The Presentation for Cross-Site Scripters Who Can't Stack Buffer Overflow Good and Want to Do Other Stuff Good Too

@justinsteven

Last updated 2016-07-27

whoami

- PSIRT/CSIRT
- OSCP
- OSCE

Agenda

- Some housekeeping
- Some theory
- A demo
- Closing thoughts

This is old stuff

Easy Mode

No ASLR

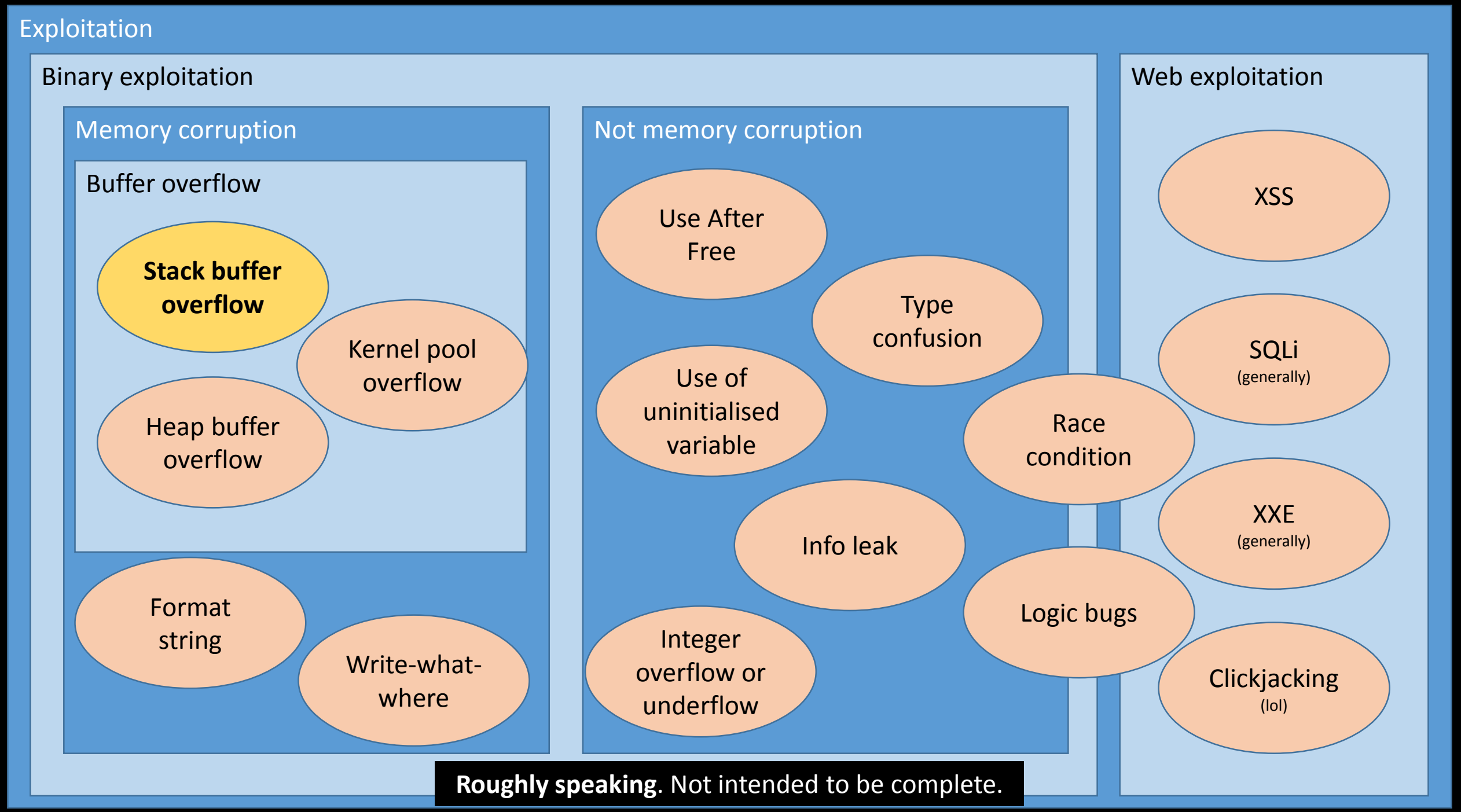
No DEP

No stack canaries

WoW64

Stack Buffer Overflows

(Saved Return Pointer Overwrites in particular)


Exploitation
Binary exploitation
Memory corruption
Buffer overflow
Stack buffer overflow
Kernel pool overflow
Heap buffer overflow
Format string
Write-what-where
Not memory corruption
Use After Free
Type confusion
Use of uninitialised variable
Race condition
Info leak
Logic bugs
Integer overflow or underflow
Web exploitation
XSS
SQLi (generally)
XXE (generally)
Clickjacking (lol)
Roughly speaking. Not intended to be complete.

