Security Session

Jan Kopecký
Czech chapter leader
rnmx123@gmail.com

About me

- OWASP CZ chapter leader
- Senior ethical hacker for ING
- Founder captes.cz
- Skills
  - Web security (server/client side)
  - Reverse engineering
  - Exploit development
  - Malware reversing
  - Pentesting

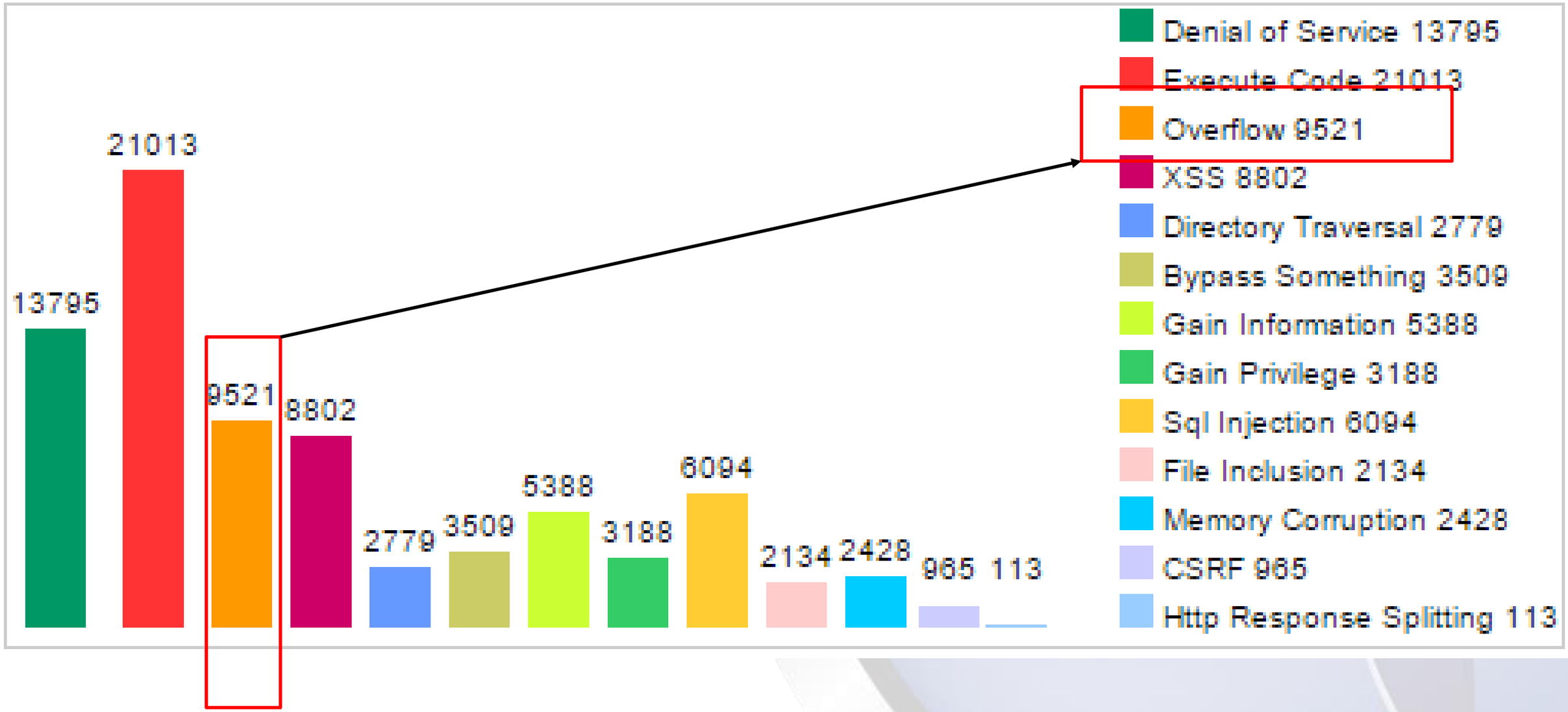
What is buffer overflow?

