Mitigating Buffer Overflows

James D. Murray, CISSP C|EH

@jdmurray| www.TechExams.net/blogs/jdmurray

Mitigation

In information security, to reduce the potential harmful effects of a threat.

Remediation

In information security, to recover from the effects of a threat and restore normal operation.

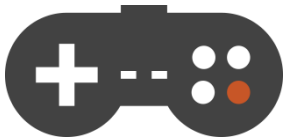
BoF Plans of Attack

BoF Attack

- Input Field
- API
- Malware
- Email Attachment
- Malicious Insider

Networked Servers & Workstations

Information Storage/Processing Systems



IoT

Mobile Devices

The BoF Attack Payoff

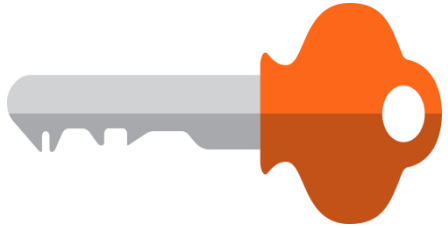
- Reconnaissance and surveillance of system and network operations
- Privilege escalation for the attacker
- Creation of unauthorized, privileged user accounts
- Command and control of the local system
- Multiple systems compromised and open to remote access
- Alteration, exfiltration, or destruction of critical information
- Temporary or permanent disablement of the system

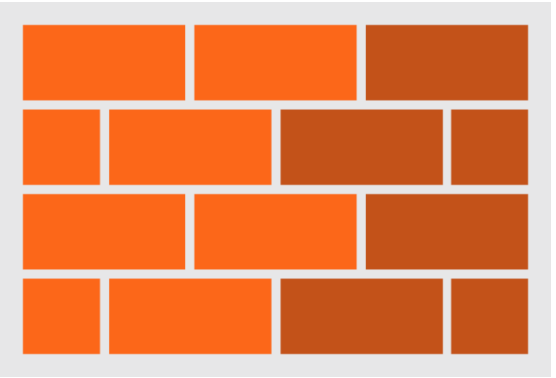
Mitigating Potential Buffer Overflows

- BoF vulnerabilities are easy for programmers to accidentally create
- Possible losses due to BoF exploitation
  - System penetration and privilege escalation
  - Data alteration, exfiltration, or destruction
  - System instability and denial of service
- Active and passive mitigation
  - Active mitigation relies on detecting buffer overflow conditions
  - Passive mitigation does not rely on detection of a threat

BoF Safeguards

- Safeguards are *proactive* security controls…
- …that attempt to prevent a threat from causing harm
- Uninstall unneeded programs
- Patch all installed programs
- Kernel and firmware anti-BoF features
  - DEP - Data Execution Prevention
  - ASLR - Address Space Layout Randomization
  - KASLR - Kernel Address Space Layout Randomization
- OS and kernel security configurations
  - EMET – Enhanced Mitigation Experience Toolkit





BoF Countermeasures

- Countermeasures are *reactive* security controls…
- …that attempt to mitigate the harm caused by an active threat
- Host-based security monitoring software
  - Firewalls, Anti-Malware software
- Network-based security monitoring software
  - Firewalls, proxies, intrusion detection/prevention
- Countermeasures cannot prevent all possible loss from a threat

Detecting Buffer Overflows

Buffers overflow inside of running processes

Network detection of buffer overflow attacks

Host detection of buffer overflow attacks

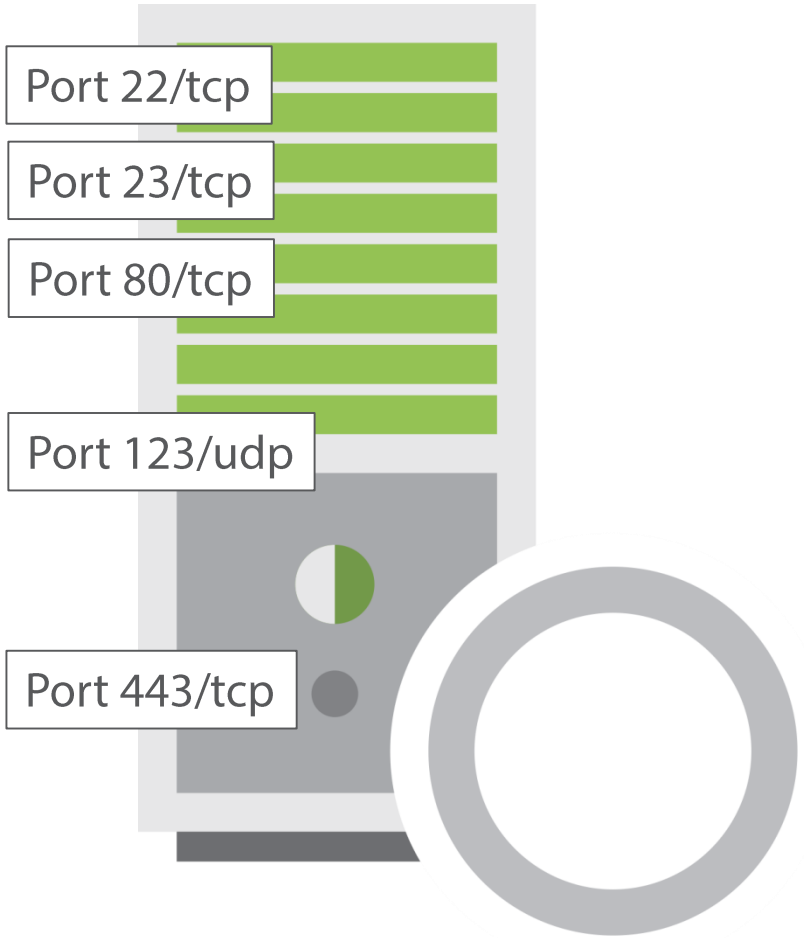
In-process detection of buffer overflow conditions

Report buffer overflow events

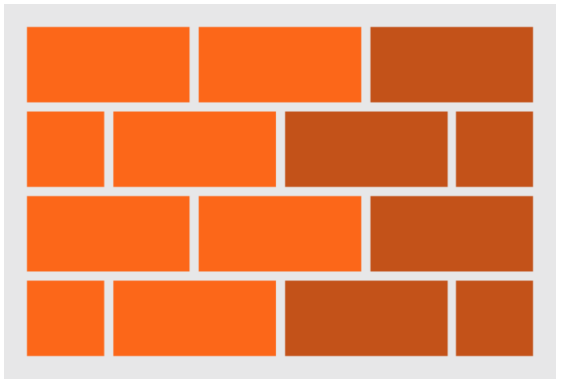
Actively mitigate buffer overflow conditions

External Buffer Overflow Detection

47E853045C948750
9090909090909090
3049855809456B98
45D0984560984560
19845695864905AF



Network Intrusion Detection Systems



NIDS

www.snort.org

```
alert tcp any any -> 192.168.1.0/24 80 (content: "|9090 9090 FFFF|/bin/sh"; msg: "Buffer Overflow attack detected!";)
```

NIDS

splunk>

www.splunk.com

Local Buffer Overflow Detection

- Host-layer defenses
- Malware exploiting buffer overflow vulnerabilities
- Anti-virus detection
  - Files
  - Email attachments
- Anti-malware solutions
  - Email and Instant Messaging scanning
  - Suspicious program behavior
  - Malware running in memory

www.clamav.net

Host-based Intrusion Detection Systems

- HIDS are a hybrid of many security systems
  - Network firewall and attack detection
  - Web browser site and content filtering
  - Detect unusual or threating system behavior
  - File system change monitoring
  - Peripheral device monitoring
  - Application monitoring

