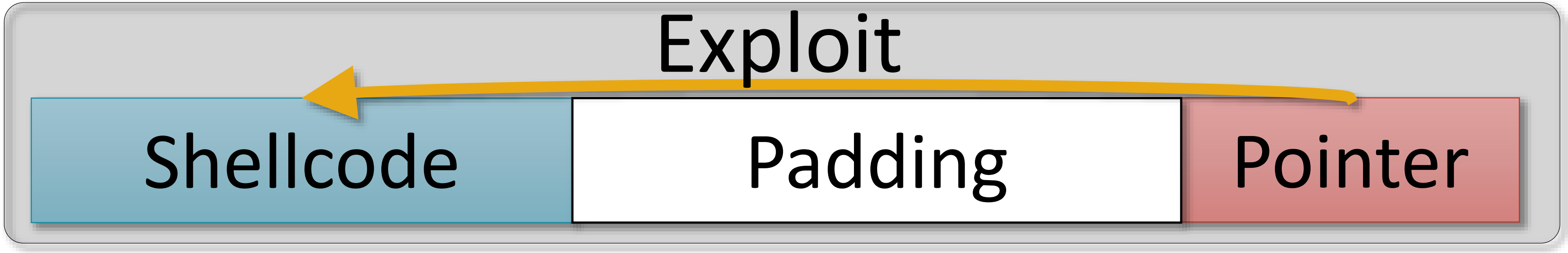
Code Reuse Attacks and How to Find Them

Edward J. Schwartz
Software Engineering Institute/CERT
Carnegie Mellon University

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Background: Traditional Control Flow Exploits

A <u>control flow exploit</u> executes code of the attacker's choosing in place of the intended application code



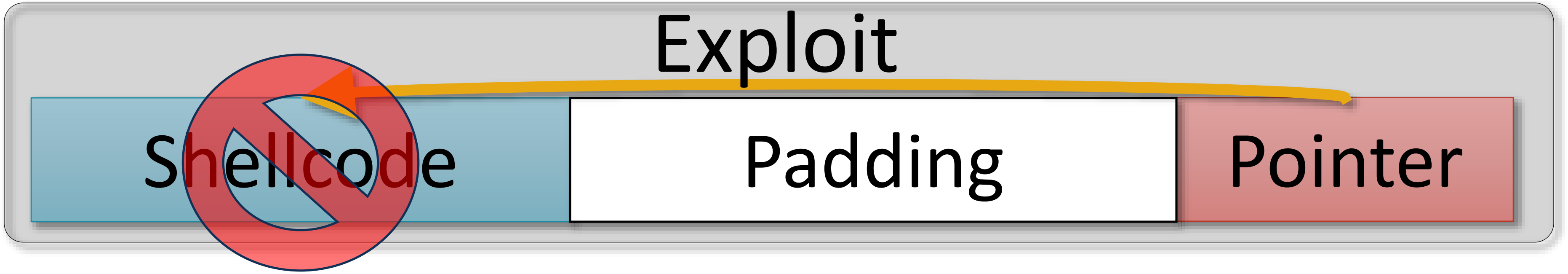
Computation

What does the exploit do?

Control Flow Vulnerability

How to control IP?

Background: Traditional Control Flow Exploits



~1995: OS defenses made the creation of an exploit difficult ☺

- DEP: Data Execution Prevention
- Prevent attacker from simply injecting new attacker code into process

