EXERCISE 25

REFERENCE MONITORS REVIEW

Write your name and answer the following on a piece of paper

Give an example of a safety property that a reference monitor might enforce. How would an inline reference monitor work to enforce that safety property?

Leaking ~~PII~~ confidental data

- tracking the flow of ~~PII~~ data from sources by adding tracking instructions to the binary/IR/source code
- Add a halt / sanitizer before a sink

EXERCISE 25 SOLUTION

REFERENCE MONITORS REVIEW

Write your name and answer the following on a piece of paper

Give an example of a safety property that a reference monitor might enforce. How would an inline reference monitor work to enforce that safety property?

CONTROL-FLOW INTEGRITY

EECS 677/777: Software Security Evaluation

Drew Davidson

LAST TIME: REFERENCE MONITORS

REVIEW: LAST LECTURE

Enforce Safety Properties

Disallow a bad event from happening

Enforce Mechanisms

IRM
Wrapper RM
Kernel RM

LECTURE OUTLINE

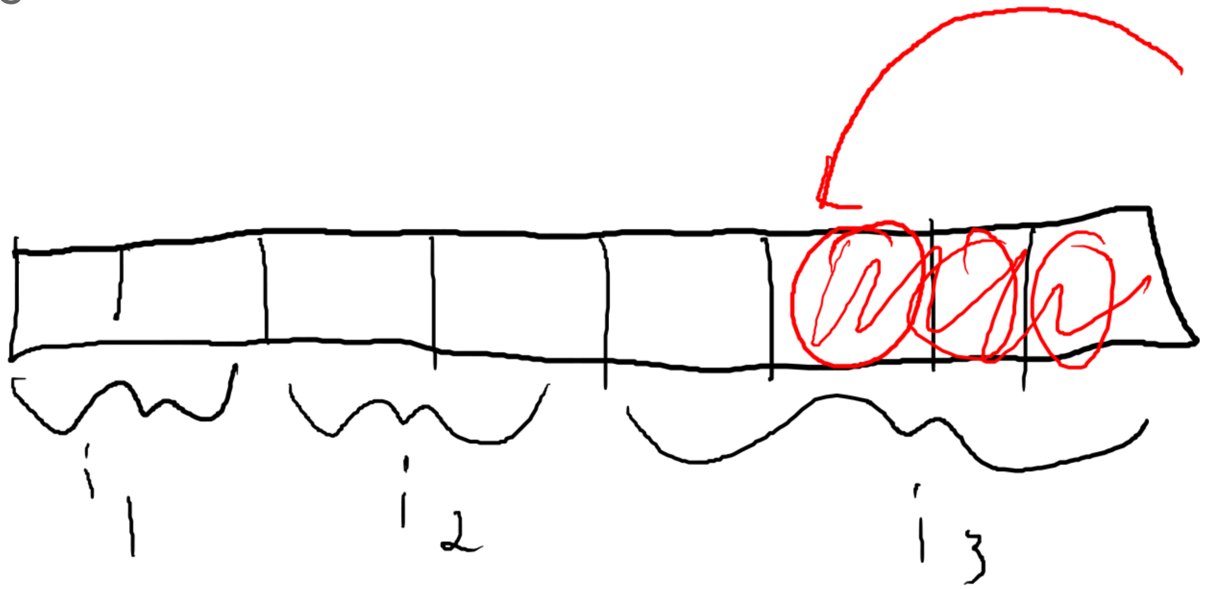
- Motivation
- Implementation considerations
- Practical manifestations