ossec.github.io

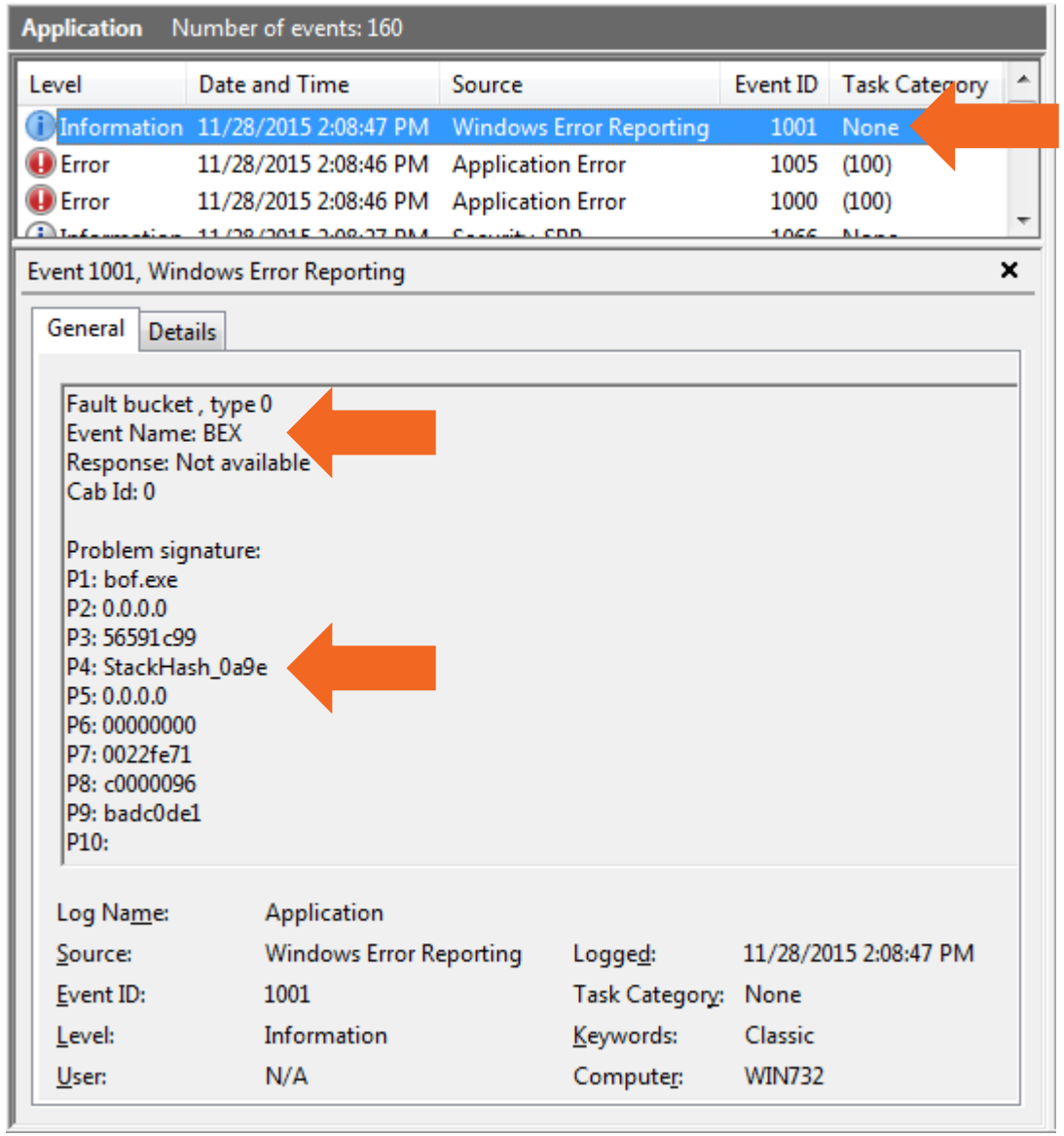
System Event Logs

- Do you know where your system's event are logged?
- Do you occasionally review your event logs to see what they contain?
- Find your logs stored in…
  - files in a subdirectory on local storage
  - a local or remote database
  - an event log collection server
  - a Security Information Event Management (SIEM) server
- Backup your log files and secure them against alteration
- Don't rely on the unreliable UDP syslog protocol

Buffer Overflows in Event Logs

- Programs that detect buffer overflows must alert that the event happened
- Microsoft Enhanced Mitigation Experience Toolkit (EMET)
  - https://www.microsoft.com/en-us/download/details.aspx?id=50766
  - It's free! Get it! Use it!
- Keywords indicating BoF conditions are not always "overflow" and "overrun"
- Look for abnormal termination events too
- User reports of on-screen error messages are important too

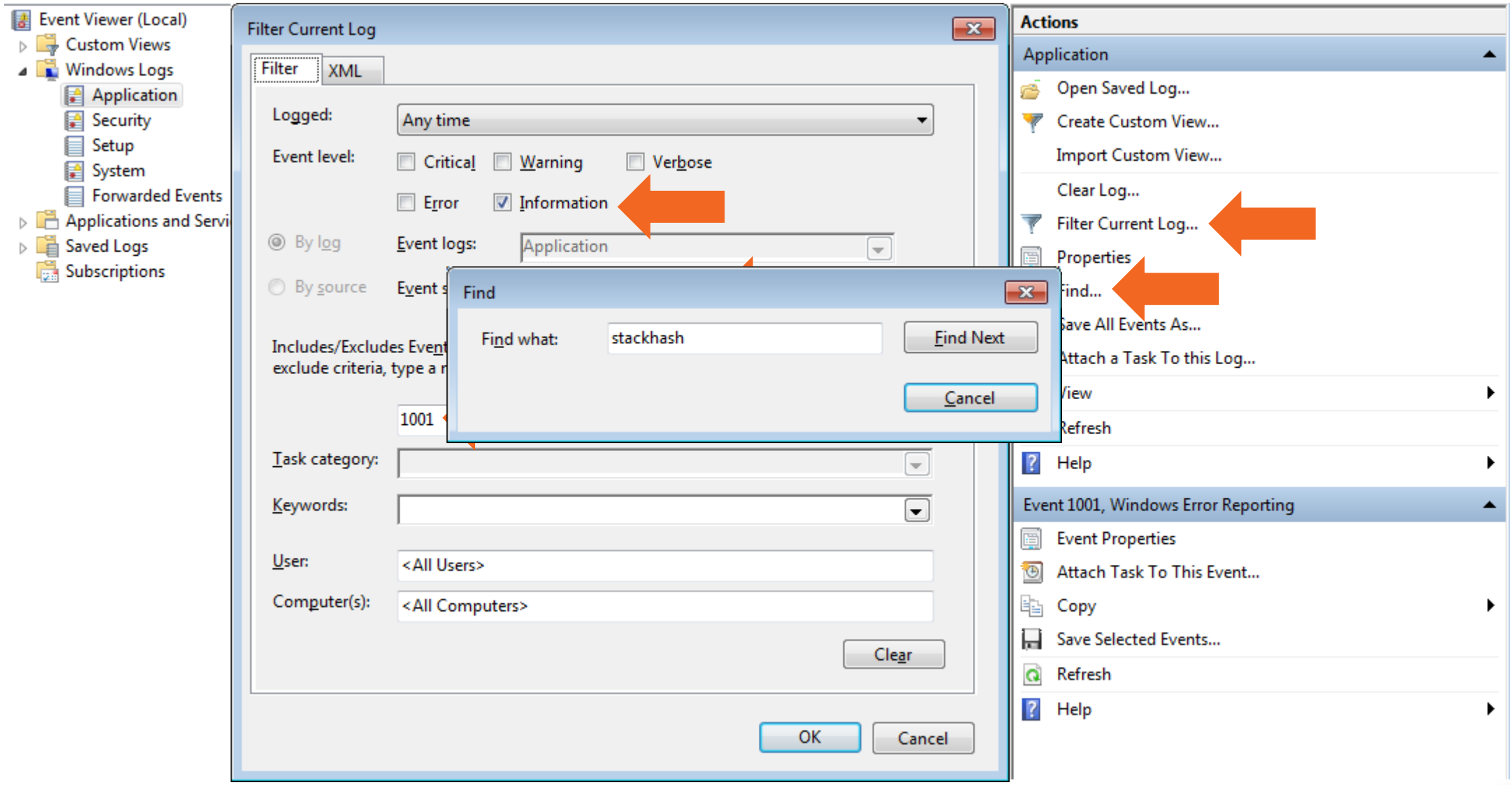
Windows Event Messages



Log Name: Application
Source: Windows Error Reporting
Date: 11/28/2015 2:08:47 PM
Event ID: 1001
Task Category: None
Level: Information
Keywords: Classic
User: N/A
Computer: WIN732
Description: Fault bucket , type 0
Event Name: BEX
Response: Not available
Cab Id: 0

Problem signature:
P1: bof.exe
P2: 0.0.0.0
P3: 56591c99
P4: StackHash_0a9e
P5: 0.0.0.0
P6: 00000000
P7: 0022fe71
P8: c0000096
P9: badc0de1
P10:

Windows Event Viewer



In-process Buffer Overflow Detection

Detect

Mitigate

Report

Buffer overflows cause data coruption

Compare stack or heap data to known good copy

Write your own overflow detection mechanism…

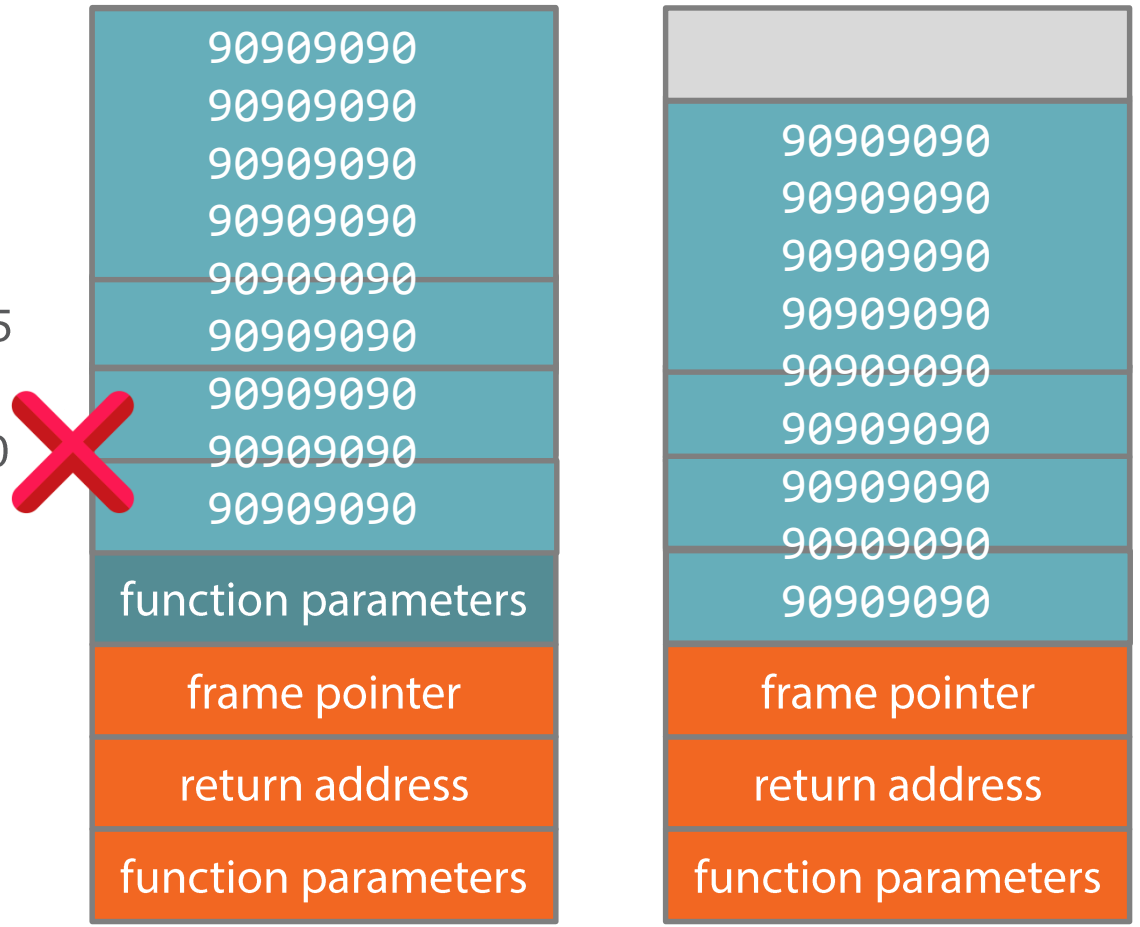
…or use the tools that the compiler gives you

Try to report the overflow event!

Of Cookies and Canaries

Security cookie = 0x2E7A2155

Security cookie = 0x90909090



- Use exception handler to report the overflow condition
- Log the event
- Mitigate the event
- Stop and restart process

Enabling Compiler Security Extensions

GCC StackGuard

-fstack-protector

-fstack-protector-all

-fstack-protector-strong

-fno-stack-protector

ProPolice

Stack Shield

No cookies or canaries

Validates stack frame return address

stackshield

shieldgcc

shieldg++

Microsoft Visual Studio

Buffer Security Check

Detects buffer overruns in the return address

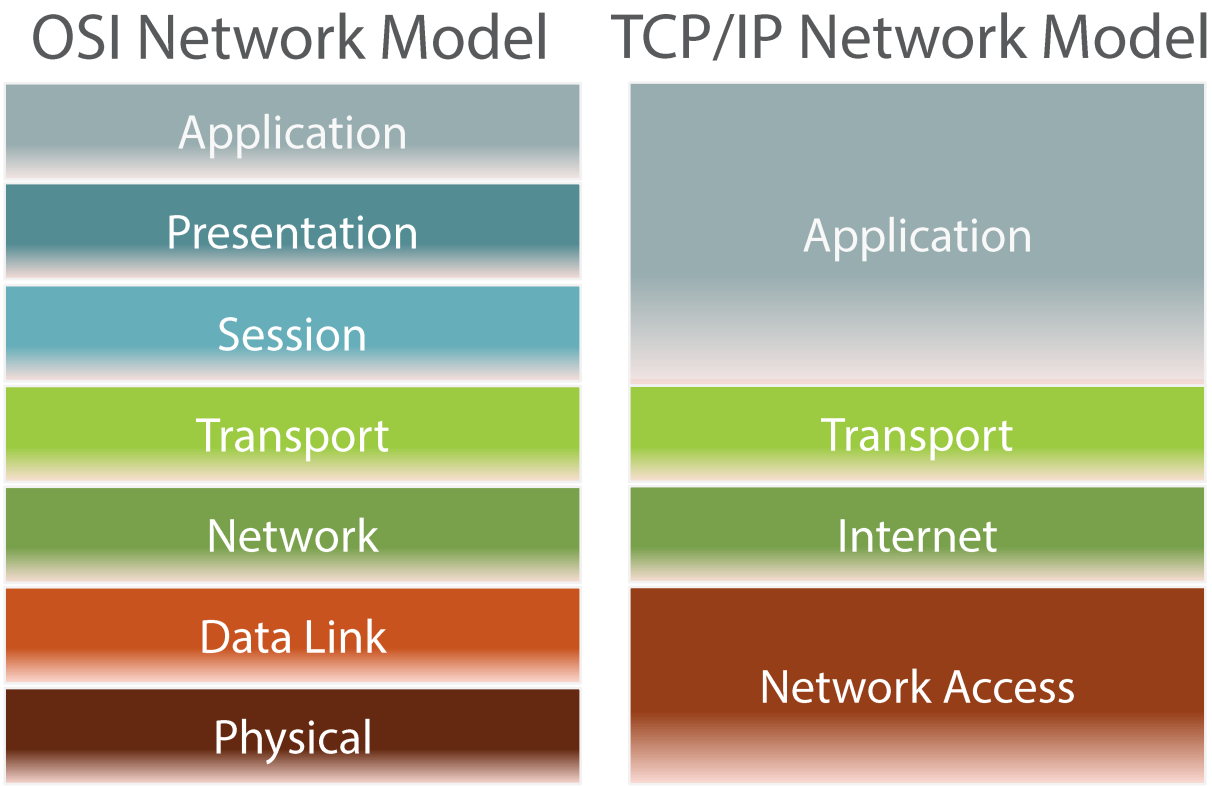
Protects against overflows of function parameters

/GS /GS-

Preventing Buffer Overflows

- The ultimate goal of mitigation is *prevention*
- Active prevention uses countermeasures to detect and mitigate threats
  - NIPS, HIPS, Anti-Malware
- Eliminate the attack payload
  - Clear memory, clean, quarantine, or delete files
- Eliminate the attack vector
  - Block network communications, suspend process execution
- Passive prevention uses safeguards to deter and mitigate threats
- Existing safeguards prevent buffer overflow attacks from succeeding
  - Firewalls, proxy servers, system configuration, user education

External Buffer Overflow Prevention



- Intrusion Prevention Systems
- Application Firewalls
- Proxies (inbound)
- Reverse Proxies (outbound)

Network Intrusion Prevention Systems

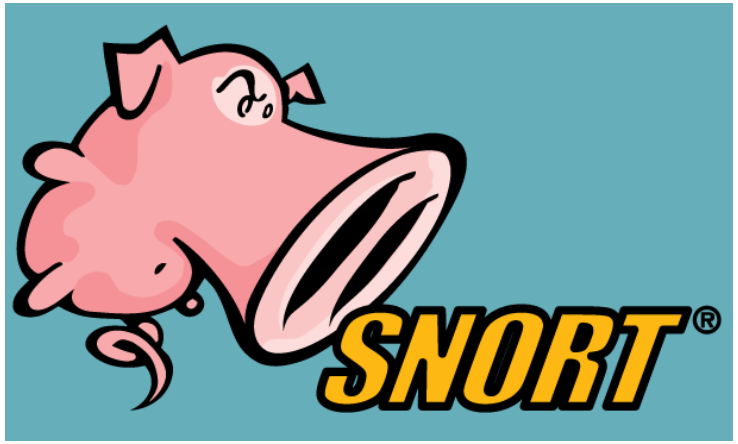
NIDS

Passive

Detect, log, and alert only

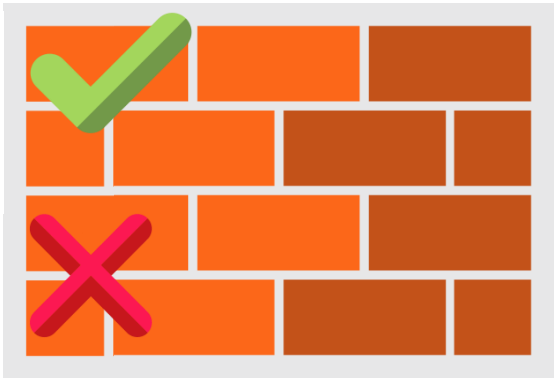
NIPS

Active, rule-based network traffic control



Many NIDS/NIPS solutions contain the Snort engine

NIPS in Action



NIPS

NIPS

Proxy Servers

- Proxy servers are real-time packet and session filters
- Proxies help define network security boundaries
- Traffic moves through proxies from more trusted to less trustworthy networks
- Web proxies and Internet access gateways are proxy servers
- Traffic moves through reverse proxies from less trusted to more trusted networks
- Load balancers and application gateways are reverse proxies