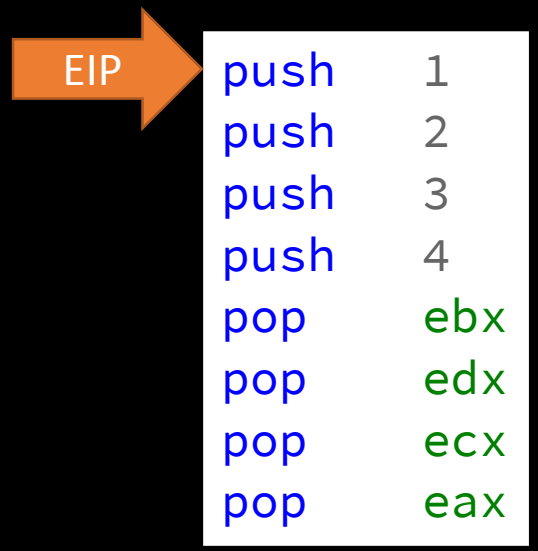
Registers

EAX, ECX, EDX, EBX, ESI, EDI

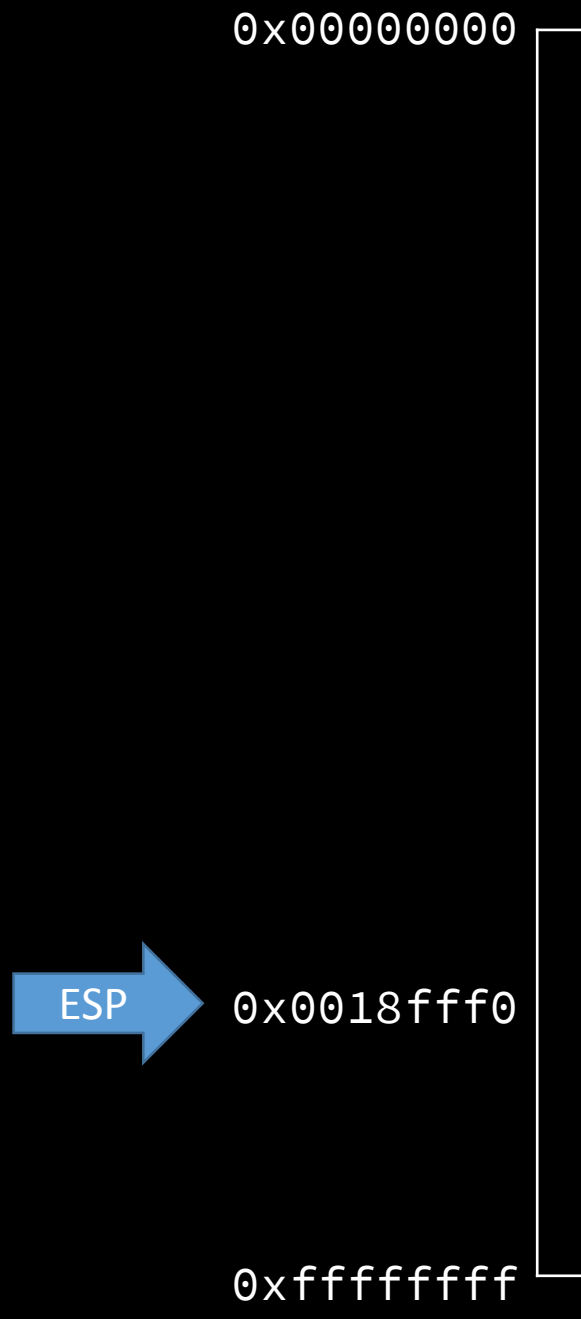
EBP, ESP

EIP

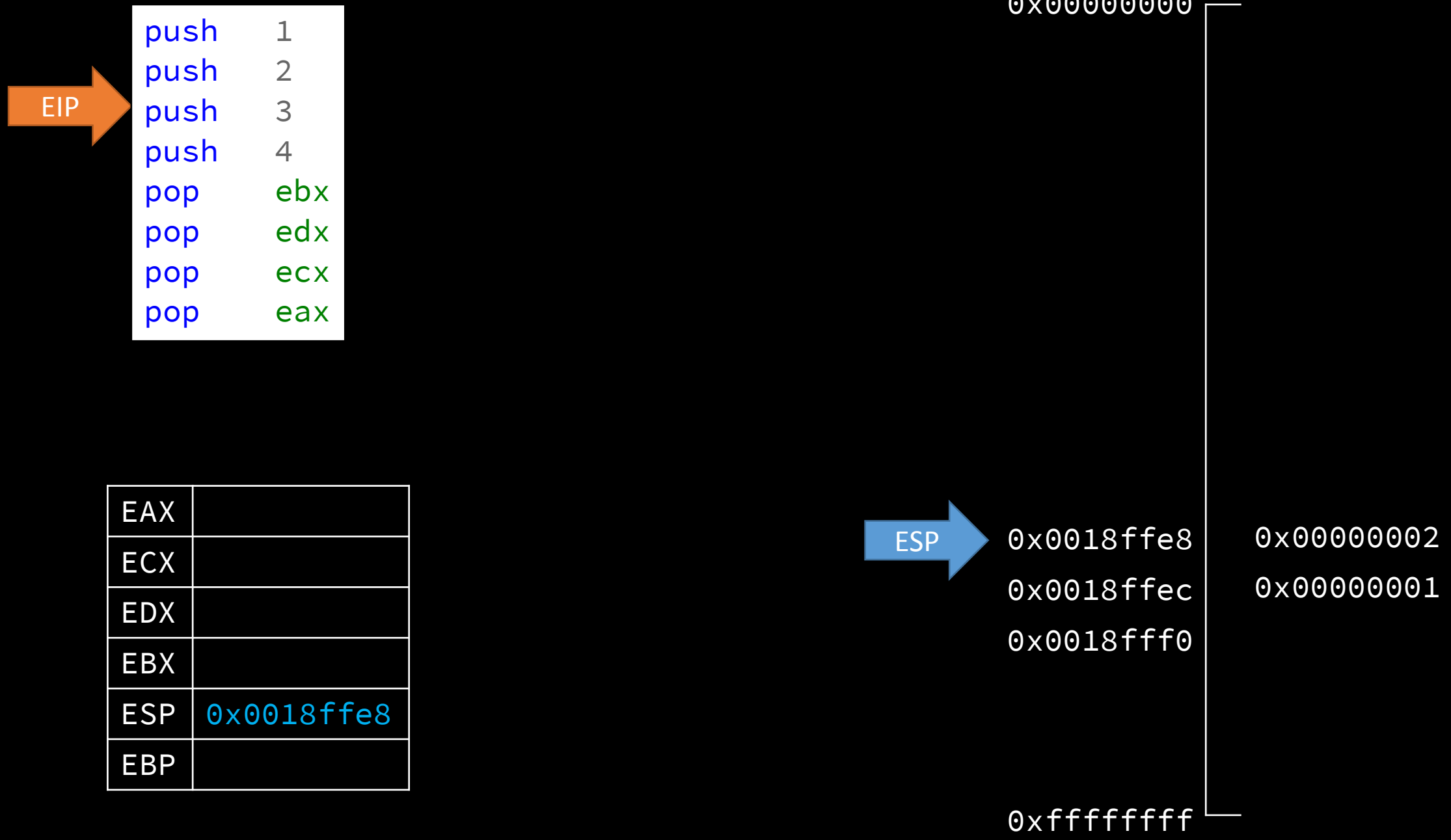
Stacks



EAX	
ECX	
EDX	
EBX	
ESP	0x0018fff0
EBP	




EIP
push 1
push 2
push 3
push 4
pop ebx
pop edx
pop ecx
pop eax
EAX
ECX
EDX
EBX
ESP 0x0018ffec
EBP
0x00000000
ESP
0x0018ffec
0x0018fff0
0x00000001
0xffffffff


EIP
push 1
push 2
push 3
push 4
pop ebx
pop edx
pop ecx
pop eax
EAX
ECX
EDX
EBX
ESP 0x0018ffe8
EBP
0x00000000
ESP
0x0018ffe8
0x0018ffec
0x0018fff0
0x00000002
0x00000001
0xffffffff


push 1
push 2
push 3
push 4
pop ebx
pop edx
pop ecx
pop eax
EIP
EAX
ECX
EDX
EBX
ESP 0x0018ffe4
EBP
0x00000000
ESP
0x0018ffe4 0x00000003
0x0018ffe8 0x00000002
0x0018ffec 0x00000001
0x0018fff0
0xffffffff


push 1
push 2
push 3
push 4
pop ebx
pop edx
pop ecx
pop eax
EIP
EAX
ECX
EDX
EBX
ESP 0x0018ffe0
EBP
ESP
0x00000000
0x0018ffe0
0x0018ffe4
0x0018ffe8
0x0018ffec
0x0018fff0
0xffffffff
0x00000004
0x00000003
0x00000002
0x00000001


push 1
push 2
push 3
push 4
pop ebx
pop edx
pop ecx
pop eax
EIP
EAX
ECX
EDX
EBX 0x00000004
ESP 0x0018ffe4
EBP
0x00000000
0x0018ffe0
0x0018ffe4
0x0018ffe8
0x0018ffec
0x0018fff0
0xffffffff
ESP
0x00000004
0x00000003
0x00000002
0x00000001


push 1
push 2
push 3
push 4
pop ebx
pop edx
pop ecx
pop eax
EIP
EAX
ECX
EDX 0x00000003
EBX 0x00000004
ESP 0x0018ffe8
EBP
0x00000000
0x0018ffe0 0x00000004
0x0018ffe4 0x00000003
ESP
0x0018ffe8 0x00000002
0x0018ffec 0x00000001
0x0018fff0
0xffffffff


push 1
push 2
push 3
push 4
pop ebx
pop edx
pop ecx
pop eax
EIP
EAX
ECX 0x00000002
EDX 0x00000003
EBX 0x00000004
ESP 0x0018ffec
EBP
0x00000000
0x0018ffe0 0x00000004
0x0018ffe4 0x00000003
0x0018ffe8 0x00000002
ESP
0x0018ffec 0x00000001
0x0018fff0
0xffffffff

```
push    1
push    2
push    3
push    4
pop     ebx
pop     edx
pop     ecx
pop     eax
```