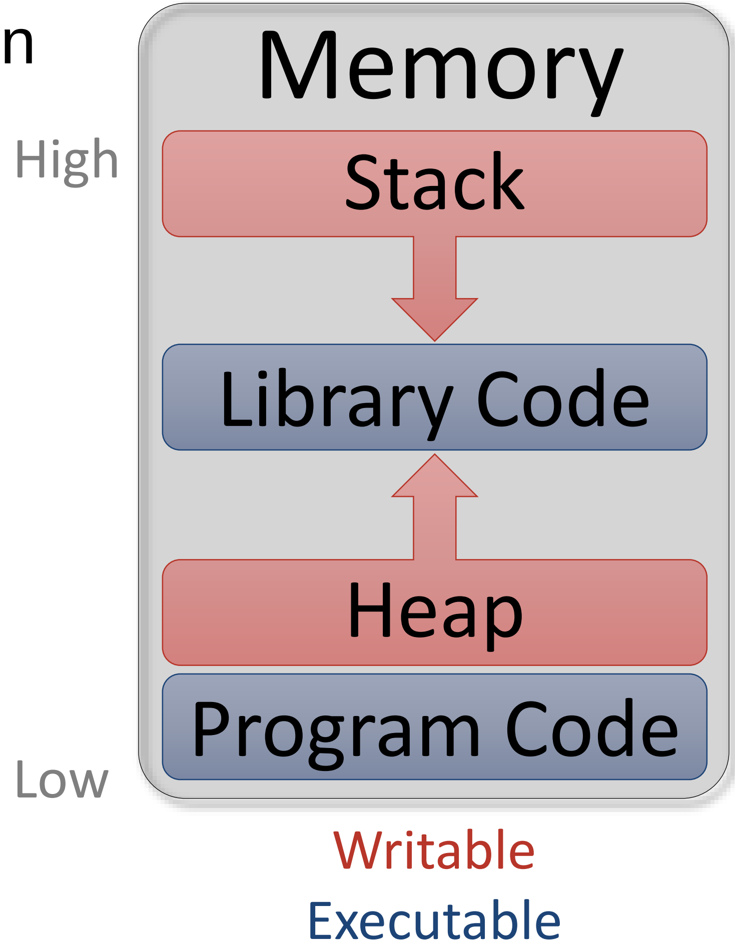
Background: Data Execution Prevention

Executable code usually known at compile time

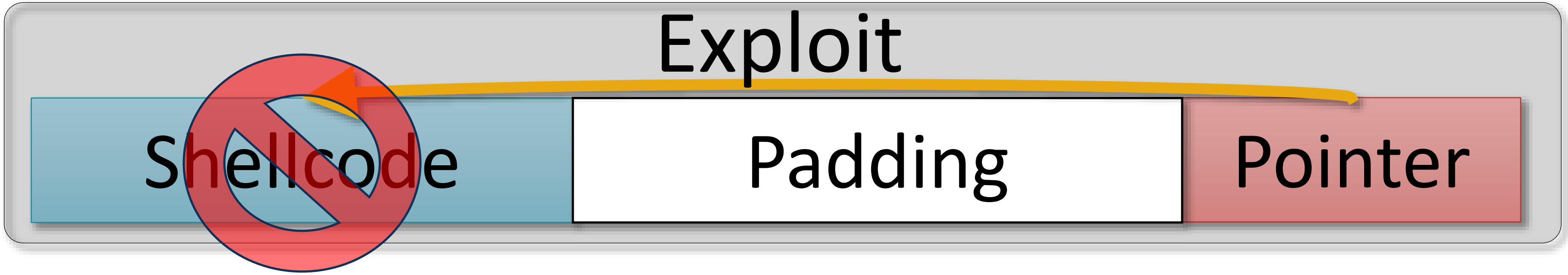
- Memory should (almost) never be writable and executable at the same time
- Code regions are executable (but not writable)
- Stack and heap are writable (but not executable)

Prevents attacker from injecting new code into the memory space

Widely available in many computing devices (even phones and tablets!)

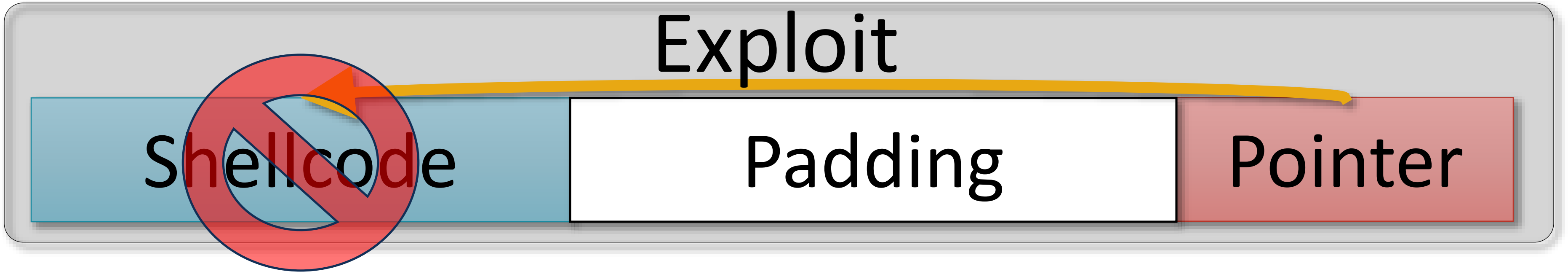


Background: Data Execution Prevention



Program crashes because shellcode is not executable

Background: Traditional Control Flow Exploits



~1995: OS defenses made the creation of an exploit difficult ☺

- DEP: Data Execution Prevention
- Prevent attacker from simply injecting new attacker code into process

~1997: Attackers figured out they can still create exploits by reusing code already in the program