Proxy Servers

- Deep packet payload inspection
  - Source/destination IP addresses
  - Network port numbers
  - Session IDs
  - Application-layer protocol information
- Decrypt HTTPS/SSL/TLS traffic
- Uses information from NIDS and Anti-virus scanning

www.squid-cache.org

Application Firewalls

- Deep packet inspection and network traffic flow analysis
- Detect common and anomalous application security attacks
- Network traffic signatures and behaviors
- Bots, crawlers, scanners, DoS, and suspicious/malicious activity
- Vulnerability scans, brute force login attack, transferring malicious files
- Web Application Firewalls (WAF)
  - CSRF, XSS, URL hacking, SQL Injection, BoF
  - Telnet, FTP, SQL, LDAP
  - HTTPS SSL/TLS, SCP, SSH

modsecurity

Open Source Web Application Firewall

www.modsecurity.org/

Local Buffer Overflow Prevention

HIPS

Host Intrusion Prevention System

Multiple tools integrated into a hybrid security solution

Hiding Buffer Overflow Attacks

- NOP sleds are not present in good programming
- Obfuscate attack data using
  - Data encoding
    - Base64 VGhpcyBpcyBhIGJ1ZmZlciBvdmVyZmxvdy
    - Unicode \u0054\u0068\u0069\u0073 \u0069\u0073
  - Data encryption
  - Network traffic manipulation
    - Very small packets
    - Irregular packet timing intervals

In-process Buffer Overflow Prevention

Find and fix code causing buffer overflow vulnerabilities

Don't create the vulnerabilities in the first place

Patching or uninstalling programs

DEP

ASLR

SEHOP

In-process Buffer Overflow Mitigation

- Buffer overflows that have happened can't be prevented (obviously), but they can be mitigated
- Throw an exception and let the programmer decide
- Terminate the process and allow it to restart and recover
- Many buffer overflow events occur because of
  - poor programming and testing practices
  - bad input data
- Most buffer overflows are not the result of a deliberate attack

Heap Overflow Mitigations

Heap Exploit	Mitigation Feature	Feature Description
• Coalesce unlink overwrite • Critical section unlink overwrite	Safe unlinking	Verifies unlink operation is performed on chunk record list
• Data corruption detection • Heap cache attacks	Heap entry metadata randomization Heap entry cookie check	Verifies heap record headers have not changed before use
• Heap data structure overwrite • LFH bucket overwrite • Heap cache attacks	Data Execution Prevention	Prevent code from executing in heap memory
	Address Space Layout Randomization	Randomize the starting address of heap memory

Preventing Buffer Overflows in Code

- Code-based BoF Vulnerability Causes
  - Unchecked buffer size and datatype
  - Unexamined/unvalidated input
  - Use of unsafe functions and APIs
  - Use of unvalidated pointers
  - Insufficient security testing
  - Insufficient security training for software designers and programmers
- Code-based BoF Mitigations
  - Range/bounds checking
  - Input filtering/validation
  - Use of safe functions and APIs only
  - Canary tokens/Security cookies
  - Up-to-date patching
  - Security code reviews and testing
  - Training in Secure coding practices

DEP - Data Execution Prevention

Memory is readable, writable, and executable

Stack and Heap are executable by default

DEP marks memory as non-executable

NX bit

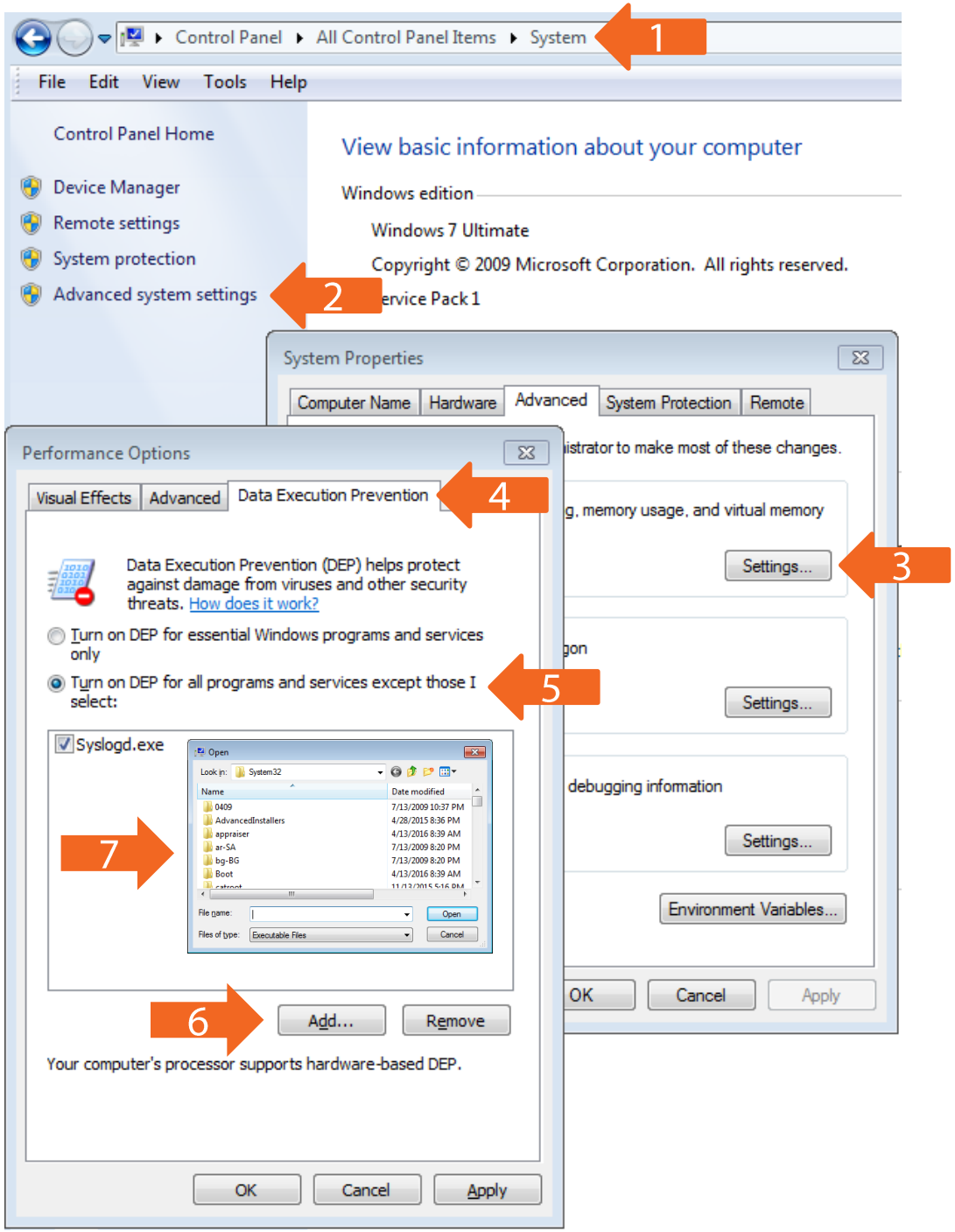
No eXecute

If a process attempts to execute code in DEP-protected memory…

…an exception is thrown; the process is terminated

Hardware and Software DEP

- Hardware memory protection feature
  - Intel CPUs - XD bit (eXecute Disabled) and EDB (Execute Disable Bit)
  - IBM ARM CPUs - XN bit (eXecute Never)
  - AMD64 CPUs - NX bit (No eXecute) and Enhanced Virus Protection
- Software memory protection feature
  - Windows XP service pack 2, Windows Server 2003 service pack 1
  - Apple in Mac OS X 10.4.4 (Tiger), 10.5 (Leopard), 10.7 (Lion)
  - Linux, UNIX - PaX (Linux), Exec Shield (Red Hat Linux), W^X (OpenBSD UNIX)
- Software-only DEP still provides some protection



1. Start the **System** applet
2. Click **Advanced system settings**
3. Click the **Settings** button in the Performance box
4. Click the **Data Execution Prevention** tab
5. Click the **Turn on DEP for all programs** radio button
6. Click the **Add** button
7. Browse to the executable file to remove from DEP protection

