Exploitation in a hostile world

Warren Levin

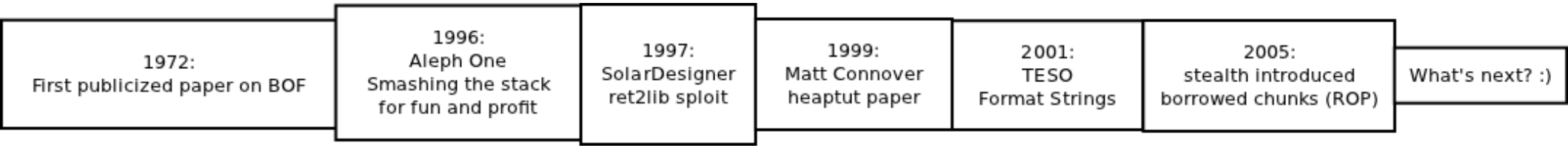
Who's this dude talking?

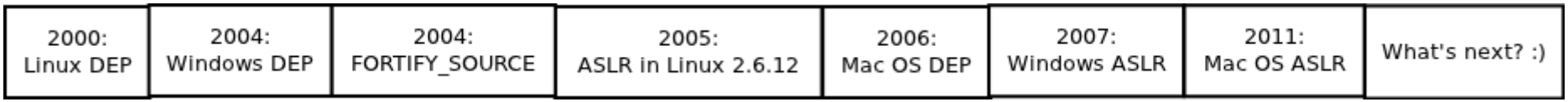
- Warren Levin or more commonly known as m_101 in the infosec community
- Student in a MSc in Forensic Computing
- Hobbyist in computer security

Agenda

- Exploitation timeline
- Exploitation in the past
  - Format strings
  - Buffer overflows
- Exploitation in the present
  - Mitigations: GS, SafeSEH, DEP/NX, ASLR
  - Bypass: GS, SafeSEH, DEP/NX, ASLR
- The future of exploitation?