EIP

EAX	0x00000001
ECX	0x00000002
EDX	0x00000003
EBX	0x00000004
ESP	0x0018fff0
EBP	


0x00000000
\x04\x00\x00\x00\x03\x00\x00\x00\x02\x00\x00\x00\x01\x00\x00\x00
0x0018ffe0 0x00000004
0x0018ffe4 0x00000003
0x0018ffe8 0x00000002
0x0018ffec 0x00000001
0x0018fff0
0xffffffff

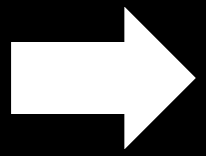
Function CALL/RETN mechanics

```
void main() {
  foo();
  throw a debugger breakpoint;
}

void foo() {
  char mystring[16];

  fill mystring with A's;

  return;
}
```

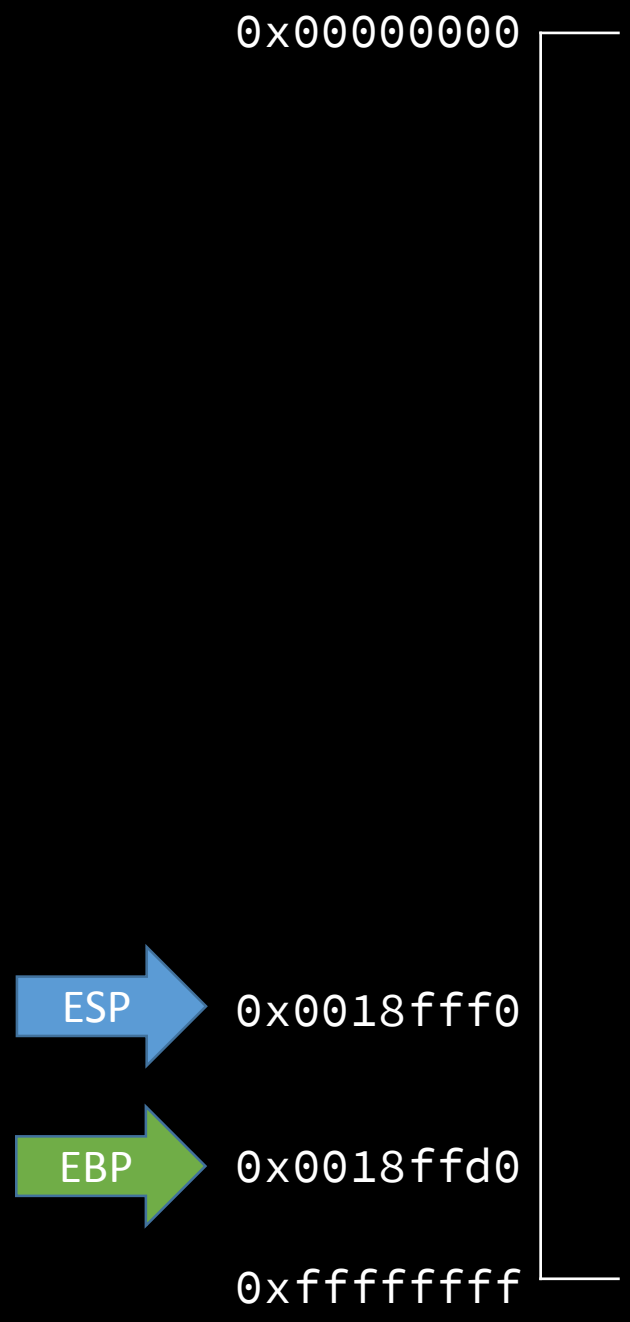


```
0x08040000:   call    0x08041000
0x08040005:   int3
...
0x08041000:   push    ebp
0x08041001:   mov     ebp, esp
0x08041003:   sub     esp, 0x10
0x08041006:   fill l-vars w/ "A"
0x08041010:   mov     esp, ebp
0x08041012:   pop     ebp
0x08041013:   retn
```

EIP

```
0x08040000:   call   0x08041000
0x08040005:   int3
...
0x08041000:   push   ebp
0x08041001:   mov    ebp, esp
0x08041003:   sub    esp, 0x10
0x08041006:   fill l-vars w/ "A"
0x08041010:   mov    esp, ebp
0x08041012:   pop    ebp
0x08041013:   retn
```