Exploitson XP – Stack cookie, SEH, DEP

Exploits on Win7 – ASLR

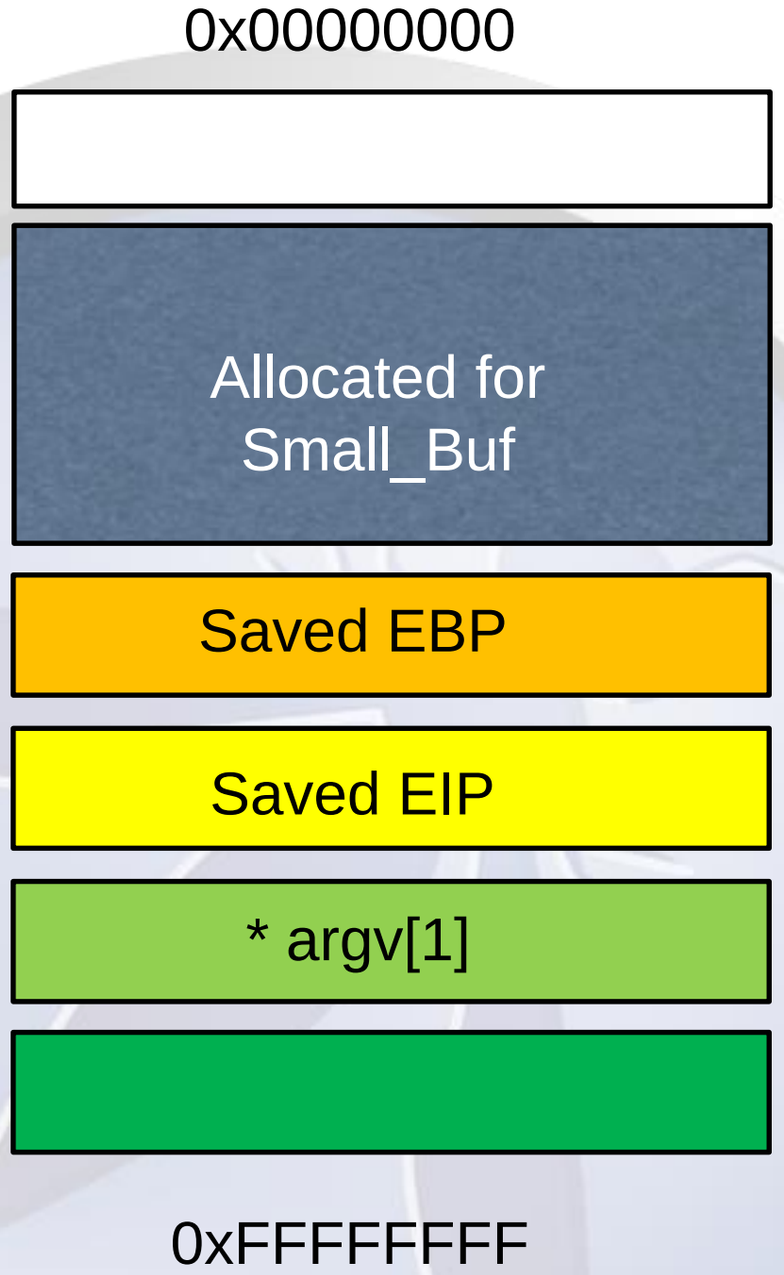
QA

Vulnerabilities By Type



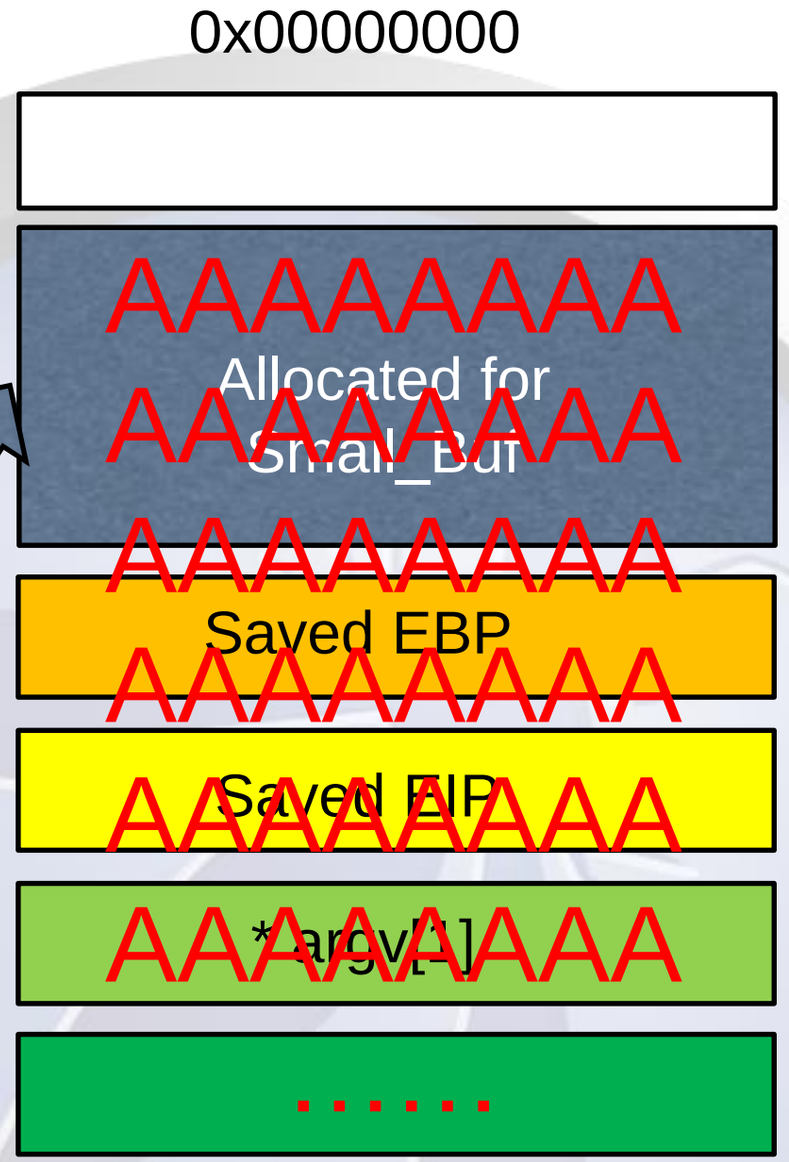
Buffer overflow