Exploitation timeline



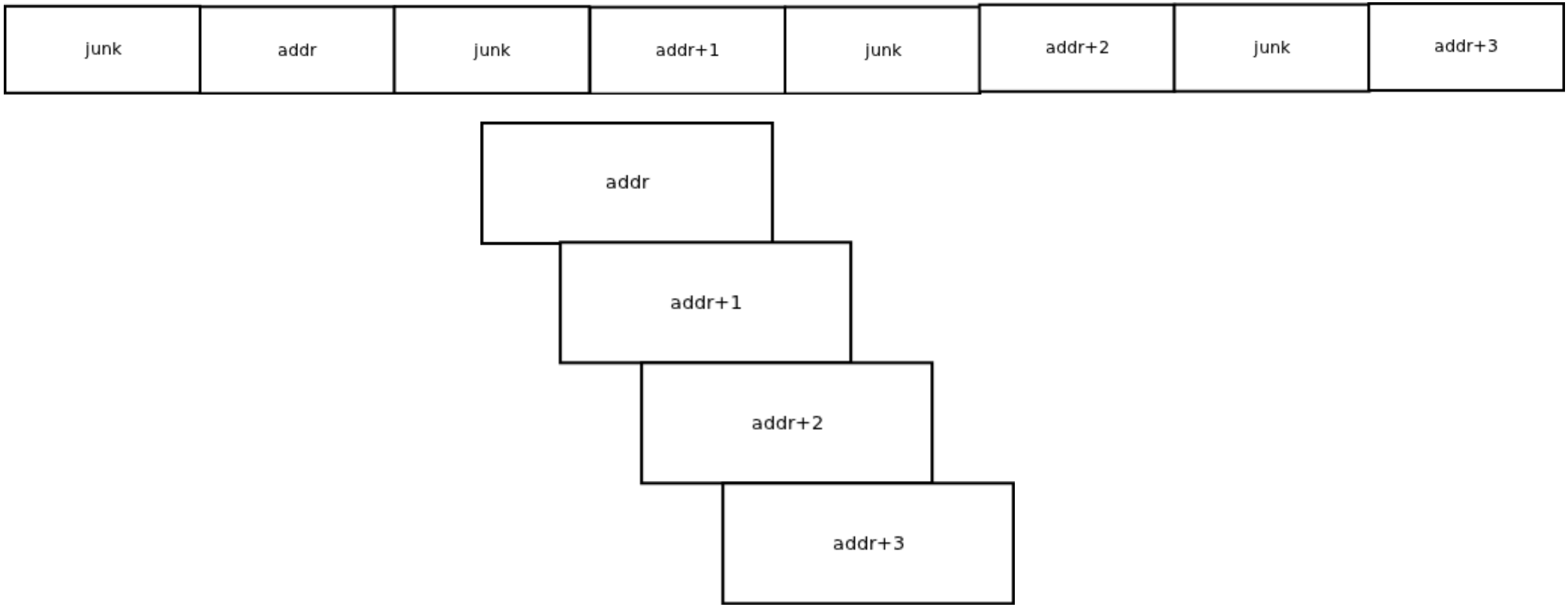


Format strings

- printf(), sprintf(), snprintf(), etc
- Thanks to bad usage:
  - printf(str); != printf("%s", str);
- Arbitrary number of arbitrary writes
- Formats:
  - %n : write number of bytes written to a variable
  - %x : read hex digit in stack
- Length specifiers:
  - h : short

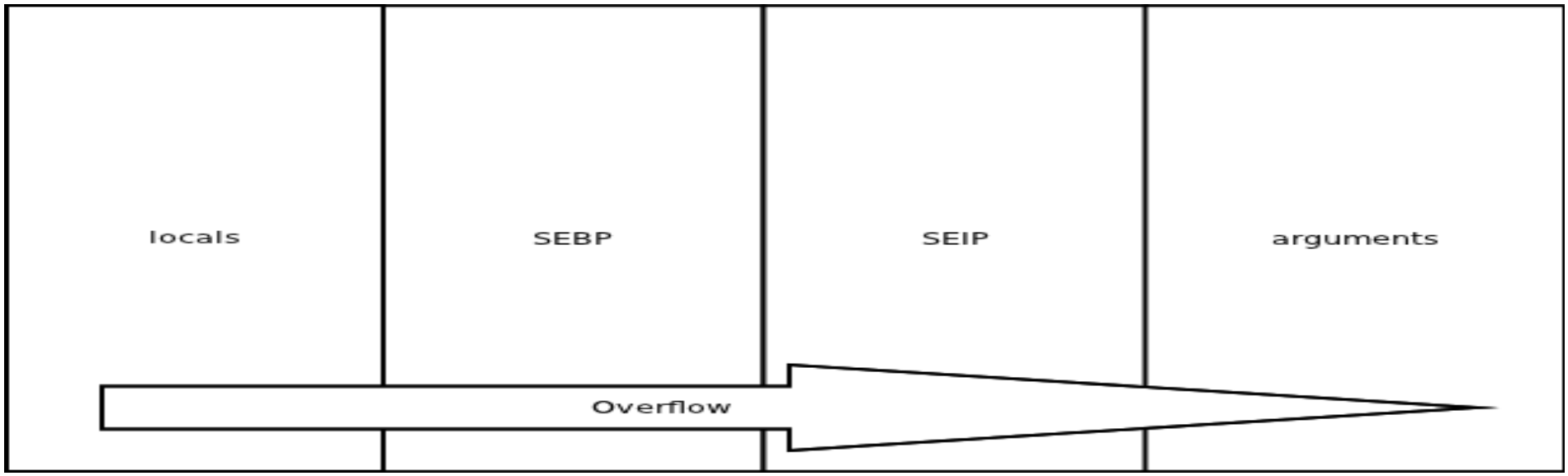
Format strings : long writes

- “0x08046889%x%n0x08046890%x %n0x08046891%x%n0x08046892%x%n”



Buffer Overflows

- Buffer overflows first publicly released by AlephOne
- They allows arbitrary code execution



The “faulty instructions”

- Here they are:

  mov esp, ebp

  pop ebp

  ret

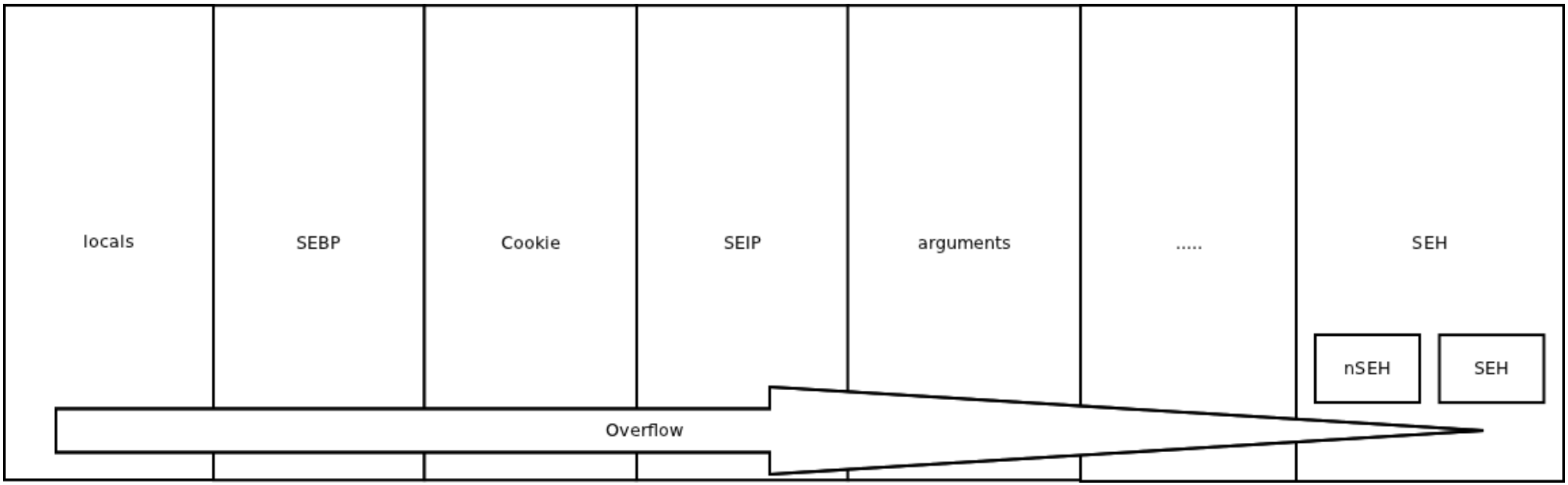
- Same as:

  leave

  ret

SEH Exploitation

- Based on Windows Exception handling (<= XP)
- More reliable than direct ret overwrite



Windows memory protections

Memory Protection Mechanisms

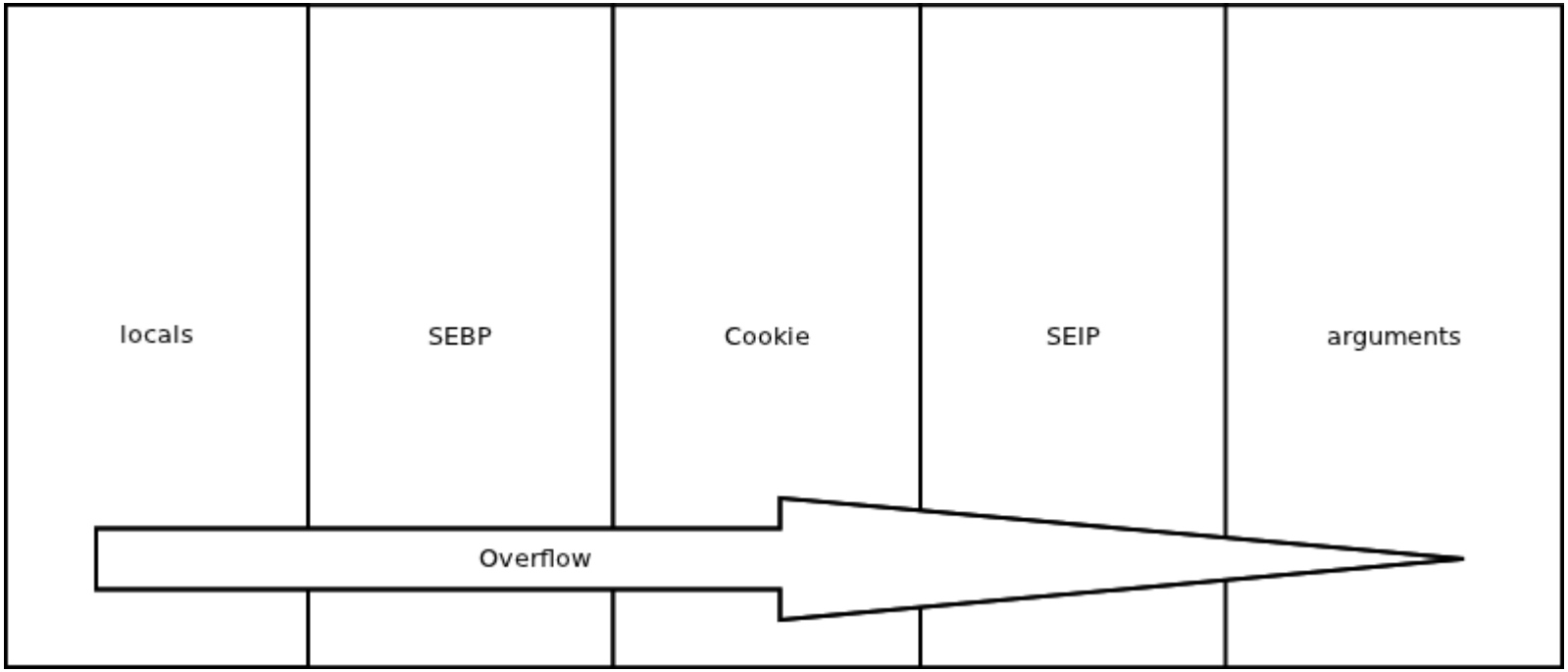
	XP SP2, SP3	2003 SP1, SP2	Vista SP0	Vista SP1	2008 SP0
GS					
stack cookies	yes	yes	yes	yes	yes
variable reordering	yes	yes	yes	yes	yes
#pragma strict_gs_check	no	no	no	?	?
SafeSEH					
SEH handler validation	yes	yes	yes	yes	yes
SEH chain validation	no	no	no	yes [1]	yes
Heap protection					
safe unlinking	yes	yes	yes	yes	yes
safe lookaside lists	no	no	yes	yes	yes
heap metadata cookies	yes	yes	yes	yes	yes
heap metadata encryption	no	no	yes	yes	yes
DEP					
NX support	yes	yes	yes	yes	yes
permanent DEP	no	no	no	yes	yes
OptOut mode by default	no	yes	no	no	yes
ASLR					
PEB, TEB	yes	yes	yes	yes	yes
heap	no	no	yes	yes	yes
stack	no	no	yes	yes	yes
images	no	no	yes	yes	yes