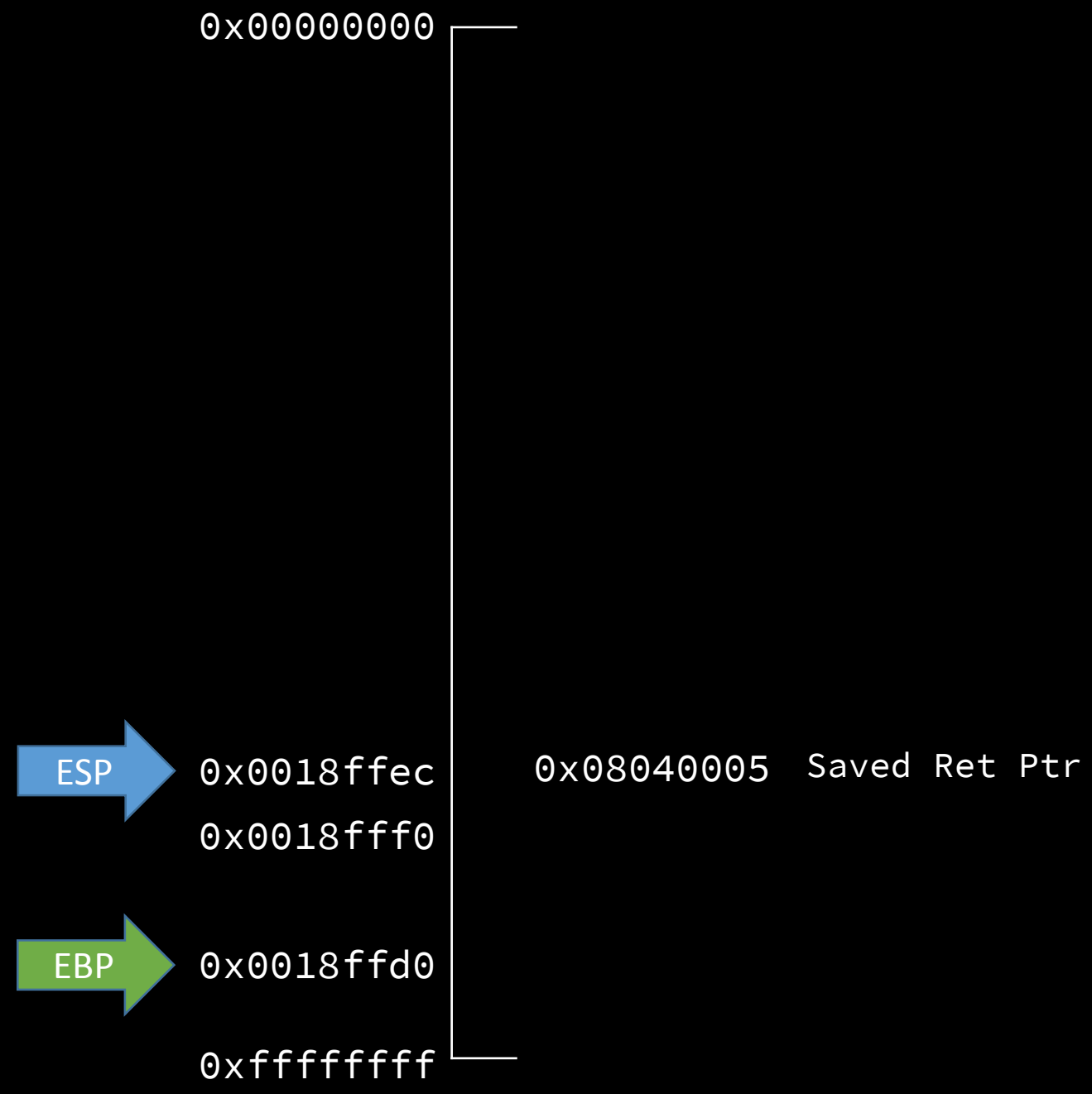
EAX	
ECX	
EDX	
EBX	
ESP	0x0018fff0
EBP	0x0018ffd0



```
0x08040000:   call   0x08041000
0x08040005:   int3
...
0x08041000:   push   ebp
0x08041001:   mov    ebp, esp
0x08041003:   sub    esp, 0x10
0x08041006:   fill l-vars w/ "A"
0x08041010:   mov    esp, ebp
0x08041012:   pop    ebp
0x08041013:   retn
```

EIP

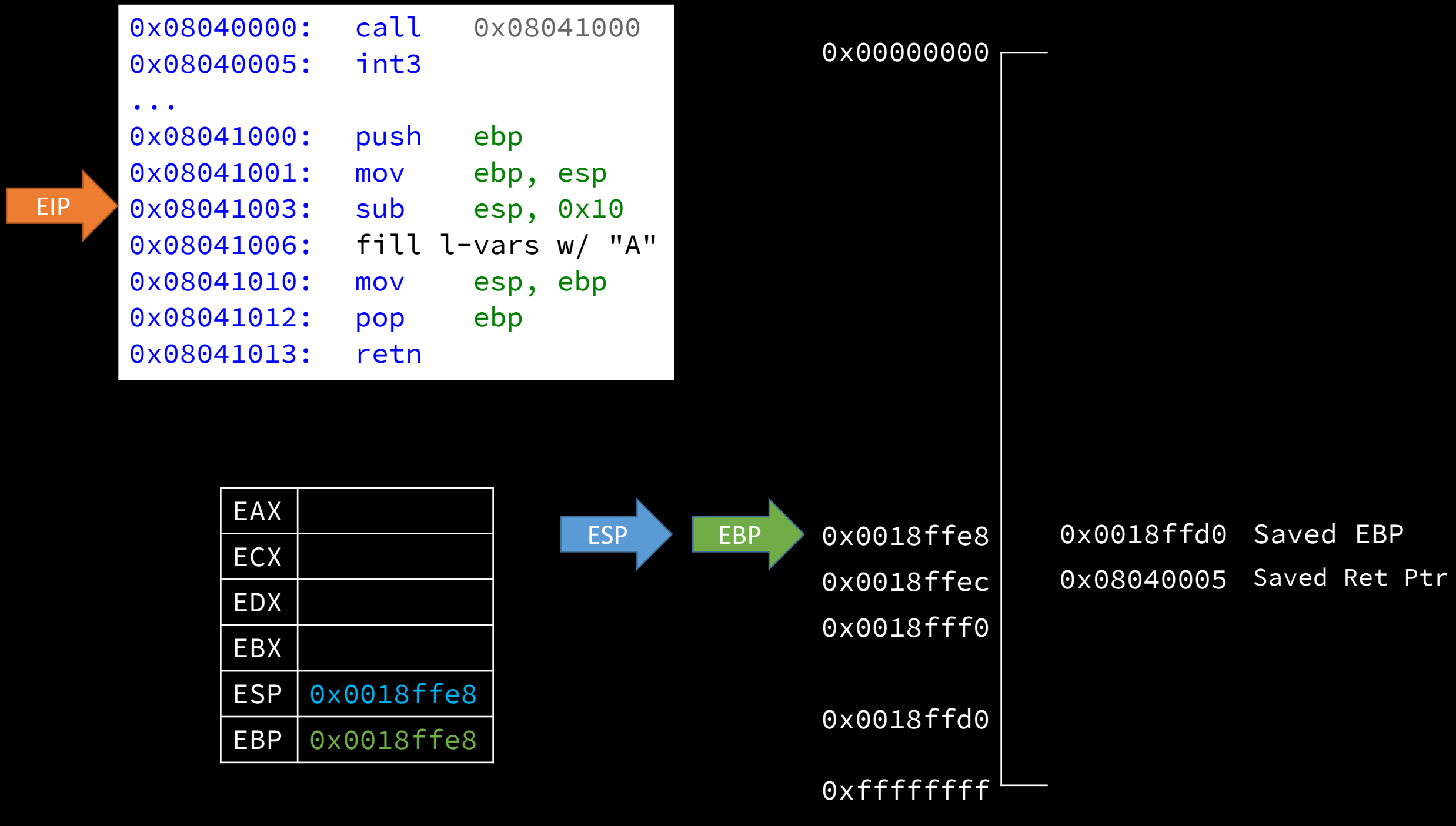
EAX	
ECX	
EDX	
EBX	
ESP	0x0018ffec
EBP	0x0018ffd0