RE-ENGAGING AN OLD FOE…

MOTIVATION

Jumping where you shouldn't

- This certainly includes ROP
- Might also involve other attacks



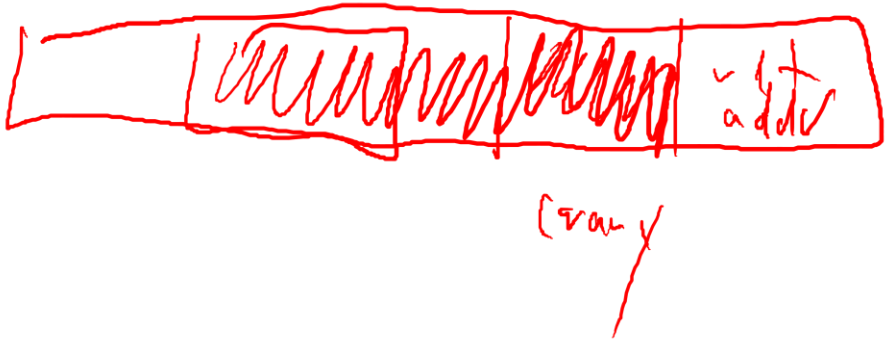
RE-ENGAGING AN OLD FOE…

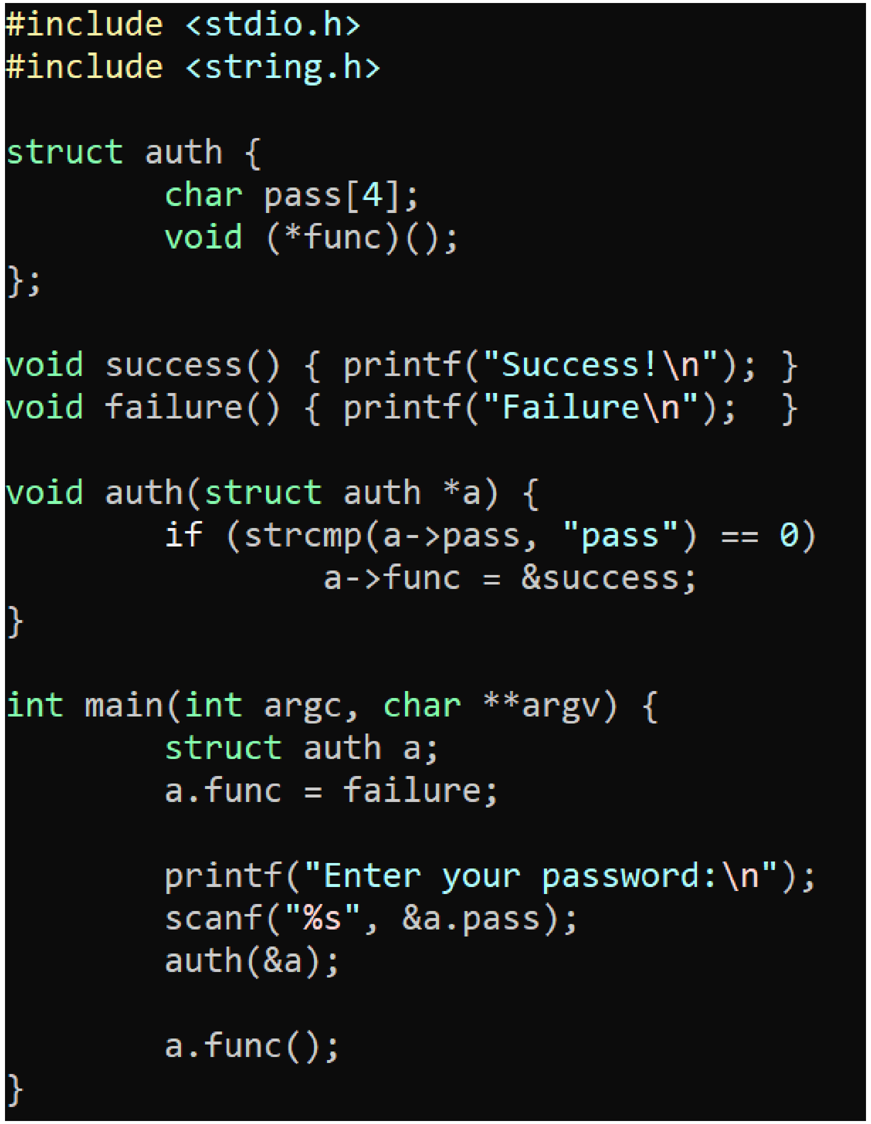
MOTIVATION

JUMPING WHERE YOU SHOULDN'T

- This certainly includes ROP
- Might also involve other attacks

LOOK, NO RET OVERWRITE!





```c
#include <stdio.h>
#include <string.h>

struct auth {
        char pass[4];
        void (*func)();
};

void success() { printf("Success!\n"); }
void failure() { printf("Failure\n");  }

void auth(struct auth *a) {
        if (strcmp(a->pass, "pass") == 0)
                a->func = &success;
}

int main(int argc, char **argv) {
        struct auth a;
        a.func = failure;

        printf("Enter your password:\n");
        scanf("%s", &a.pass);
        auth(&a);

        a.func();
}
```

LECTURE OUTLINE

- Motivation
- Implementation considerations
- Practical manifestations

KEY IDEA

KEEP THE CONTROL FLOW “ON RAILS”

HOW TO IMPLEMENT?

