



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

Copyright 2019 Carnegie Mellon University.

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

DM19-0948

Background: Traditional Control Flow Exploits

A control flow exploit 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

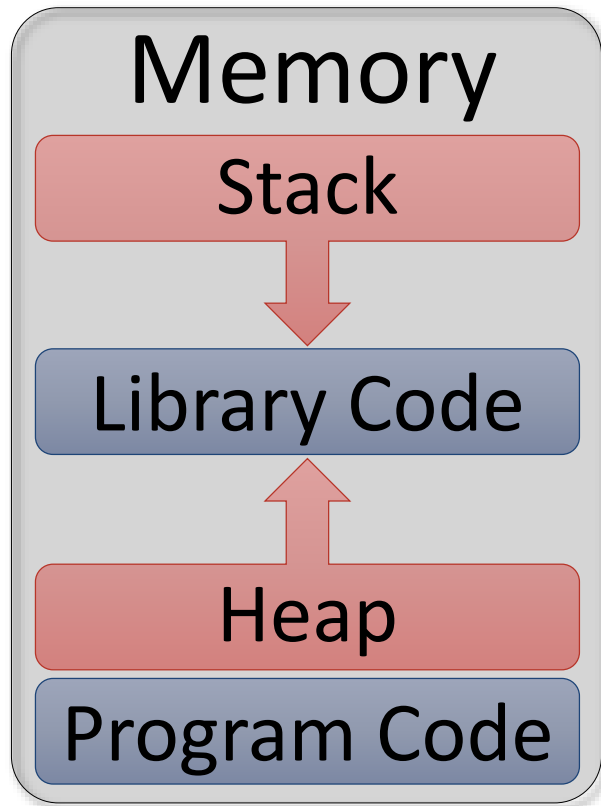
Executable code usually known at compile time

- 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!)