Setting DEP in Windows 7/8/10 Command Line

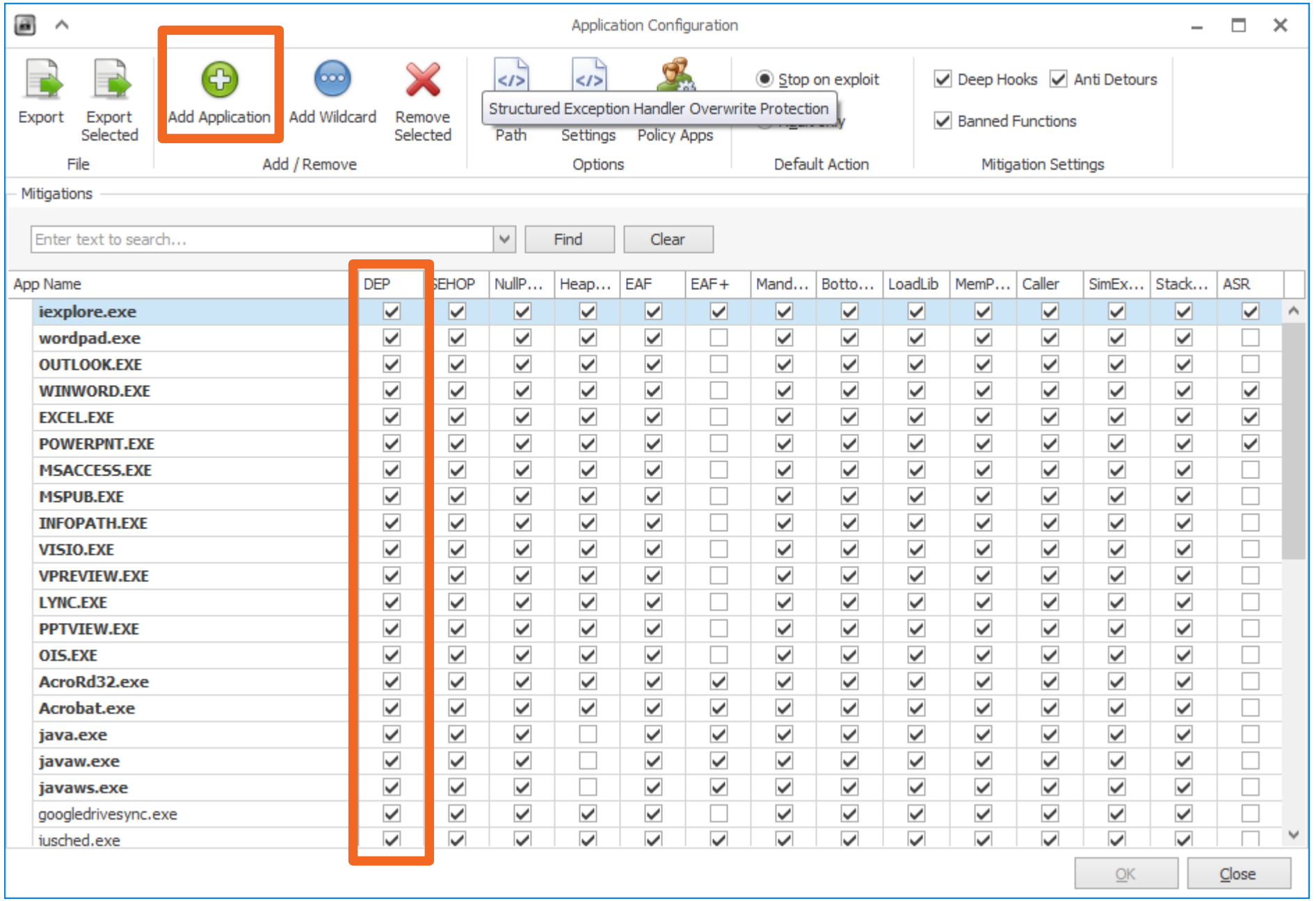
```
C:\> bcdedit.exe /set {current} nx OptIn

C:\> bcdedit.exe /set {current} nx OptOut

C:\> bcdedit.exe /set {current} nx AlwaysOn

C:\> bcdedit.exe /set {current} nx AlwaysOff

C:\> bcdedit.exe /enum
```


Application Configuration
Export
Export Selected
File
Add Application
Add Wildcard
Remove Selected
Add / Remove
Path
Settings
Policy Apps
Structured Exception Handler Overwrite Protection
Options
Stop on exploit
Default Action
Deep Hooks
Anti Detours
Banned Functions
Mitigation Settings
Mitigations
Enter text to search...
Find
Clear
App Name
DEP
SEHOP
NullP...
Heap...
EAF
EAF+
Mand...
Botto...
LoadLib
MemP...
Caller
SimEx...
Stack...
ASR
iexplore.exe
wordpad.exe
OUTLOOK.EXE
WINWORD.EXE
EXCEL.EXE
POWERPNT.EXE
MSACCESS.EXE
MSPUB.EXE
INFOPATH.EXE
VISIO.EXE
VPREVIEW.EXE
LYNC.EXE
PPTVIEW.EXE
OIS.EXE
AcroRd32.exe
Acrobat.exe
java.exe
javaw.exe
javaws.exe
googledrivesync.exe
iusched.exe
OK
Close

Checking DEP in Linux

```
# dmesg | grep NX
[    0.000000] NX (Execute Disable) protection: active
[    1.633272] NX-protecting the kernel data: 3252k

# dmesg | grep NX
[    0.000000] NX (Execute Disable) protection: disabled
by kernel command line option
```

Setting DEP in Linux

```
gksudo gedit /etc/default/grub


GRUB_DEFAULT=0
GRUB_TIMEOUT=5
GRUB_DISTRIBUTOR=`lsb_release -i -s 2> /dev/null || echo
Debian`
GRUB_CMDLINE_LINUX_DEFAULT="quiet noexec noexec32"
GRUB_CMDLINE_LINUX="initrd=/install/initrd.gz"
```

Setting DEP in Linux per Executable File

```
# execstack -s myprogam

# execstack -q myprogam
X myprogam

# execstack -c myprogam

# execstack -q myprogam
- myprogam
```

Defeating DEP

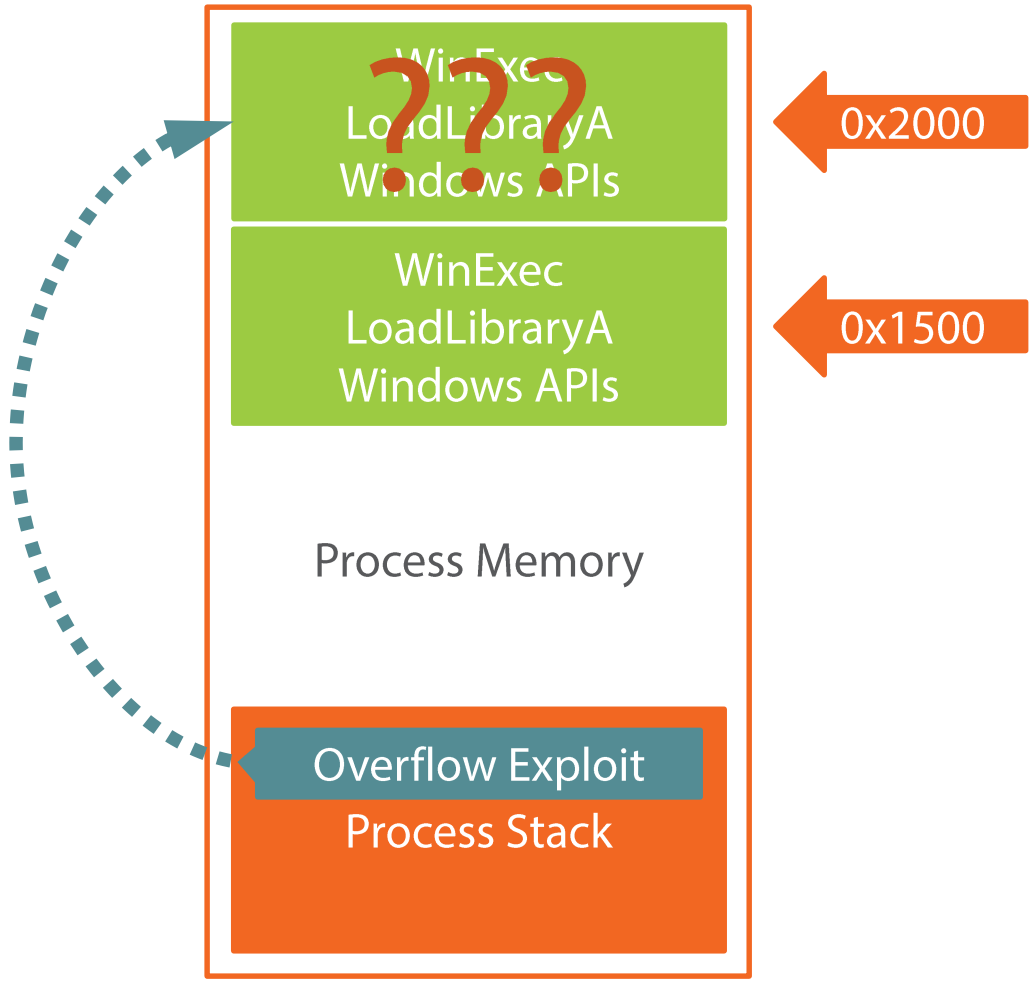
- DEP is hackable by anything with administrator (root) privileges
- Attack will not need to disable DEP for entire system, only for a few processes
- Prevent access to administrator accounts
  - Complex passwords
  - Two-factor authentication
  - Limit number of people with admin privileges
- Mitigate privilege escalation vulnerabilities
  - Remove unnecessary programs
  - Keep remaining programs updated and patched
  - Audit system and access logs

ASLR - Address Space Layout Randomization

- Process memory space is predicable
- System resources are always mapped to the same process memory addresses
- Overflow exploits know where to look in memory or can make a good guess
- Predictability is a great aid in crafting successful buffer overflow exploits
- ASLR randomizes the location of objects in process memory
- Simple overflow exploits will fail; harder for smarter exploits to succeed