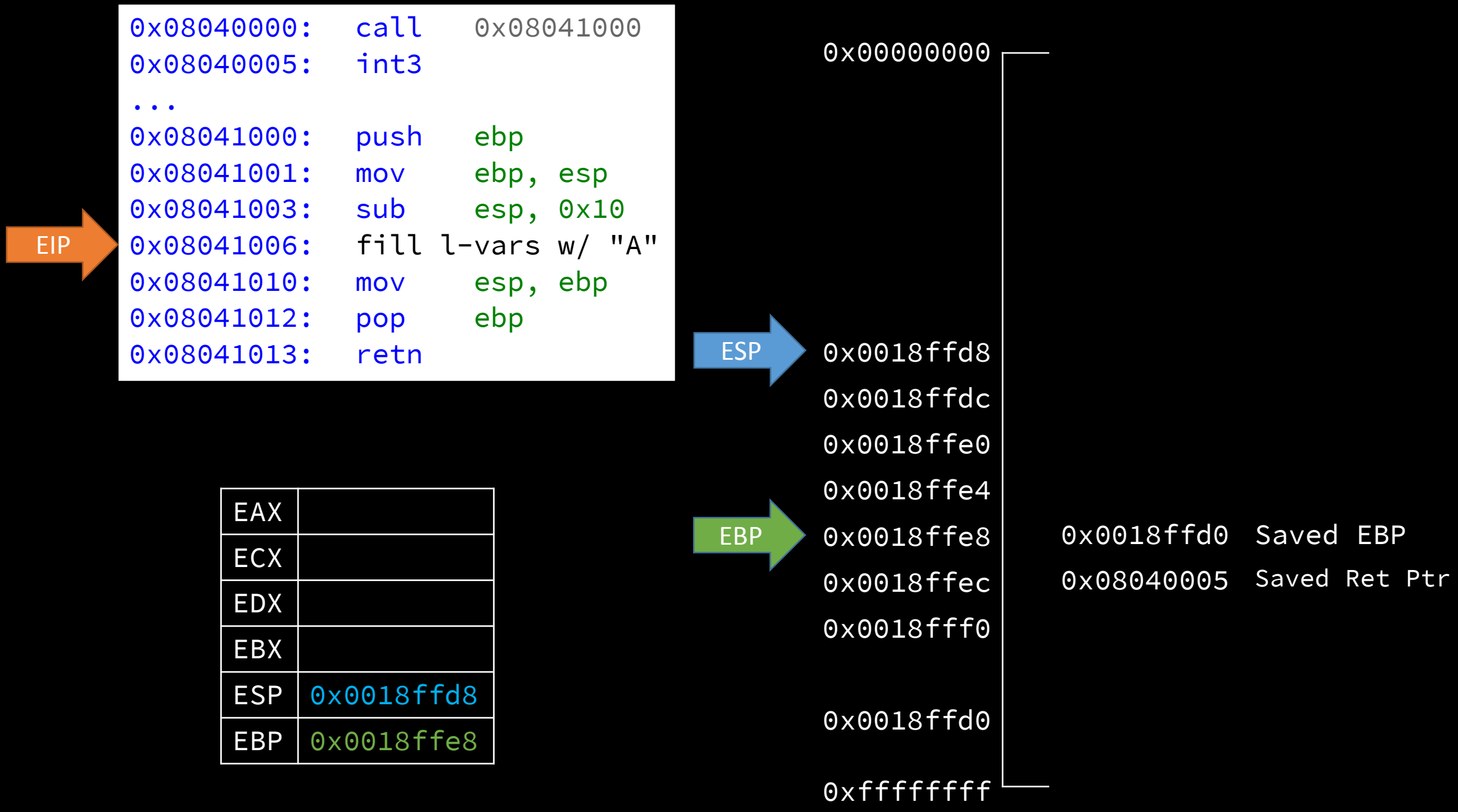


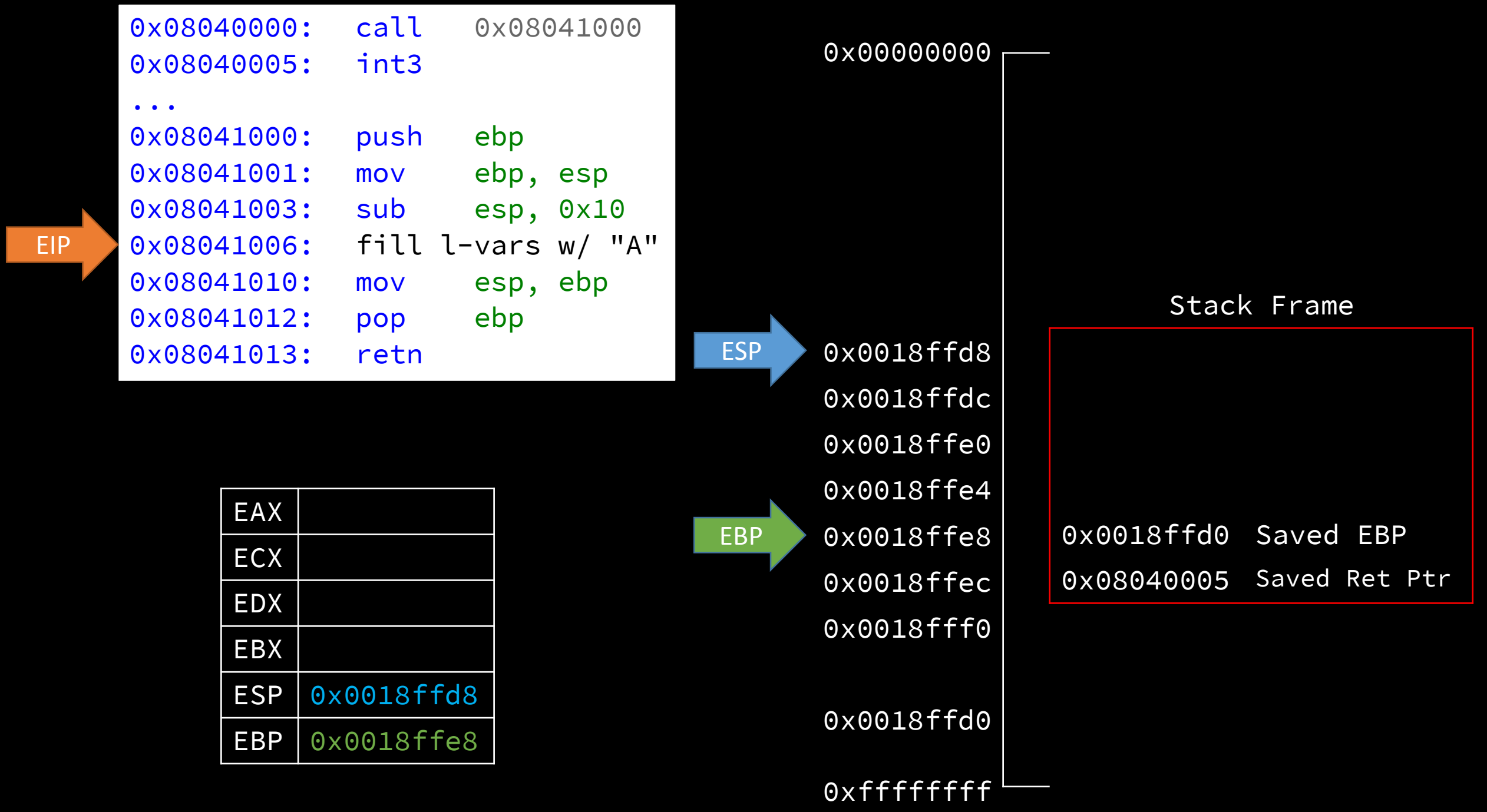
```
0x08040000:   call   0x08041000
0x08040005:   int3
...
0x08041000:   push   ebp
0x08041001:   mov    ebp, esp
0x08041003:   sub    esp, 0x10
0x08041006:   fill l-vars w/ "A"
0x08041010:   mov    esp, ebp
0x08041012:   pop    ebp
0x08041013:   retn
```