IMPLEMENTATION CONSIDERATIONS

Naïve Approach:

Encode the entire CFG into the program text

Issues:

Dynamic: overhead

Static: precision

LECTURE OUTLINE

- Motivation
- Implementation considerations
- Practical manifestations

INTEL CET

PRACTICAL MANIFESTATIONS

Control-Flow Enhancement Technology

Requires recompilation of software to support

Requires hardware support (!)

Scope

1) Prevent ret overwriting with a shadow stack

INTEL CET

PRACTICAL MANIFESTATIONS

Control-Flow Enhancement Technology

Requires recompilation of software to support

Requires hardware support (!)

Scope

1) Prevent ret overwriting with a shadow stack

2) Prevent indirect jumps into gadgets

INTEL CET

PRACTICAL MANIFESTATIONS

CET HARDWARE CHANGES

Altered semantics of the CALL and JMP

Moves a processor state machine into the WAIT_FOR_ENDBRANCH state

In WAIT_FOR_ENDBRANCH, next instruction must be the ENDBRANCH instruction

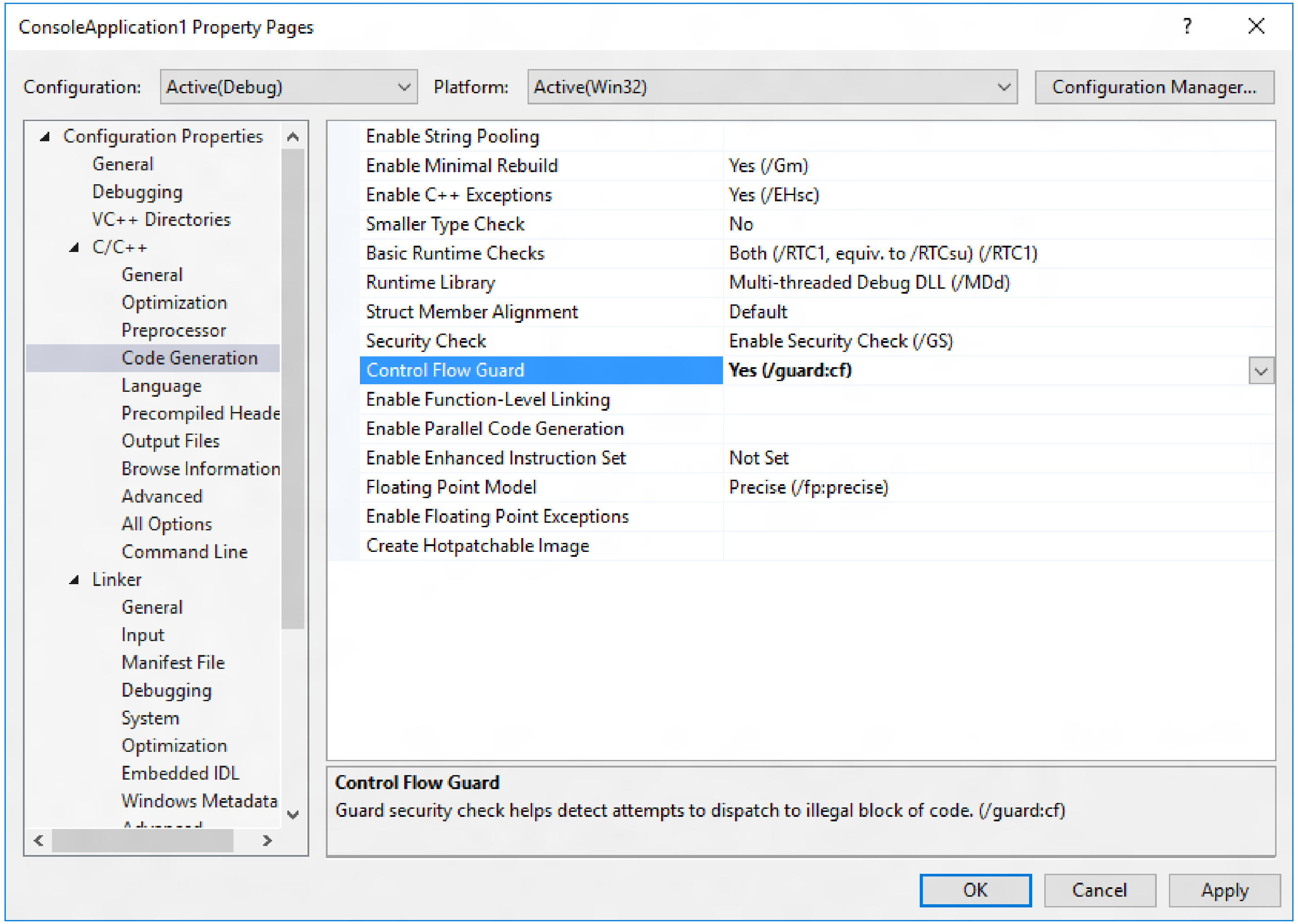
Added a new instruction at control-transfer targets