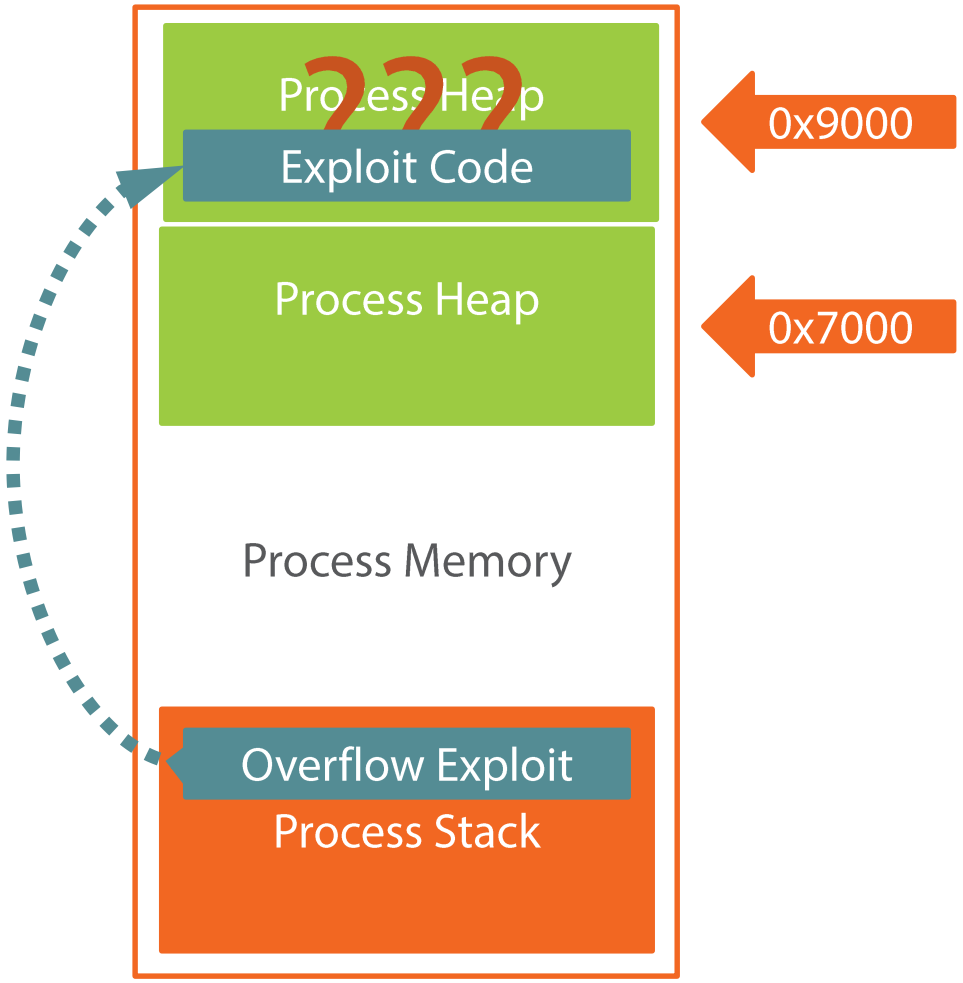
ASLR Memory Mapping Randomization

- Many overflow exploits always expect to call a function at the same memory location
- ASLR selects a random memory location to load Windows EXE and DLL images
- The overflow's execution redirection does not find the function where it expects
- The result is "undefined"



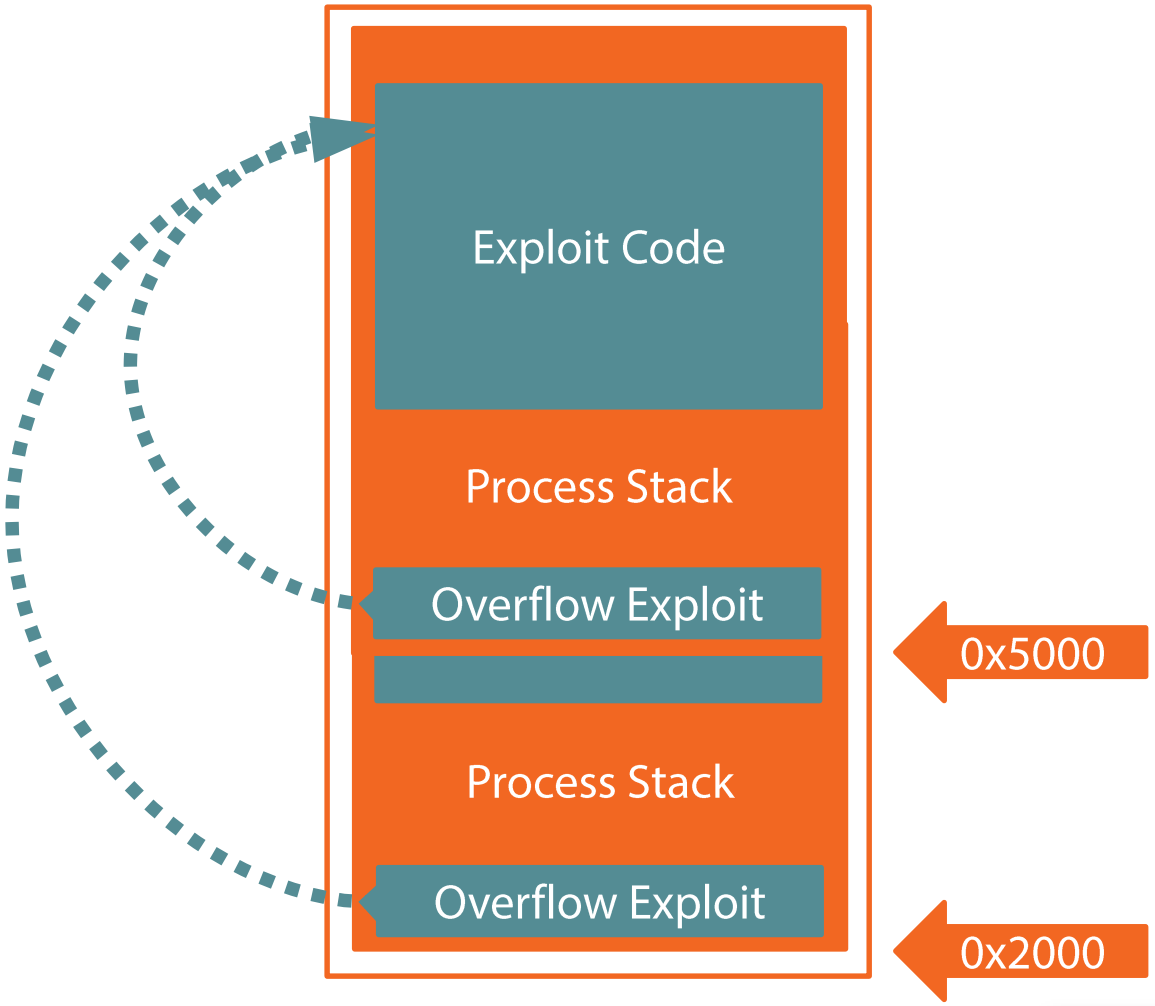
ASLR Heap Randomization

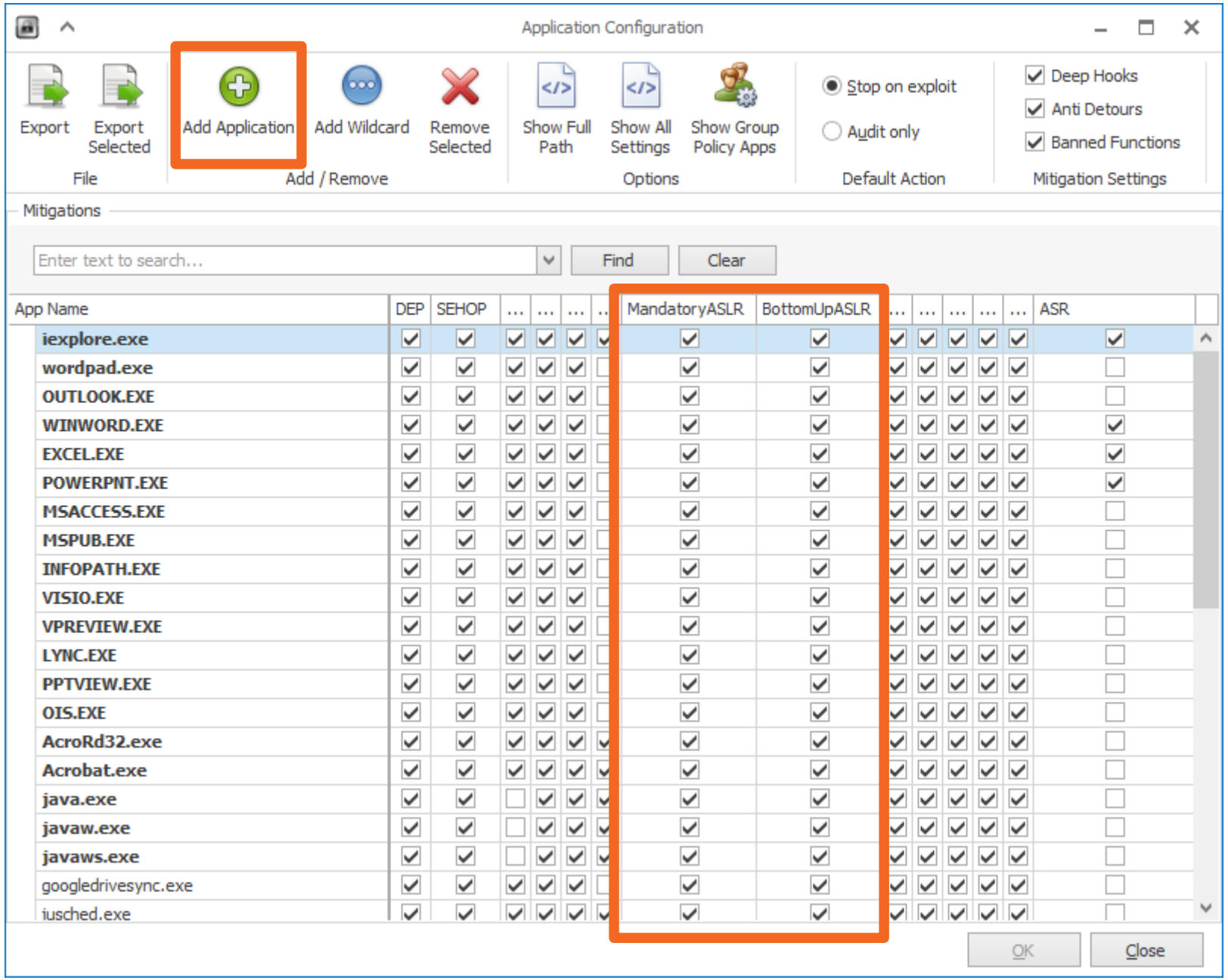
- Heap overflows load exploit code in the heap and attempt to execute it
- The base address of the heap is always the same in every process
- ASLR selects a random base memory address to locate the heap in an unpredictable location
- The overflow does not find the heap at the address expected