```
void TriggerOverflow(char *Buffer)
{
    char Small_Buf[512];
    strcpy(Small_Buf,Buffer);
}

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

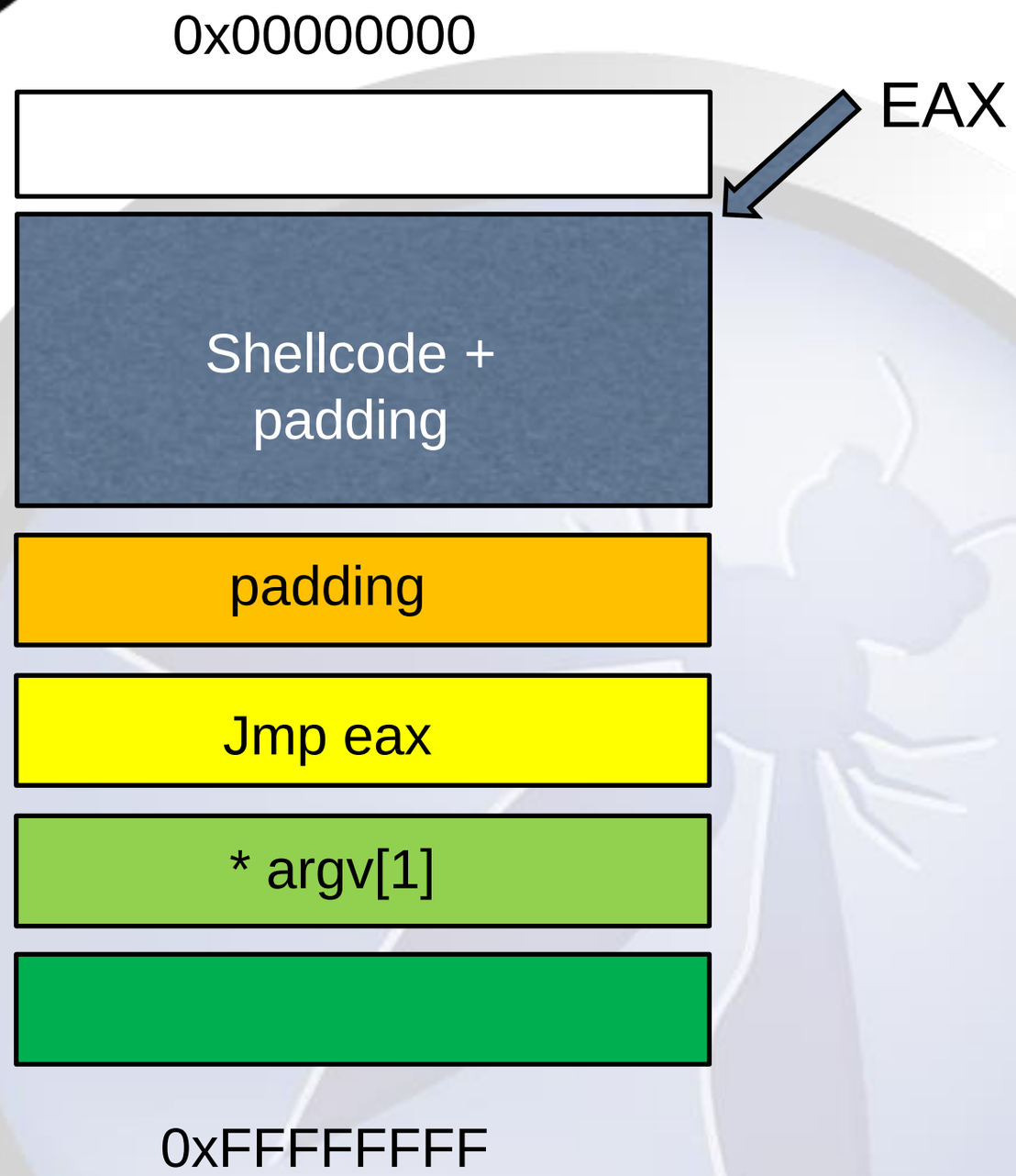


