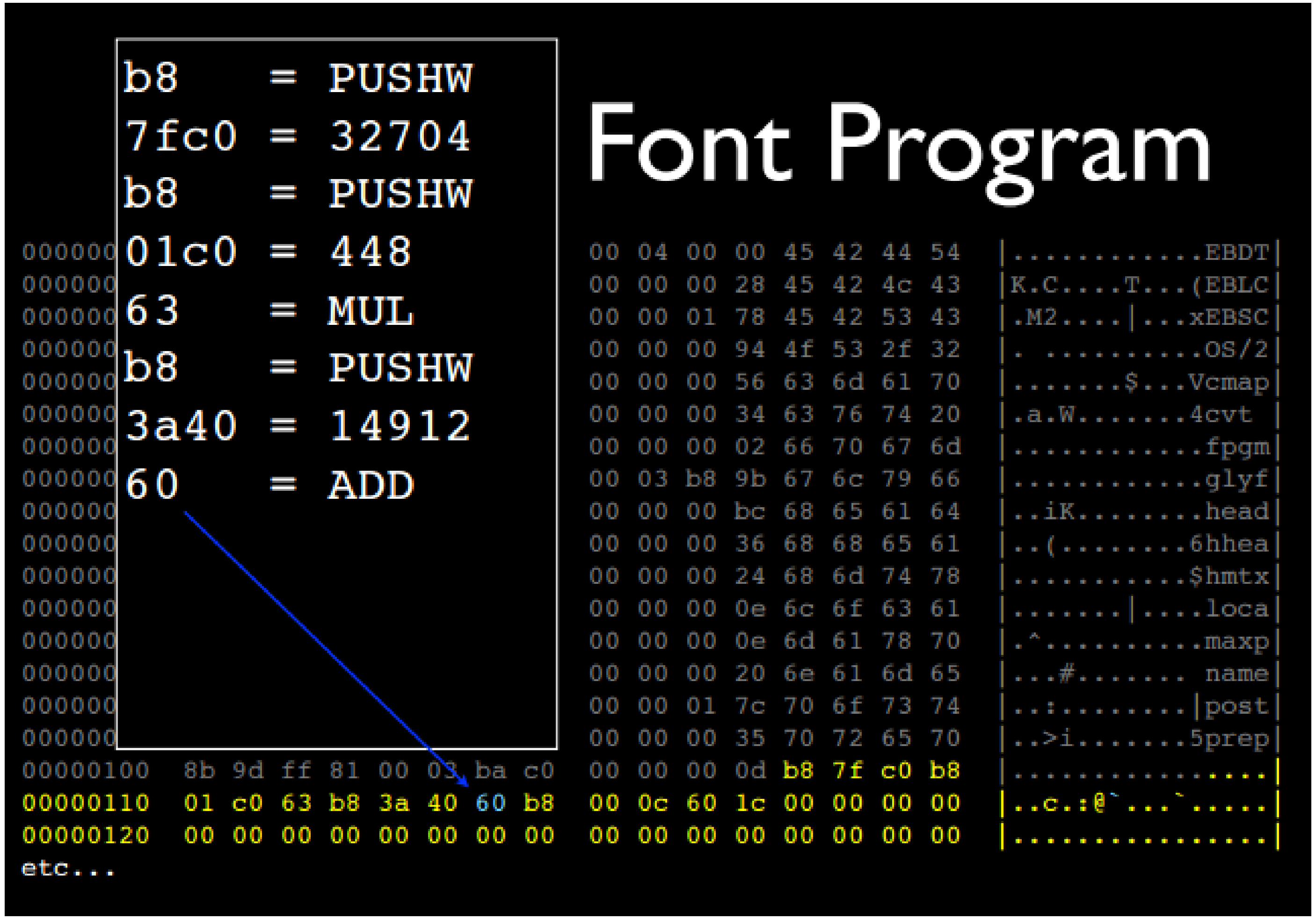


Example memory corruption: Cloudburst (Kortchinsky, 2009)





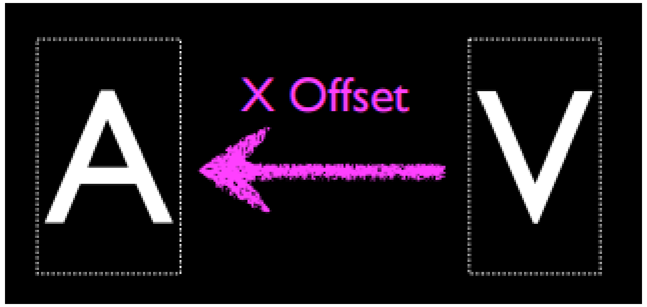
Example memory corruptions (TrueType Font Engine Vulnerability)



Font Program

```
b8    = PUSHW
7fc0  = 32704
b8    = PUSHW
01c0  = 448
63    = MUL
b8    = PUSHW
3a40  = 14912
60    = ADD
```

```
000000                               00 04 00 00 45 42 44 54  |............EBDT|
000000                               00 00 00 28 45 42 4c 43  |K.C....T...(EBLC|
000000                               00 00 01 78 45 42 53 43  |.M2....|...xEBSC|
000000                               00 00 00 94 4f 53 2f 32  |. ..........OS/2|
000000                               00 00 00 56 63 6d 61 70  |.......$...Vcmap|
000000                               00 00 00 34 63 76 74 20  |.a.W.......4cvt |
000000                               00 00 00 02 66 70 67 6d  |............fpgm|
000000                               00 03 b8 9b 67 6c 79 66  |............glyf|
000000                               00 00 00 bc 68 65 61 64  |..iK........head|
000000                               00 00 00 36 68 68 65 61  |..(........6hhea|
000000                               00 00 00 24 68 6d 74 78  |...........$hmtx|
000000                               00 00 00 0e 6c 6f 63 61  |.......|....loca|
000000                               00 00 00 0e 6d 61 78 70  |.^..........maxp|
000000                               00 00 00 20 6e 61 6d 65  |...#....... name|
000000                               00 00 01 7c 70 6f 73 74  |..:.........|post|
000000                               00 00 00 35 70 72 65 70  |..>i.......5prep|
00000100  8b 9d ff 81 00 03 ba c0   00 00 00 0d b8 7f c0 b8  |................|
00000110  01 c0 63 b8 3a 40 60 b8   00 0c 60 1c 00 00 00 00  |..c.:@`...`.....|
00000120  00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00  |................|
etc...
```



Thank you!