ASLR Stack Randomization

- Stack overflows load exploit code in the stack and attempt to execute it
- The base address of the stack is always the same in every process
- ASLR selects a random base memory address to locate the stack in an unpredictable location
- The exploit code is not found in the stack at the address expected




Application Configuration
Export
Export Selected
Add Application
Add Wildcard
Remove Selected
Show Full Path
Show All Settings
Show Group Policy Apps
Stop on exploit
Audit only
Deep Hooks
Anti Detours
Banned Functions
File
Add / Remove
Options
Default Action
Mitigation Settings
Mitigations
Enter text to search...
Find
Clear
App Name
DEP
SEHOP
MandatoryASLR
BottomUpASLR
ASR
iexplore.exe
wordpad.exe
OUTLOOK.EXE
WINWORD.EXE
EXCEL.EXE
POWERPNT.EXE
MSACCESS.EXE
MSPUB.EXE
INFOPATH.EXE
VISIO.EXE
VPREVIEW.EXE
LYNC.EXE
PPTVIEW.EXE
OIS.EXE
AcroRd32.exe
Acrobat.exe
java.exe
javaw.exe
javaws.exe
googledrivesync.exe
iusched.exe
OK
Close

Setting ASLR in Linux

```
# sysctrl –a | grep kernel.randomize_va_space
kernel.randomize_va_space = 2
# sysctl -w kernel.randomize_va_space=0
# vi /etc/sysctl.conf
kernel.randomize_va_space = 0
# sysctl –p
# setarch `uname -m` -R /bin/bash
```

Defeating ASLR

- ASLR randomization occurs only at system startup
- Identical randomized location applied to all processes
- Know the memory location for one process and you know for them all
- Randomly chosen from a fixed set of memory locations
  - 0x1234**56**78
  - Creating 256 possible locations (0x1234**00**78 to 0x1234**FF**78)
  - Some exploit code can simply try each possible location until successful

Missing ASLR

- ASLR is not always supported or enabled by every system
  - Linux (2001)
  - OpenBSD UNIX (2003)
  - Windows Vista (January 2007)
  - Apple OS X 10.5 (Leopard, October 2007)
  - Apple iOS 4.3 (March 2011)
  - Android 4.0.3 (Ice Cream Sandwich, December 2011)
- Upgrade or replace OS or ASLR-incompatible software
- Enable ASLR per-process (using Microsoft EMET)

Kernel Address Space Layout Randomization

- KASLR
- Randomizes were objects are placed in kernel memory
- First added to Linux kernel in 2005 (version 2.6.12)
- Full KASLR support added to Linux kernel in 2014 (version 3.14)
- Windows Vista (January 2007)
- Object locations are randomized at boot time
- Memory locations discovered by brute force or memory leaks
- KASLR can be disabled by default

SEHOP

Structured Exception Handling (SEH)

SEH is susceptible to buffer overflows causing arbitrary code execution

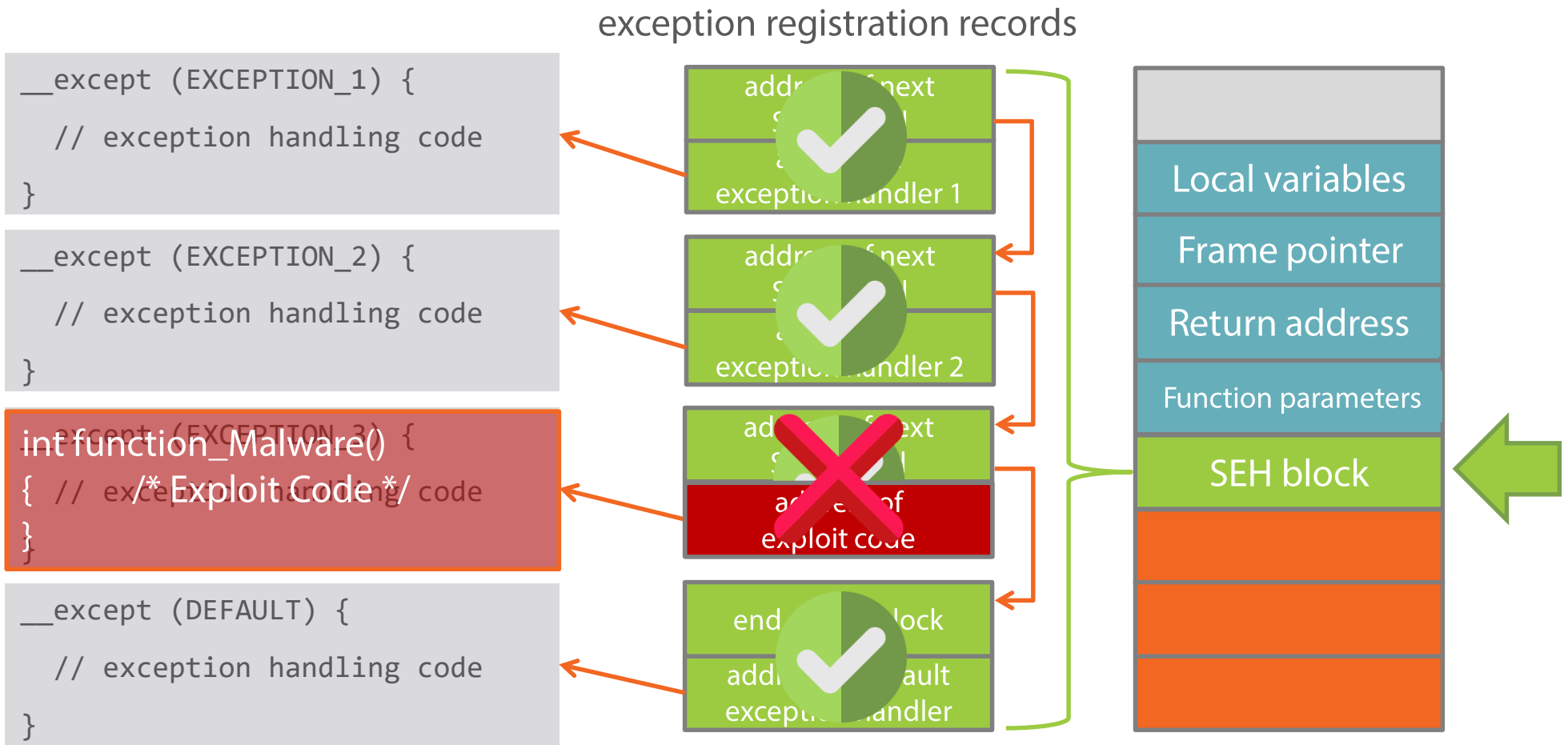
Structured Exception Handling Overwrite Protection