(Sotirov)

GS/StackGuard

- Place a cookie/canary on stack before ret address
  - Before overwriting ret, we also overwrite cookie
- Types: random, random xor, terminator, null
- Cookie is checked before function returns
  - Unmatched cookies can lead to a killed process

GS/StackGuard



GS/StackGuard Bypass

- Overwrite EIP without writing GS (XOR it!)
  - Format string
  - Using a pointer: go for EIP, cookie (.idata), vfunc, ...
- Trigger SEH handler before cookie verification
- Vuln in old VisualStudio: overwrite default handler

SafeSEH

- Verification if handler addr is included in protected binary
- Forbid ret2code (p/p/r) in SafeSEH module through SEH exploit

SafeSEH Bypass

- Use non SafeSEH module
- Direct RET overwrite (no SEH sploit ...)

ASLR

- ASLR = Address Space Layout Randomization
- Randomization of address space layouts
  - Executable mapping: through PIC
  - Stack, Heap, etc
- In Linux since kernel 2.6.12
- In Windows since Vista
- In Mac OS since … never?

ASLR bypass

- Information leaks
- Partial overwrite
- Layout of stack is the same (offsets are static)
- ret2code (no PIE)
- Pointers laying in the stack
- Heap spraying (mostly using JavaScript)
- Brute forcing (ugly!)

DEP/NX

- Set pages as non executable
  - Through software implementation (PAX, grsecurity)
  - Using hardware capabilities: NX, XD bits
- Multiple policies on Windows:
  - OptIn
  - OptOut
  - AlwaysOn
  - AlwaysOff
- Forbid direct execution of payload in stack