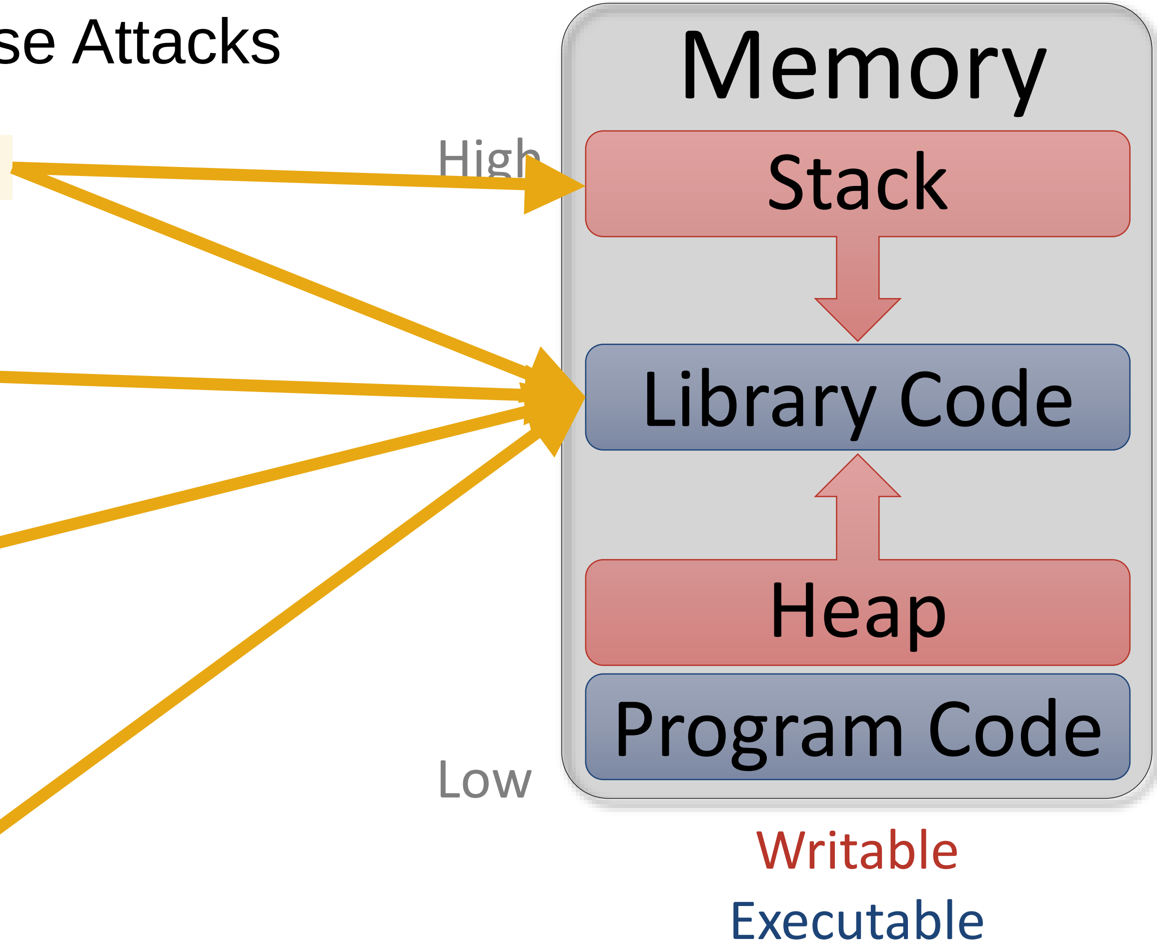
Background: Code Reuse Attacks

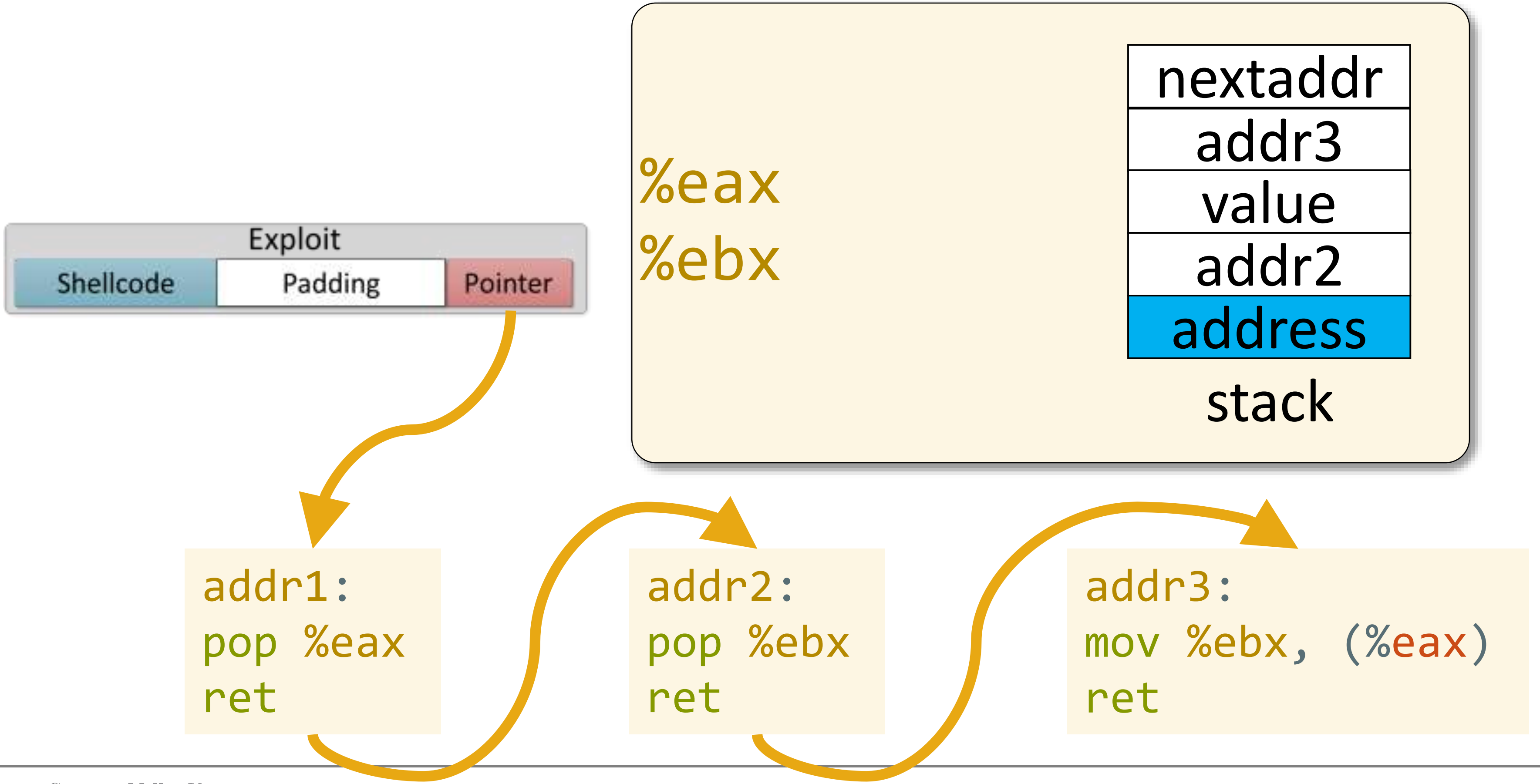
```
movl $42, 0x8048423
```

```
addr1:
pop %eax
ret
```

```
addr2:
pop %ebx
ret
```

```
addr3:
mov %ebx, (%eax)
ret
```





```
addr1:
pop %eax
ret
```

```
addr2:
pop %ebx
ret
```

```
addr3:
mov %ebx, (%eax)
ret
```