The new ENDBRANCH instruction

Backwards compatible

MICROSOFT CONTROL FLOW GUARD

PRACTICAL MANIFESTATIONS



Configuration: Active(Debug)

Platform: Active(Win32)

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)

HISTORICAL DETOUR

PRACTICAL MANIFESTATIONS: MS CONTROL-FLOW GUARD

HISTORICAL DETOUR

PRACTICAL MANIFESTATIONS: MS CONTROL-FLOW GUARD

Recall from last time…

ROP attacks considered harmful

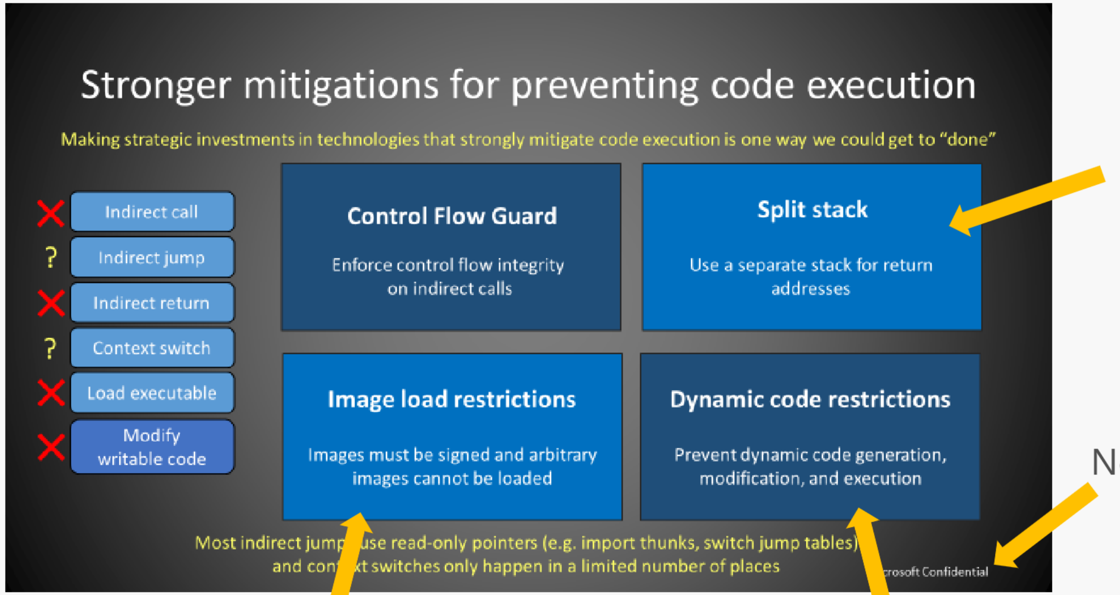
How industry responded

MS CFG as a case study in a lot of interesting aspects of software security

HISTORICAL DETOUR

PRACTICAL MANIFESTATIONS: MS CONTROL-FLOW GUARD

2012 Strategy Slide Deck



Stronger mitigations for preventing code execution

Making strategic investments in technologies that strongly mitigate code execution is one way we could get to "done"

- ✗ Indirect call
- ? Indirect jump
- ✗ Indirect return
- ? Context switch
- ✗ Load executable
- ✗ Modify writable code

Control Flow Guard
Enforce control flow integrity on indirect calls

Split stack
Use a separate stack for return addresses

Image load restrictions
Images must be signed and arbitrary images cannot be loaded

Dynamic code restrictions
Prevent dynamic code generation, modification, and execution

Most indirect jumps use read-only pointers (e.g. import thunks, switch jump tables) and context switches only happen in a limited number of places

Microsoft Confidential

Intel CET (hardware shadow stack)