DEP/NX bypass

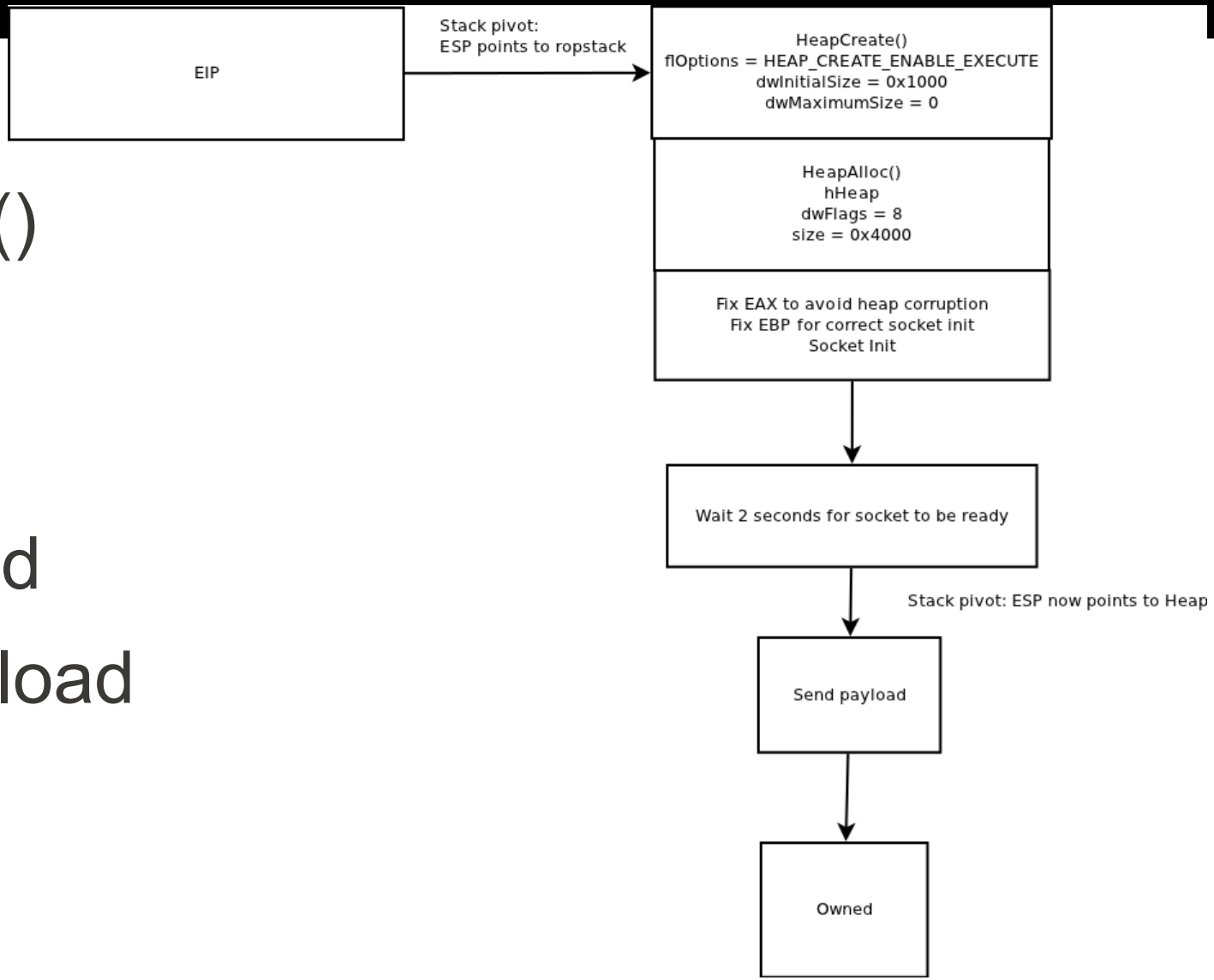
- Mostly based on ret2code techniques
- VirtualProtect()
- VirtualAlloc()
- SetProcessDEPPolicy()
- NtSetInformationProcess()
- WriteProcessMemory()
- etc

Partial ROP Payload

- Trigger vulnerability
- Access to executable/writable memory
  - Memory protection off (SetProcessDEPPolicy(), etc)
  - Allocate memory (VirtualAlloc(), HeapAlloc(), etc)
  - Use of existing memory
- Copy payload
- Execute payload

NovaCTF: Partial ROP

- HeapCreate()
- HeapAlloc()
- Sleep(2)
- Send payload
- Execute payload



Full ROP

- Addresses and data only in payload! No code!
- ROP is turing complete
- Stack construction using gadgets such as:
  - mov [eax], ecx
  - add [eax], ecx
  - sub [eax], ecx
  - ...
- EIP “slide” through the addresses

Virtuosa: Full ROP multistage

- Badchars: all caps
  - Forbids a lot of payloads, even alpha2
  - Bypass with encoder or ROP
- imports resolution (in ws2_32.dll)
- Fixing: string table, arguments, addresses, etc
- socket programming using ROP
- Send payload
- Execute payload

The future of exploitation?

- Hardened sandboxes
  - Use of VT-X or Pacifica HVM technologies
- Similar ACLs as Android for example
- Increased use of BOF free languages
- Kernel exploitation
- Race conditions / Timing attacks
- Web/Cloud based attacks (SQLi, etc)
- GPUs for heavy computation
- Social Engineering

The end ...

Questions?