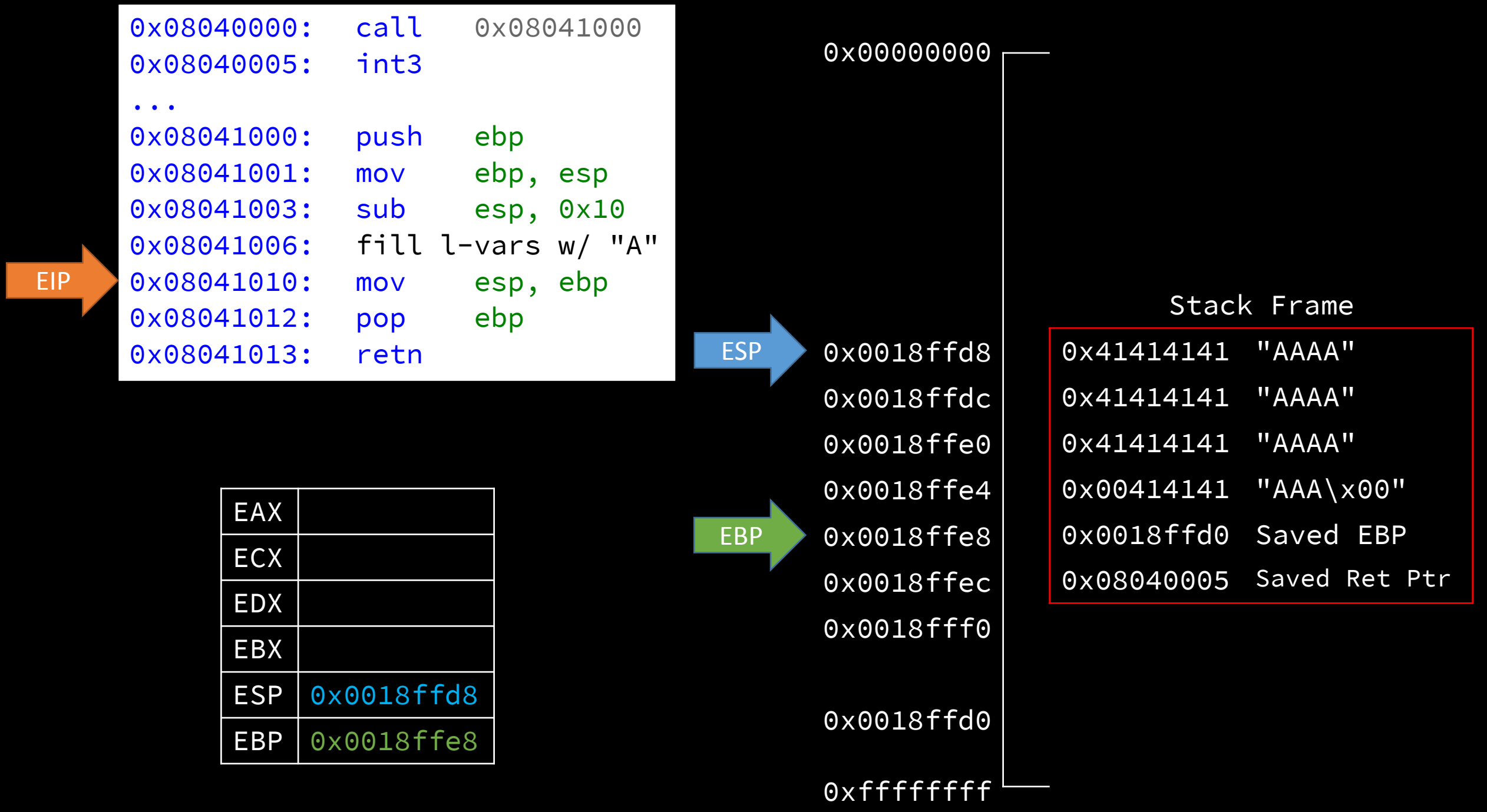
EIP

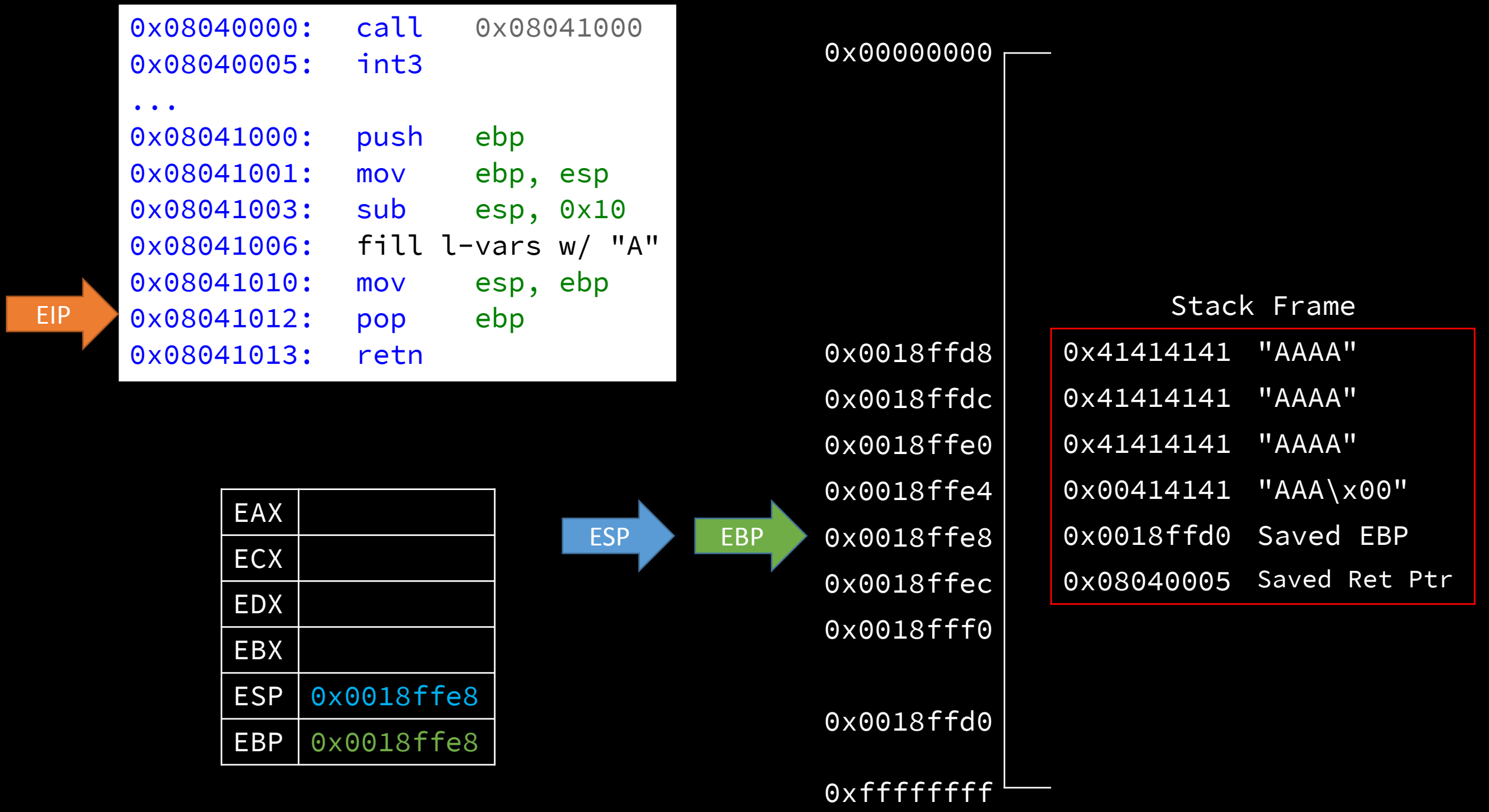
EAX	
ECX	
EDX	
EBX	
ESP	0x0018ffe8
EBP	0x0018ffd0