High



Low

Writable

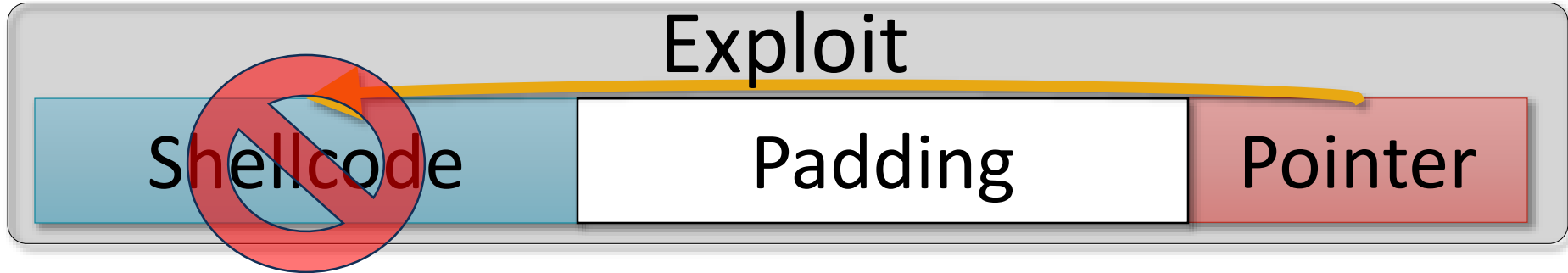
Executable

Background: Data Execution Prevention



Program crashes because
shellcode is not executable

Background: Traditional Control Flow Exploits

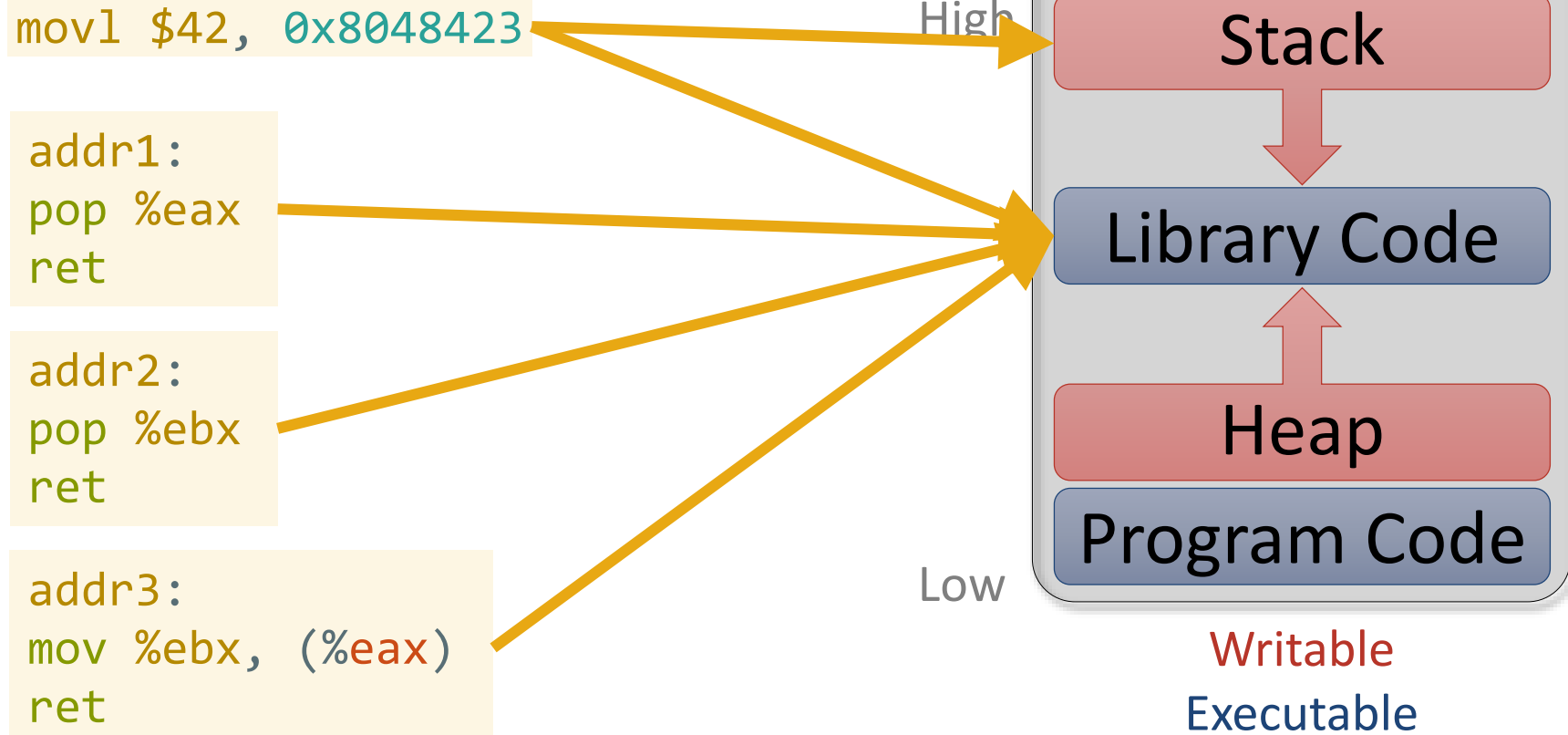


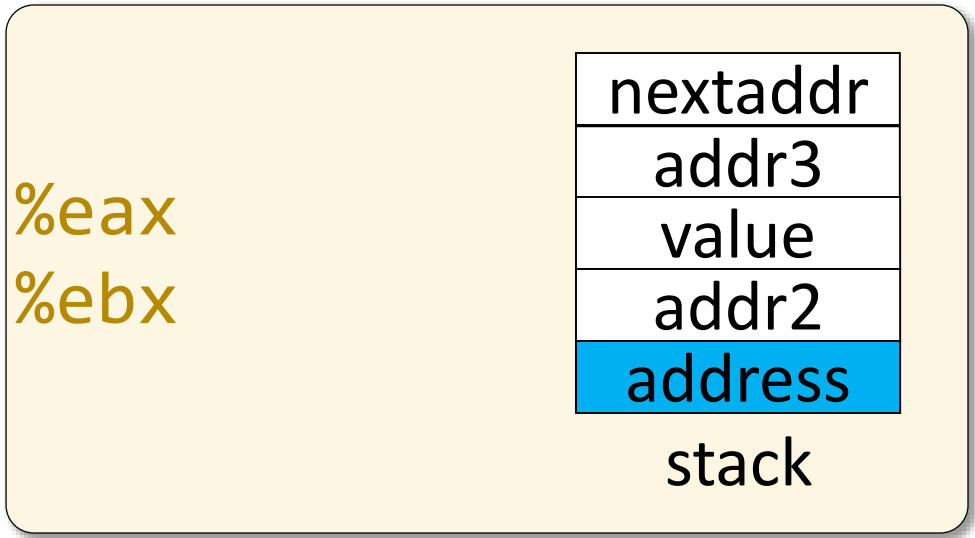
~1995: OS defenses made the creation of an exploit difficult 😊

- 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





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

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

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

Background: Code Reuse Attacks

Return Oriented Programming (ROP)

- ROP \subset 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)

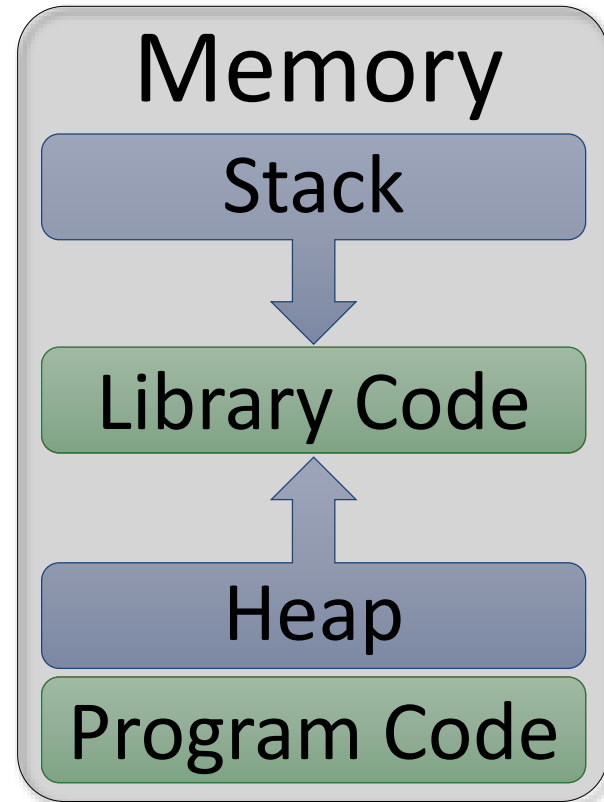
- Pre-ASLR: Code is always at the same address
- Early ASLR: Library code is randomized
- Modern ASLR: Most code is randomized