SEHOP detects illicit changes in the SEH control structure

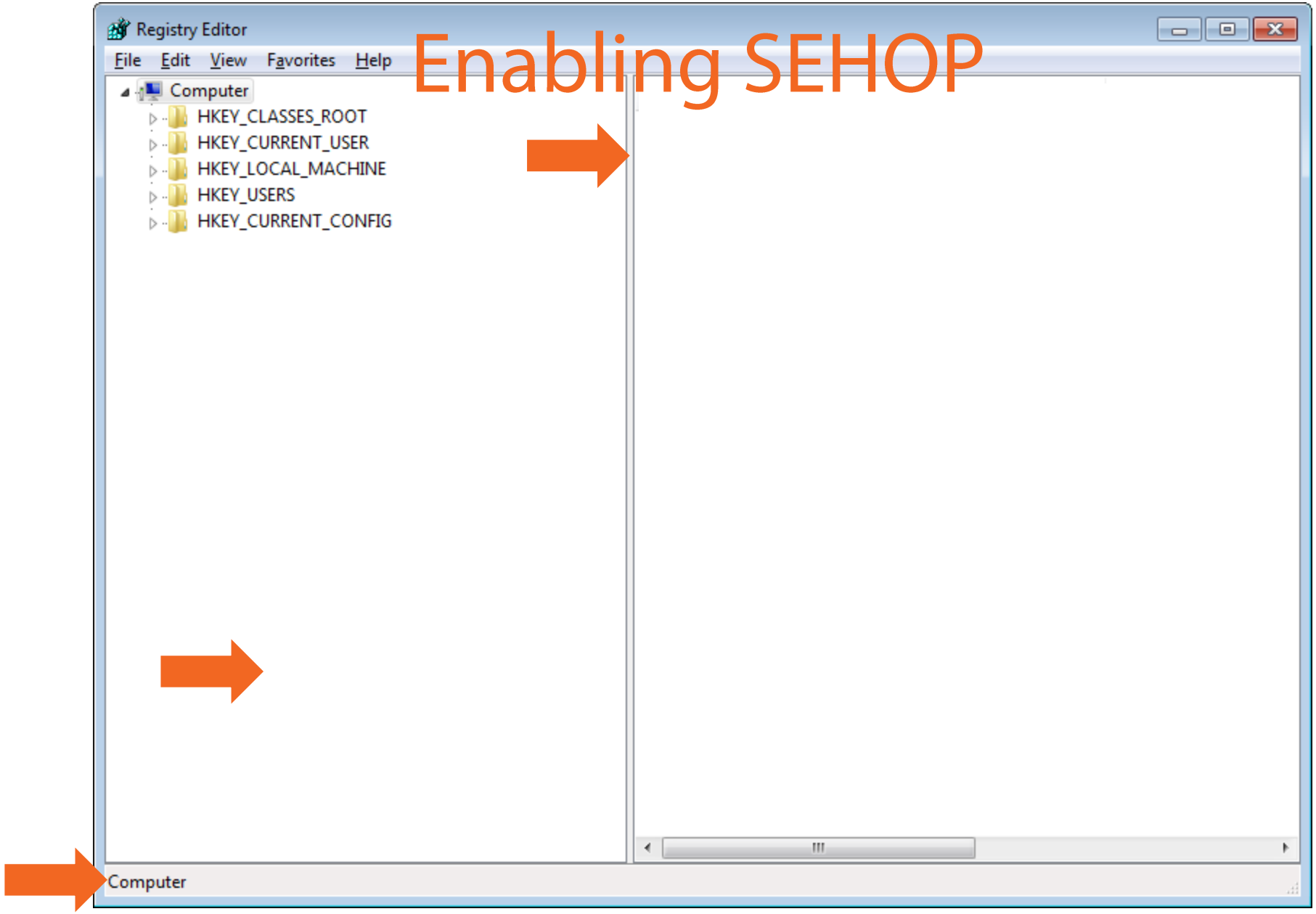
Both SEH and SEHOP are present in all Windows processes

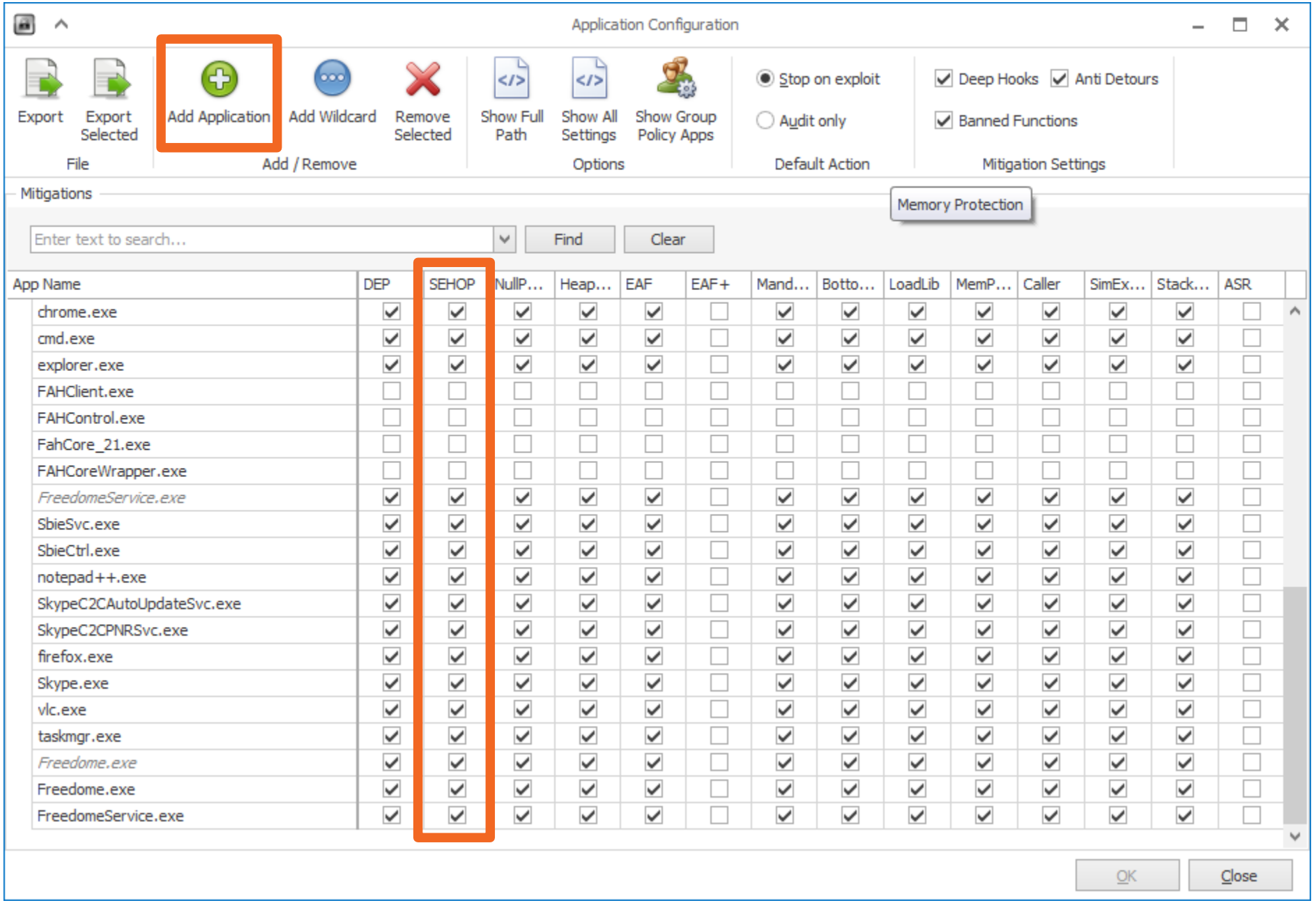
SEHOP is controlled per process using EMET

SEHOP in Action



Enabling SEHOP




Application Configuration
Export
Export Selected
File
Add Application
Add Wildcard
Remove Selected
Add / Remove
Show Full Path
Show All Settings
Show Group Policy Apps
Options
Stop on exploit
Audit only
Default Action
Deep Hooks
Anti Detours
Banned Functions
Mitigation Settings
Mitigations
Memory Protection
Enter text to search...
Find
Clear
App Name
DEP
SEHOP
NullP...
Heap...
EAF
EAF+
Mand...
Botto...
LoadLib
MemP...
Caller
SimEx...
Stack...
ASR
chrome.exe
cmd.exe
explorer.exe
FAHClient.exe
FAHControl.exe
FahCore_21.exe
FAHCoreWrapper.exe
FreedomeService.exe
SbieSvc.exe
SbieCtrl.exe
notepad++.exe
SkypeC2CAutoUpdateSvc.exe
SkypeC2CPNRSvc.exe
firefox.exe
Skype.exe
vlc.exe
taskmgr.exe
Freedome.exe
Freedome.exe
FreedomeService.exe
OK
Close

/SAFESEH

- Security feature in Visual Studio C/C++
- SAFESEH is an alternative protection mechanism to SEHOP
- Relocates the SEH block to a safe memory location outside the program stack
- SAFESEH is enabled by default
- All modules in a program must be compiled using /SAFESEH flag
- SAFESEH is disabled by the compiler flag /SAFESEH:NO
- SAFESEH is use with 32-bit executable programs only

A Brief Intro to EMET

- Microsoft security feature configuration utility for Microsoft Windows
- Download EMET 5.5 from Official Microsoft Download Center
  - https://www.microsoft.com/en-us/download/details.aspx?id=50766
- Enhanced Mitigation Experience Toolkit - EMET - TechNet Security
  - https://technet.microsoft.com/en-us/security/jj653751
- EMET Mitigations Guidelines
  - https://support.microsoft.com/en-us/kb/2909257

EMET and Windows

EMET 1.0.2 released in October 2009

Windows XP SP3, Windows Vista SP1

EMET 5.5 released in January 2016

Windows Vista SP 2, Windows Server 2008 SP2

Support for Windows 10 added in EMET 5.5

Support for Windows XP dropped in EMET 5.0

Summary

- ✓Mitigation and remediation
- ✓How and what is attacked using BoF
- ✓Safeguards and countermeasures
- ✓Network Intrusion Detection Systems
- ✓System logs and anti-Malware
- ✓Host Intrusion Detection Systems
- ✓Cookies, canaries, and compiler extensions

Summary

- ✓Network Intrusion Prevention Systems
- ✓Proxy servers and application firewalls
- ✓Host Intrusion Prevention Systems
- ✓Memory validation and randomization
- ✓Buffer overflow obfuscation
- ✓Causes and fixes for BoF vulnerabilities

Summary

- ✓Data Execution Prevention (DEP)
- ✓Address Space Layout Randomization (ASLR)
- ✓Structured Exception Handling Overflow Protection (SEHOP)
- ✓Microsoft EMET
- ✓No one security mechanism is a “silver bullet”
- ✓Layered, defense in depth is the best strategy

Prevention is ideal, but detection is a must.

— Dr. Eric Cole, SANS