Background: Code Reuse Attacks

Return Oriented Programming (ROP)

- ROP ⊂ Code Reuse
- Find *gadgets*, code sequences ending in `ret`, that perform useful actions
  - Very similar to processor instructions

```
addr1:
pop %eax
ret
```

```
addr2:
pop %ebx
ret
```

```
addr3:
mov %ebx, (%eax)
ret
```

- `ret` allows gadgets to be chained together
- Used in virtually all practical exploits of memory safety vulnerabilities
- Turing-complete: can simulate arbitrary programs!

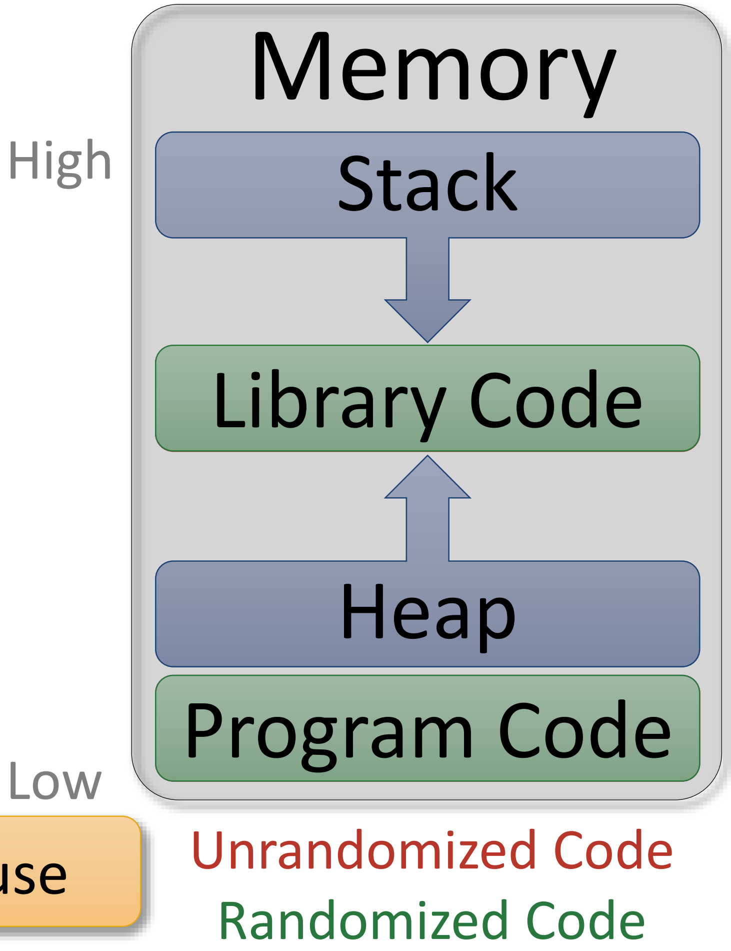
Modern Code Reuse Attacks

Address Space Layout Randomization (ASLR)

- Pre-ASLR: Code is always at the same address
- Early ASLR: Library code is randomized
- Modern ASLR: Most code is randomized