Modern defenses

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

Defenses → Less Code Available for Reuse

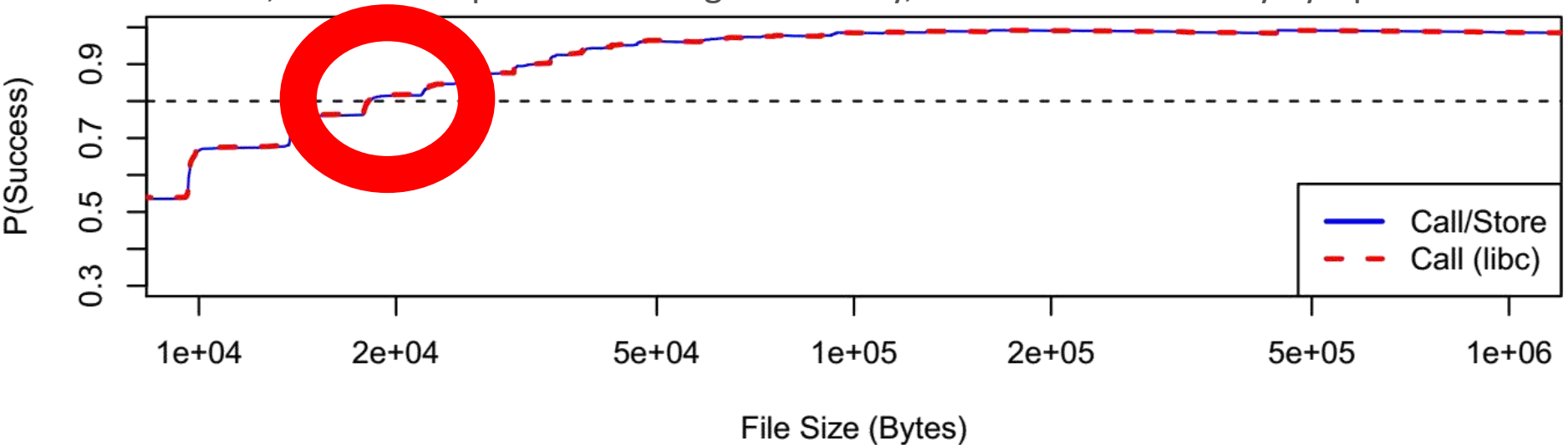
High



Unrandomized Code
Randomized Code

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?

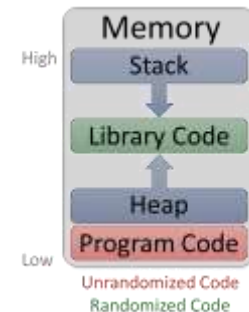
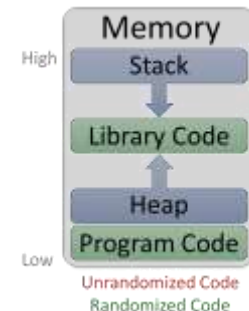
- Linux: <https://github.com/slimm609/checksec.sh>

```
ed master /tmp/checksec.sh bash checksec --file=/bin/r
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
```

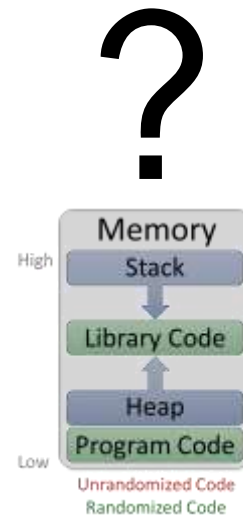




How Can I Tell If My Program Uses DEP and ASLR?

- 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?

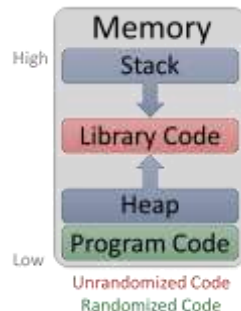
- Windows: <https://github.com/NetSPI/PESecurity>

```
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
```

ASLR

DEP



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): `c1 /gs`
 - Control Flow Integrity (Windows): `c1 /guard:cf`

What Can I Do as a Developer?

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

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

Enable String Pooling	Yes (/Gm)
Enable Minimal Rebuild	Yes (/EHsc)
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	

Stack Cookies

CFI

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

OK Cancel Apply

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