```
void TriggerOverflow(char *Buffer)
{
    char Small_Buf[512];
    strcpy(Small_Buf,Buffer);
}

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

“A” * 560



- Our goal is to overwrite EIP value which is stored in stack
- EAX points at the beginning of Small_Buf.
- When we overwritre EIP value with address of „jmp eax“ we land at the beginning of our shellcode




```
PUSH EBP
MOV EBP,ESP
SUB ESP,218
MOV EAX,DWORD PTR SS:[EBP+8]
MOV DWORD PTR SS:[ESP+4],EAX
LEA EAX,DWORD PTR SS:[EBP-208]
MOV DWORD PTR SS:[ESP],EAX
CALL <JMP.&msvcrt.strcpy>          strcpy
LEAVE
RETN
```

```
0022FF58 0022FF78 x ".
0022FF5C 004012EA ê@. RETURN to BufferOv.004012EA from BufferOv.00401290
0022FF60 003E249A Ü$>. ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
0022FF64 00670067 g.g.
0022FF68 003F2090 é #>.
```

```
0022FF24   41414141  AAAA
0022FF28   41414141  AAAA
0022FF2C   41414141  AAAA
0022FF30   41414141  AAAA
0022FF34   41414141  AAAA
0022FF38   41414141  AAAA
0022FF3C   41414141  AAAA
0022FF40   41414141  AAAA
0022FF44   41414141  AAAA
0022FF48   41414141  AAAA
0022FF4C   41414141  AAAA
0022FF50   41414141  AAAA
0022FF54   41414141  AAAA
0022FF58   41414141  AAAA
0022FF5C   11111111  ◄◄◄◄
0022FF60   42424242  BBBB
0022FF64   42424242  BBBB
0022FF68   42424242  BBBB
0022FF6C   42424242  BBBB
0022FF70   42424242  BBBB
0022FF74   42424242  BBBB
0022FF78   42424242  BBBB
0022FF7C   42424242  BBBB
0022FF80   42424242  BBBB
0022FF84   42424242  BBBB
0022FF88   42424242  BBBB
0022FF8C   42424242  BBBB
0022FF90   42424242  BBBB
0022FF94   42424242  BBBB
0022FF98   42424242  BBBB
0022FF9C   42424242  BBBB
```

DEMO

„Stack cookie“ is one of the mitigations used by Microsoft to prevent exploitation of BO.

```
LEA EDI,DWORD PTR SS:[EBP-2E4]
MOV ECX,0B3
MOV EAX,CCCCCCCC
REP STOS DWORD PTR ES:[EDI]
MOV EAX,DWORD PTR DS:[417000]          Stack cookie
XOR DWORD PTR SS:[EBP-8],EAX
XOR EAX,EBP
MOV DWORD PTR SS:[EBP-1C],EAX
PUSH EAX
```

```
CMP ECX,DWORD PTR DS:[417000]
JNZ SHORT overflow.0041154A
PREFIX REP:
RETN
JMP overflow.00411087
```