Modern defenses

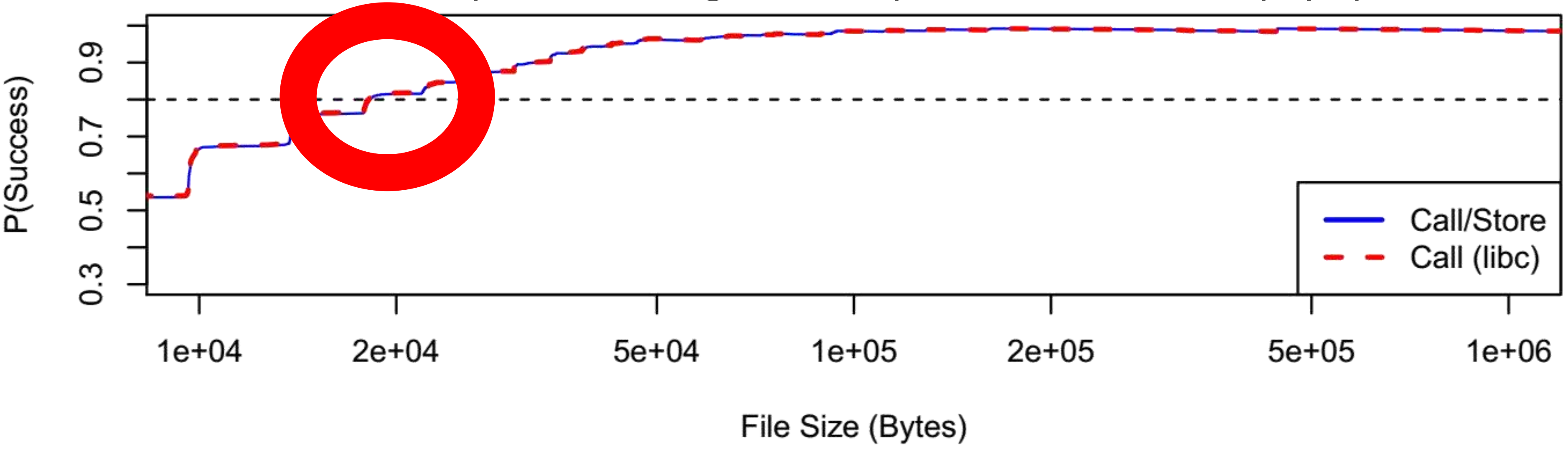
- Control Flow Integrity
- Many others
- Restrict *control flow transitions* to valid targets
  - (Usually) determined statically



Defenses ➔ Less Code Available for Reuse

How Much Code is Too Much Code?

Schwartz, et. al. Q: Exploit Hardening Made Easy, 2011 USENIX Security Symposium.



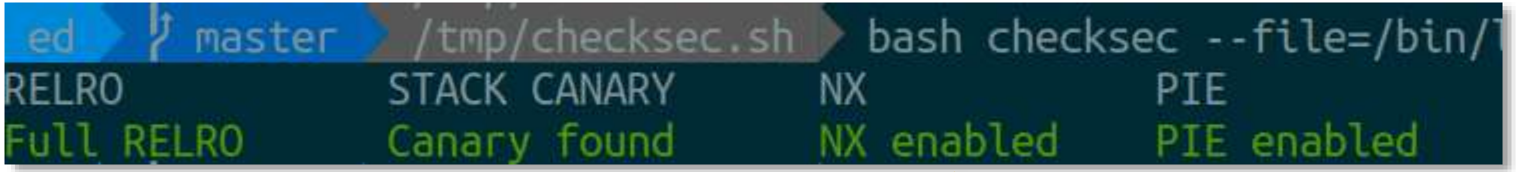
In 80% of executables larger than `/bin/true` (20 KiB), we can create a code reuse attack that calls any libc function with any argument.

What Can I Do as a Developer?

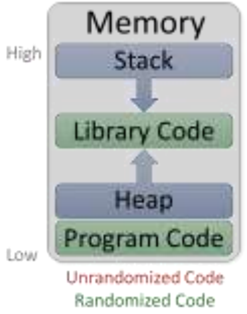
- Compile code in a way that supports DEP and ASLR
  - Linux: Compile programs as Position Independent Executables (PIEs) using `-fPIE`
  - Windows: Compile programs with `/NXCOMPAT` and `/DYNAMICBASE`
  - These are now enabled by default on modern compilers ☺

How Can I Tell If My Program Uses DEP and ASLR?