0x08040000: call 0x08041000
0x08040005: int3
...
0x08041000: push ebp
0x08041001: mov ebp, esp
0x08041003: sub esp, 0x10
0x08041006: fill l-vars w/ "A"
0x08041010: mov esp, ebp
0x08041012: pop ebp
0x08041013: retn
EIP
EAX
ECX
EDX
EBX
ESP 0x0018ffe8
EBP 0x0018ffe8
ESP
EBP
0x00000000
0x0018ffe8
0x0018ffec
0x0018fff0
0x0018ffd0
0xffffffff
0x0018ffd0 Saved EBP
0x08040005 Saved Ret Ptr


0x08040000: call 0x08041000
0x08040005: int3
...
0x08041000: push ebp
0x08041001: mov ebp, esp
0x08041003: sub esp, 0x10
0x08041006: fill l-vars w/ "A"
0x08041010: mov esp, ebp
0x08041012: pop ebp
0x08041013: retn
EIP
ESP
EBP
EAX
ECX
EDX
EBX
ESP 0x0018ffd8
EBP 0x0018ffe8
0x00000000
0x0018ffd8
0x0018ffdc
0x0018ffe0
0x0018ffe4
0x0018ffe8
0x0018ffec
0x0018fff0
0x0018ffd0
0xffffffff
0x0018ffd0 Saved EBP
0x08040005 Saved Ret Ptr


0x08040000: call 0x08041000
0x08040005: int3
...
0x08041000: push ebp
0x08041001: mov ebp, esp
0x08041003: sub esp, 0x10
0x08041006: fill l-vars w/ "A"
0x08041010: mov esp, ebp
0x08041012: pop ebp
0x08041013: retn
EIP
ESP
EBP
EAX
ECX
EDX
EBX
ESP 0x0018ffd8
EBP 0x0018ffe8
0x00000000
0x0018ffd8
0x0018ffdc
0x0018ffe0
0x0018ffe4
0x0018ffe8
0x0018ffec
0x0018fff0
0x0018ffd0
0xffffffff
Stack Frame
0x0018ffd0 Saved EBP
0x08040005 Saved Ret Ptr

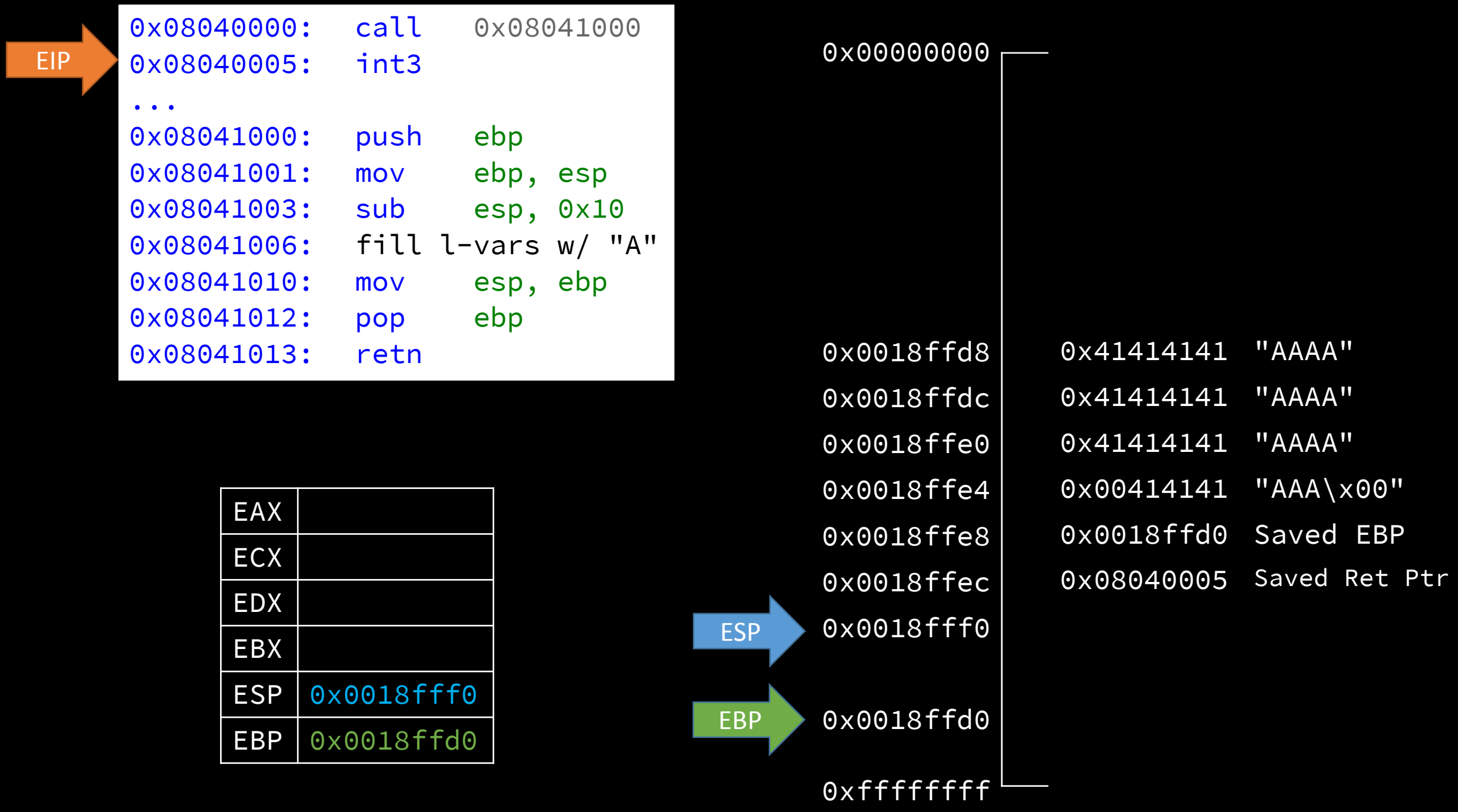

0x08040000: call 0x08041000
0x08040005: int3
...
0x08041000: push ebp
0x08041001: mov ebp, esp
0x08041003: sub esp, 0x10
0x08041006: fill l-vars w/ "A"
0x08041010: mov esp, ebp
0x08041012: pop ebp
0x08041013: retn
EIP
EAX
ECX
EDX
EBX
ESP 0x0018ffd8
EBP 0x0018ffe8
0x00000000
ESP
0x0018ffd8
0x0018ffdc
0x0018ffe0
0x0018ffe4
EBP
0x0018ffe8
0x0018ffec
0x0018fff0
0x0018ffd0
0xffffffff
Stack Frame
0x41414141 "AAAA"
0x41414141 "AAAA"
0x41414141 "AAAA"
0x00414141 "AAA\x00"
0x0018ffd0 Saved EBP
0x08040005 Saved Ret Ptr