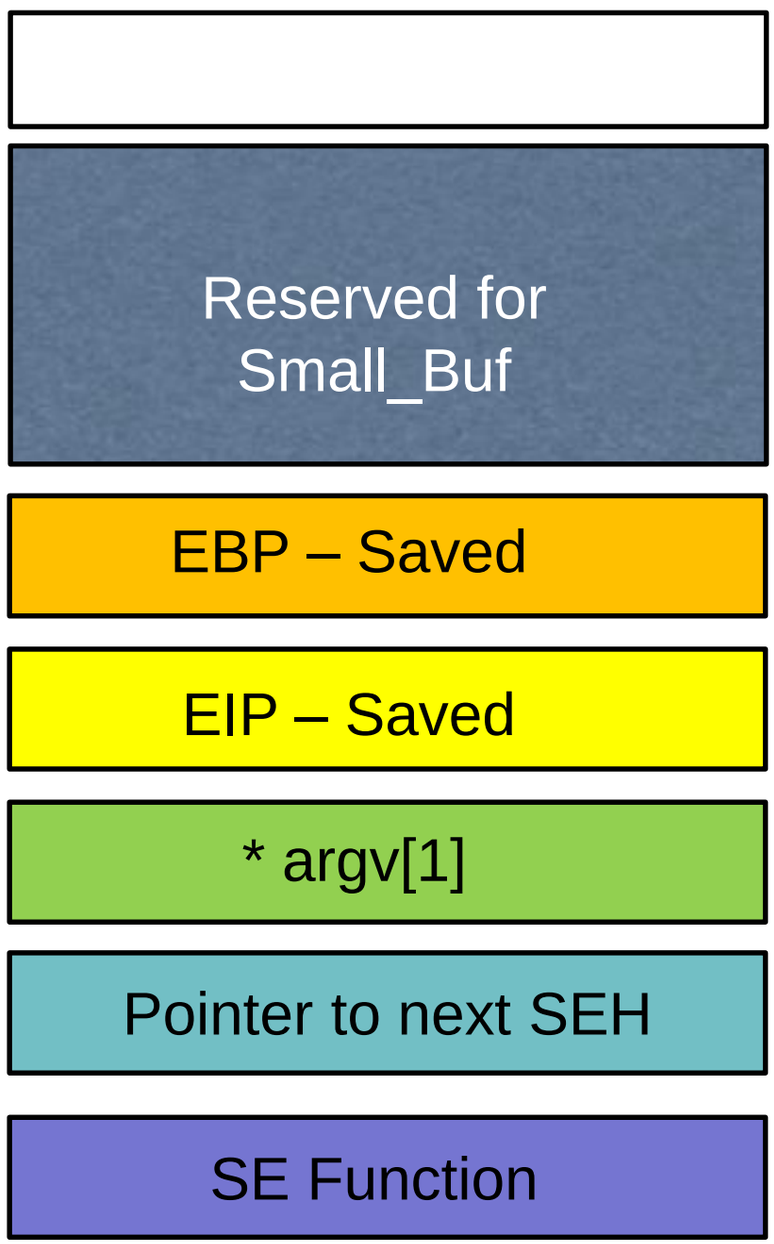
```
EAX 6FF311F8
ECX 6FF311F8
EDX 00416BD8
```

```
0012E4C0   10322667  g&2▶  RETURN to MSVCR100.10322667
0012E4C4   0012E4E4  ΣΣ$.
0012E4C8   6FF311F8  °◀≤o
0012E4CC ┌0012E510  ▶σ$.
0012E4D0 │ 10322484  ä$2▶ RETURN to MSVCR100.10322484
0012E4D4 │ 00411014  ¶▶A. overflow.00411014
0012E4D8 │ 00416BD8  ╤kA. overflow.00416BD8
0012E4DC │ 0012E9C4  −Θ$.
```