- Linux: https://github.com/slimm609/checksec.sh

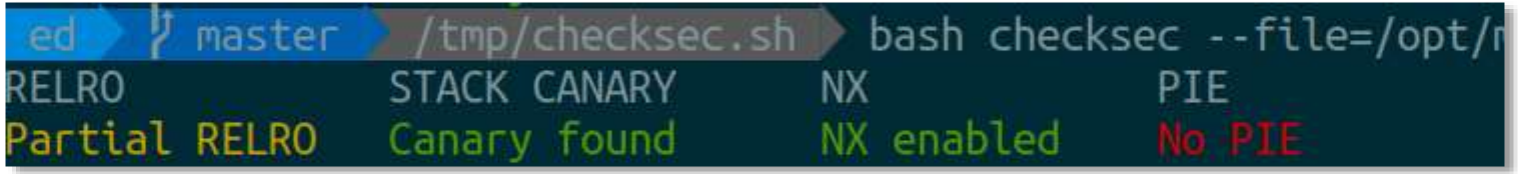


```
ed  master  /tmp/checksec.sh  bash checksec --file=/bin/l
RELRO           STACK CANARY      NX            PIE
Full RELRO      Canary found      NX enabled    PIE enabled
```

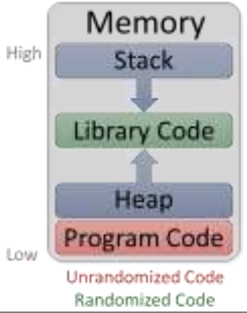


DEP

ASLR

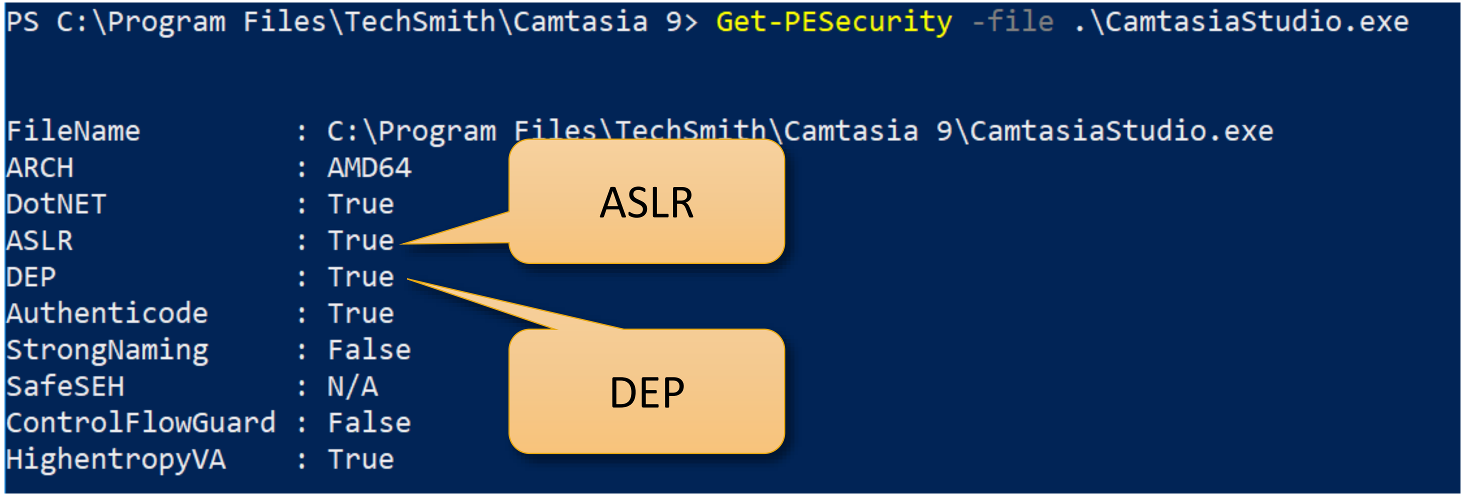


```
ed  master  /tmp/checksec.sh  bash checksec --file=/opt/r
RELRO           STACK CANARY      NX            PIE
Partial RELRO   Canary found      NX enabled    No PIE
```



How Can I Tell If My Program Uses DEP and ASLR?

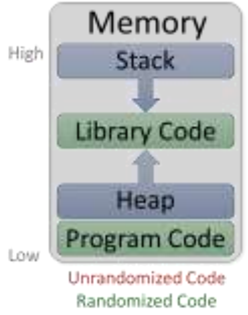
- Windows: https://github.com/NetSPI/PESecurity



```
PS C:\Program Files\TechSmith\Camtasia 9> Get-PESecurity -file .\CamtasiaStudio.exe


FileName          : C:\Program Files\TechSmith\Camtasia 9\CamtasiaStudio.exe
ARCH              : AMD64
DotNET            : True
ASLR              : True
DEP               : True
Authenticode      : True
StrongNaming      : False
SafeSEH           : N/A
ControlFlowGuard  : False
HighentropyVA     : True
```

?