0x08040000: call 0x08041000
0x08040005: int3
...
0x08041000: push ebp
0x08041001: mov ebp, esp
0x08041003: sub esp, 0x10
0x08041006: fill l-vars w/ "A"
0x08041010: mov esp, ebp
0x08041012: pop ebp
0x08041013: retn
EIP
EAX
ECX
EDX
EBX
ESP 0x0018ffe8
EBP 0x0018ffe8
ESP
EBP
0x00000000
0x0018ffd8
0x0018ffdc
0x0018ffe0
0x0018ffe4
0x0018ffe8
0x0018ffec
0x0018fff0
0x0018ffd0
0xffffffff
Stack Frame
0x41414141 "AAAA"
0x41414141 "AAAA"
0x41414141 "AAAA"
0x00414141 "AAA\x00"
0x0018ffd0 Saved EBP
0x08040005 Saved Ret Ptr

```
0x08040000:   call   0x08041000
0x08040005:   int3
...
0x08041000:   push   ebp
0x08041001:   mov    ebp, esp
0x08041003:   sub    esp, 0x10
0x08041006:   fill l-vars w/ "A"
0x08041010:   mov    esp, ebp
0x08041012:   pop    ebp
0x08041013:   retn
```

EIP

EAX	
ECX	
EDX	
EBX	
ESP	0x0018ffec
EBP	0x0018ffd0


EIP
0x08040000: call 0x08041000
0x08040005: int3
...
0x08041000: push ebp
0x08041001: mov ebp, esp
0x08041003: sub esp, 0x10
0x08041006: fill l-vars w/ "A"
0x08041010: mov esp, ebp
0x08041012: pop ebp
0x08041013: retn
EAX
ECX
EDX
EBX
ESP 0x0018fff0
EBP 0x0018ffd0
0x00000000
0x0018ffd8 0x41414141 "AAAA"
0x0018ffdc 0x41414141 "AAAA"
0x0018ffe0 0x41414141 "AAAA"
0x0018ffe4 0x00414141 "AAA\x00"
0x0018ffe8 0x0018ffd0 Saved EBP
0x0018ffec 0x08040005 Saved Ret Ptr
ESP
0x0018fff0
EBP
0x0018ffd0
0xffffffff


0x08040000: call 0x08041000
0x08040005: int3
...
0x08041000: push ebp
0x08041001: mov ebp, esp
0x08041003: sub esp, 0x10
0x08041006: fill l-vars w/ "A"
0x08041010: mov esp, ebp
0x08041012: pop ebp
0x08041013: retn
EIP
Debugger (if attached) breaks execution
EAX
ECX
EDX
EBX
ESP 0x0018fff0
EBP 0x0018ffd0
0x00000000
0x0018ffd8 0x41414141 "AAAA"
0x0018ffdc 0x41414141 "AAAA"
0x0018ffe0 0x41414141 "AAAA"
0x0018ffe4 0x00414141 "AAA\x00"
0x0018ffe8 0x0018ffd0 Saved EBP
0x0018ffec 0x08040005 Saved Ret Ptr
ESP
0x0018fff0
EBP
0x0018ffd0
0xffffffff


0x00000000
0x0018ffd8 0x41414141 "AAAA"
0x0018ffdc 0x41414141 "AAAA"
0x0018ffe0 0x41414141 "AAAA"
0x0018ffe4 0x00414141 "AAA\x00"
0x0018ffe8 0x0018ffd0 Saved EBP
0x0018ffec 0x08040005 Saved Ret Ptr
0x0018fff0
\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41\x41
\x41\x41\x41\x00\xd0\xff\x18\x00\x05\x00\x04\x08
0xffffffff

Demo

http://www.immunityinc.com/products/debugger/

https://github.com/corelan/mona

https://github.com/rapid7/metasploit-framework

```c
// dostackbufferoverflowgood.c
int __cdecl main() {
  // SNIP (network socket setup)
  while (1) {
    // SNIP (Accept connection as clientSocket)
    // SNIP run handleConnection() in a thread to handle the connection
  }
}
void __cdecl handleConnection(void *param) {
  SOCKET clientSocket = (SOCKET)param;

  while (1) {
    // SNIP recv() from the socket into recvbuf
    // SNIP for each newline-delimited "chunk" of recvbuf do:
      doResponse(clientSocket, line_start);
 }
}
int __cdecl doResponse(SOCKET clientSocket, char *clientName) {
  char response[128];

  // Build response
  sprintf(response, "Hello %s!!!\n", clientName);

  // Send response to the client
  int result = send(clientSocket, response, strlen(response), 0);

  // SNIP - some error handling for send()

  return 0;
}
```

Too long; didn’t listen (1/2)

- Trigger the bug
  - Send lots of A’s
  - Expect a crash at 0x41414141
- Discover offsets
  - Metasploit’s pattern_create.rb
  - !mona findmsp
- Test offsets
- Discover bad characters
  - Educated trial and error
  - !mona cmp
- Settle on a spot to stick some shellcode
  - ESP often points to right after Saved Return Pointer overwrite – good spot

Too long; didn’t listen (2/2)

- Use control over EIP to divert execution to shellcode location
  - Overwrite Saved Return Pointer with a pointer to a "JMP ESP"
  - !mona jmp -r esp -cpb "\x00\x0a"
  - Use an INT3 breakpoint ("\xcc") to test for execution
- Generate calc-popping shellcode
  - msfvenom -p windows/exec -b '\x00\x0A' -f python --var-name shellcode_calc CMD=calc.exe EXITFUNC=thread
- Account for the decoder stub's GetPC destroying your shellcode
  - Easy mode: NOP sled ("\x90"*16)
  - Pro mode: SUB ESP, 16 ("\x83\xec\x10")
- Run your exploit, pop calc

The solution

Code defensively
Fix known bugs

Use bounded string/memory manipulation functions

e.g. sprintf() → snprintf()

Mitigate unknown bugs

ASLR

DEP/NX

Stack Canaries

Do corelanc0d3r’s tutorials

search Google for "corelan tutorial part 1"

Questions?

Thanks!

https://github.com/justinsteven/dostackbufferoverflowgood

@justinsteven