DEMO

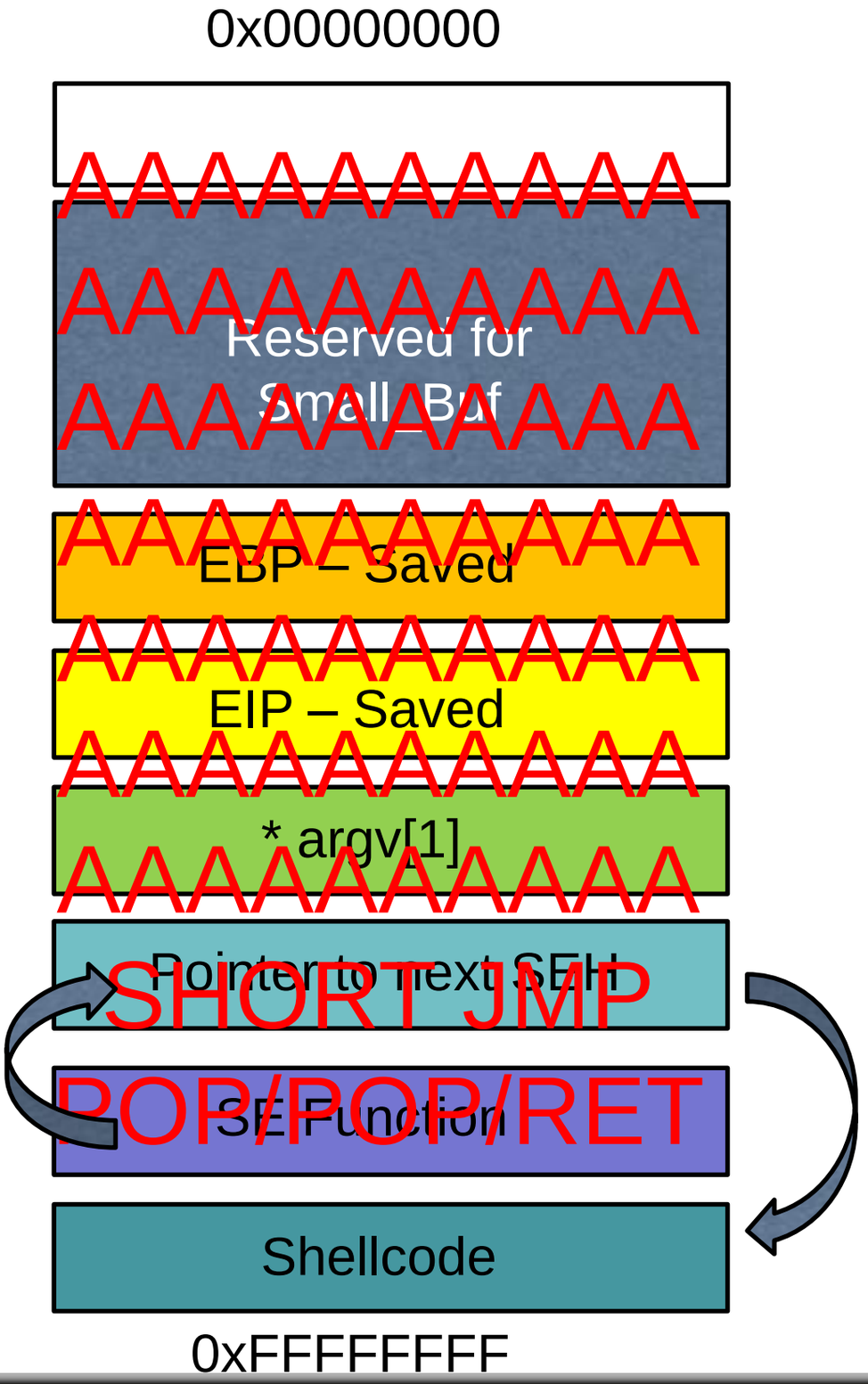
Value of cookie is compared to “master cookie”, when there is no match SEH (Exception handler) kicks in.

Program execution is terminated and shellcode will not execute.



SHE is also stored on stack, it means it can be overwritten. The question is with what?

During Exception Handler prologue, the value of “Pointer to next SEH” is stored on stack (ESP+8). If we overwrite SE Function with address of POP/POP/RET; The RET instruction will execute „Pointer to next SEH“



1. Buffer should look like:
   - SE Function -> POP/POP/RET
   - Pointer to next SEH -> SHORT JMP
2. Cookie check fail during check, our SHE is called
3. POP/POP/RET transfers execution flow (EIP) to SHORT JMP -> Shellcode

DEMO

Stack and heap are no longer executable. We cannot execute even single NOP.