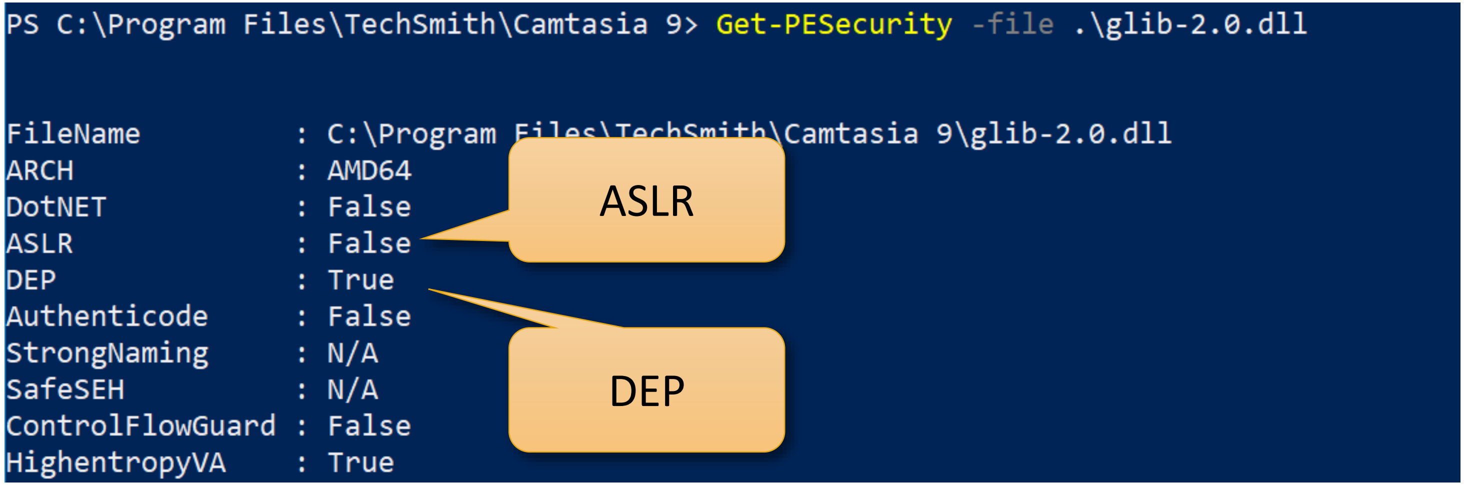
What Can I Do as a Developer?

- Compile code in a way that supports DEP and ASLR
  - Linux: Compile programs as Position Independent Executables (PIEs) using `-fPIE`
  - Windows: Compile programs with `/NXCOMPAT` and `/DYNAMICBASE`
  - These are now enabled by default on modern compilers ☺

- Ensure that 3rd party code supports DEP and ASLR
  - One bad apple spoils the bunch!

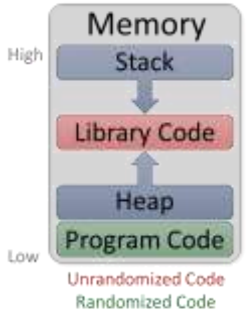
How Can I Tell If My Program Uses DEP and ASLR?

- Windows: https://github.com/NetSPI/PESecurity



```
PS C:\Program Files\TechSmith\Camtasia 9> Get-PESecurity -file .\glib-2.0.dll


FileName          : C:\Program Files\TechSmith\Camtasia 9\glib-2.0.dll
ARCH              : AMD64
DotNET            : False
ASLR              : False
DEP               : True
Authenticode      : False
StrongNaming      : N/A
SafeSEH           : N/A
ControlFlowGuard  : False
HighentropyVA     : True
```



Always remember to check libraries!