OptIn : Dep is active only for some Windows services
OptOut : DEP is active for all proceses, expect whitelisted onces
AlwaysOn : DEP is always on
AlwaysOff : DEP is always off

Windows XP SP2, XP SP3, Vista SP0 : OptIn
Windows Vista SP1 : OptIn + Permanent DEP
Windows 7: OptIn + Permanent DEP
Windows Server 2003 SP1 and up : OptOut
Windows Server 2008 and up : OptOut + Permanent DEP

Permanent DEP – works when executable compiled with /NXCOMPAT

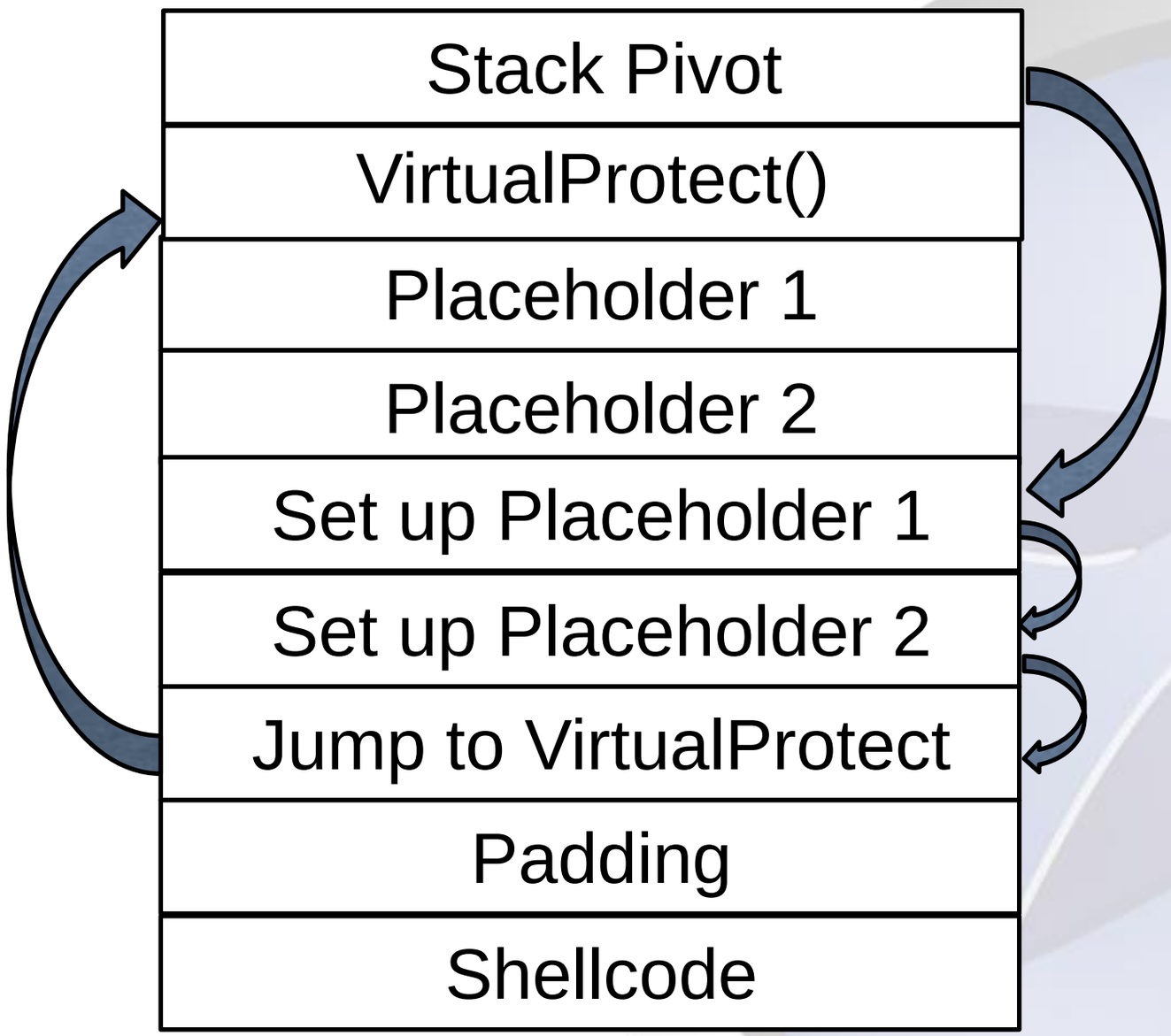
We can overwrite buffer in same way you just saw, but we cannot execute nothing.

Thankfully Dino Dai Zovi defined ROP(Return Oriented Programing):

Instruction + RET = Gadget

We have to re-use code in the application (must be „executable“) in such a way we set up all necessary arguments and execute API to disable DEP.