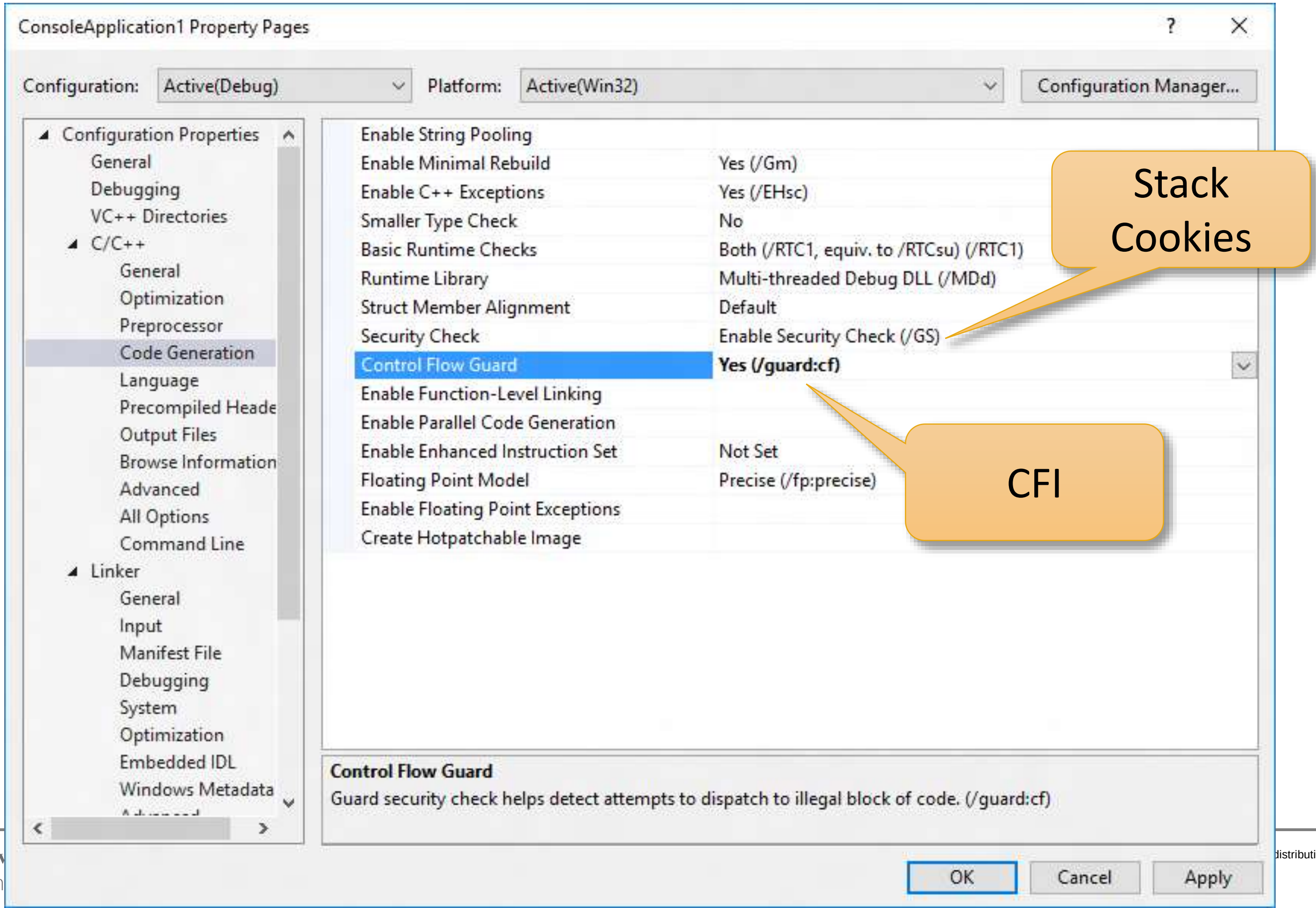
What Can I Do as a Developer?

- Compile code in a way that supports DEP and ASLR
  - Linux: Compile programs as Position Independent Executables (PIEs) using `-fPIE`
  - Windows: Compile programs with `/NXCOMPAT` and `/DYNAMICBASE`
  - These are now enabled by default on modern compilers ☺

- Ensure that 3rd party code supports DEP and ASLR
  - One bad apple spoils the bunch!

- Compile your code with extra defenses
  - Address Sanitizer (Linux): `clang/gcc –fsanitize=address`
  - Control Flow Integrity (Linux): `clang –fsanitize=cfi`
  - Stack Cookies (Windows): `cl /gs`
  - Control Flow Integrity (Windows): `cl /guard:cf`

What Can I Do as a Developer?



ConsoleApplication1 Property Pages

Configuration: Active(Debug)   Platform: Active(Win32)   Configuration Manager...

- Configuration Properties
  - General
  - Debugging
  - VC++ Directories
  - C/C++
    - General
    - Optimization
    - Preprocessor
    - Code Generation
    - Language
    - Precompiled Heade
    - Output Files
    - Browse Information
    - Advanced
    - All Options
    - Command Line
  - Linker
    - General
    - Input
    - Manifest File
    - Debugging
    - System
    - Optimization
    - Embedded IDL
    - Windows Metadata

Enable String Pooling	
Enable Minimal Rebuild	Yes (/Gm)
Enable C++ Exceptions	Yes (/EHsc)
Smaller Type Check	No
Basic Runtime Checks	Both (/RTC1, equiv. to /RTCsu) (/RTC1)
Runtime Library	Multi-threaded Debug DLL (/MDd)
Struct Member Alignment	Default
Security Check	Enable Security Check (/GS)
Control Flow Guard	**Yes (/guard:cf)**
Enable Function-Level Linking	
Enable Parallel Code Generation	
Enable Enhanced Instruction Set	Not Set
Floating Point Model	Precise (/fp:precise)
Enable Floating Point Exceptions	
Create Hotpatchable Image	

Control Flow Guard
Guard security check helps detect attempts to dispatch to illegal block of code. (/guard:cf)

What Can I Do as a Developer?

- I use a language that is:
  - Compiled to byte-code (e.g., Java, python)
  - Interpreted (e.g., shell script)
  - JIT compiled to native instructions (e.g., Javascript)

Code reuse attacks are not your responsibility!*

* Attackers can use JIT compilers to JIT produce code for them to be reuse…

Take Aways

- Prior to Data Execution Prevention (DEP), attackers would specify their computation by injecting shellcode (machine code)
- Since DEP, attackers now use code reuse attacks to specify the attacker's computation using code already in the program
- 20 KiB of unprotected code is enough to be dangerous
  - Ensure that your programs (and dependencies) are compiled for DEP & ASLR
- Bonus: Employ other runtime protections such as Control Flow Integrity (CFI)

Questions?

Contact

Edward J. Schwartz

Research Scientist

CMU/SEI/CERT/Threat Analysis

eschwartz@cert.org