- VirtualAlloc()
- SetProcessDEPPolicy()
- NtSetInformationProcess()
- VirtualProtect()
- WriteProcessMemory()


Stack
Stack Pivot
VirtualProtect()
Placeholder 1
Placeholder 2
Set up Placeholder 1
Set up Placeholder 2
Jump to VirtualProtect
Padding
Shellcode

Goal: Get „05050505“ to EBX and add 60. Save result to EAX.

1. Gadget POP EBX; RET
2. Gadget ADD EBX, 60; RET
3. Gadget MOV EAX, EBX;
   POP EDX;
   RET

34BA4378
00000005
34BA9A78
34BA4FC8
DEADBEEF

34BA4378 POP EBX; RET
34BA9A78 ADD EBX, 60; RET
34BA4FC8 MOV EAX, EBX;
POP EDX;
RET

Let’s find out whether we can execute gadgets...

DEMO

… and arbitrary code

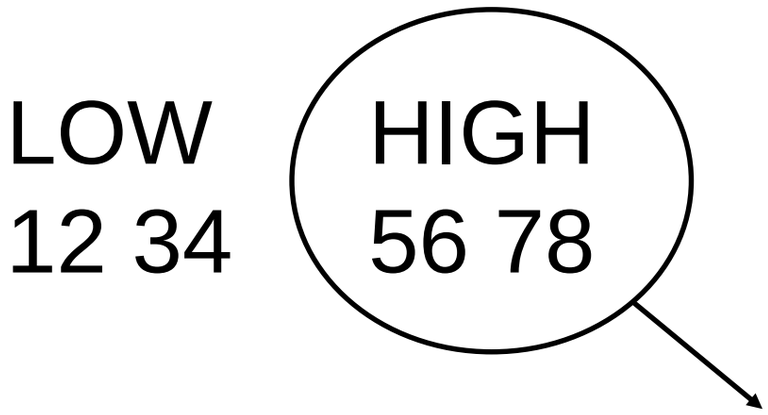
Address Space Layout Randomization (/DYNAMICBASE)

We can no longer rely on static addresses as in following examples:

004013FD JMP EAX
10024F5A POP/POP/RET
7C14DF3C CALL VirtualProtect

After every reboot module is loaded in random memory address.

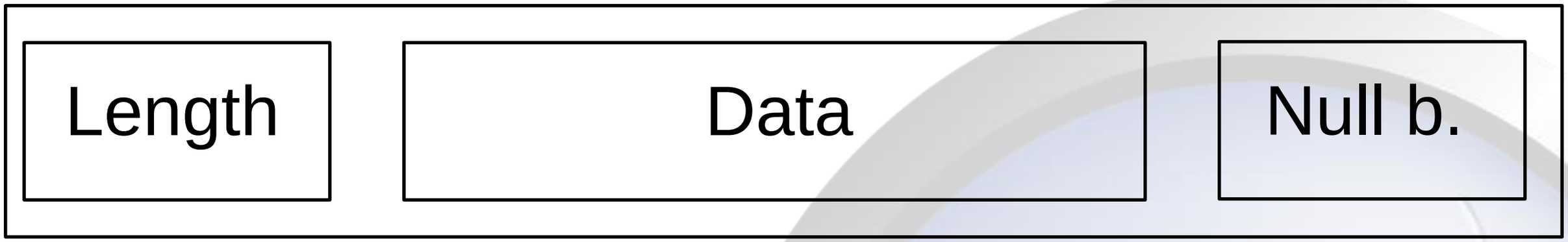
- Partial overwrite
- Non-ASLR module – still usable, problem Force ASLR
- „Leaks“ – too complex to cover – maybe next time
- Length field manipulation BSTR (not only) – most common technique nowadays, especialy when dealing with UAF



Only this part is randomized. We have to find useful instruction somewhere in 5678 XXXX .

- !mona module
- !mona jmp –r esp

DEMO



If we manage to change length field of string in proper way we will have option to read to whole address space of the browser.

DEMO

Our exploit must do following in order to bypass DEP and ASLR:

Find which object we corrupted
Corrupt size of next object
Figure out base address of flash module (MZ header scan)
Figure out Kernel32 address (IAT)
Locate VirtualProtect
Dynamicaly build ROP
Run ROP

DEMO

Q&A