Not Anymore ☺

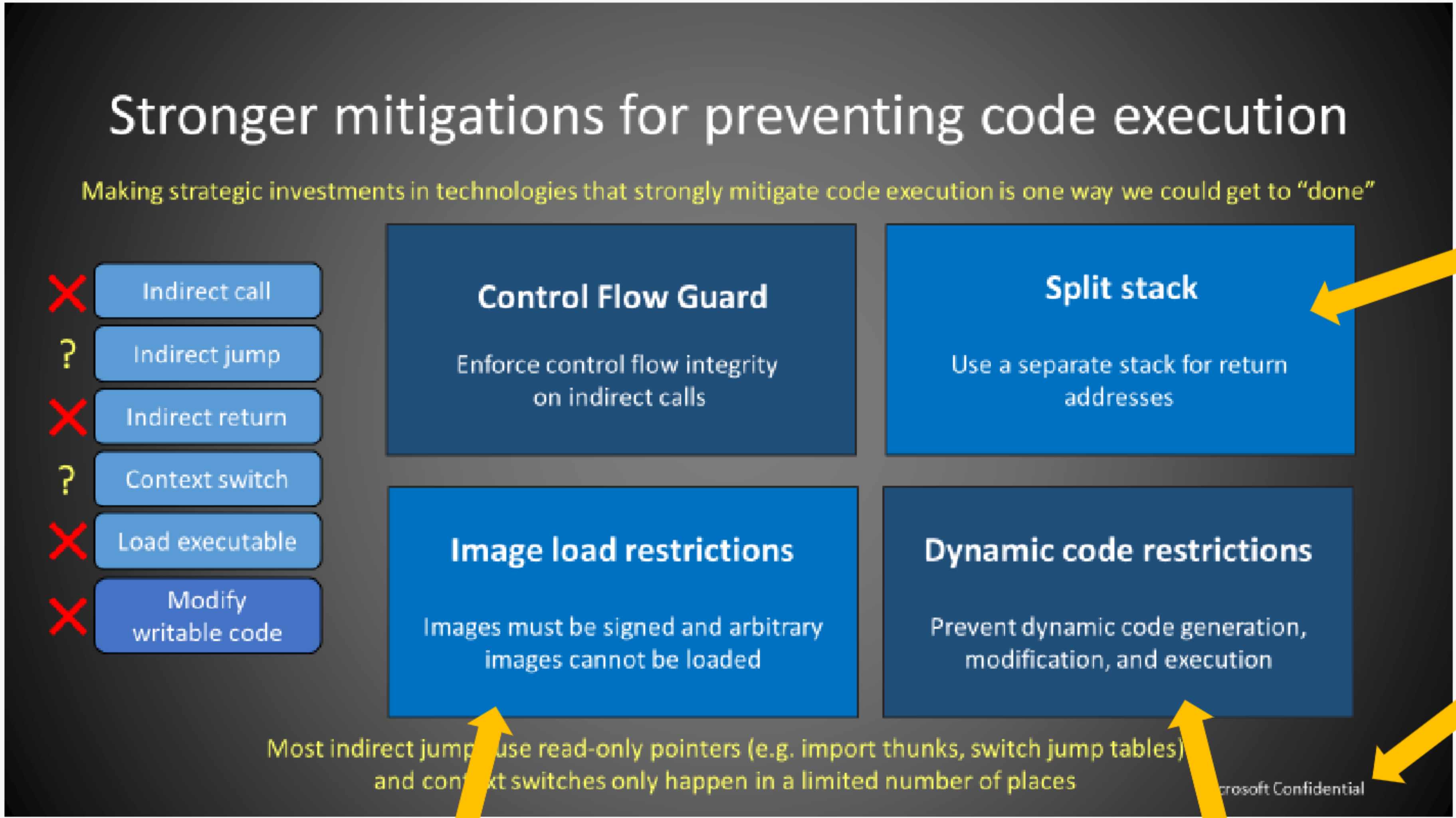
Code Integrity Guard (CIG) + NoChildProc + NoLowLabel + NoRemoteImage

Arbitrary Code Guard (ACG)

Source: https://github.com/Microsoft/MSRC-Security-Research/blob/master/presentations/2018_02_OffensiveCon/The%20Evolution%20of%20CFI%20Attacks%20and%20Defenses.pdf

2012 Strategy Slide Deck

DETOUR



Stronger mitigations for preventing code execution

Making strategic investments in technologies that strongly mitigate code execution is one way we could get to "done"

- ✗ Indirect call
- ? Indirect jump
- ✗ Indirect return
- ? Context switch
- ✗ Load executable
- ✗ Modify writable code

Control Flow Guard

Enforce control flow integrity on indirect calls

Split stack

Use a separate stack for return addresses

Image load restrictions

Images must be signed and arbitrary images cannot be loaded

Dynamic code restrictions

Prevent dynamic code generation, modification, and execution

Most indirect jumps use read-only pointers (e.g. import thunks, switch jump tables) and context switches only happen in a limited number of places

crosoft Confidential

Intel CET (hardware shadow stack)

Not Anymore ☺

Code Integrity Guard (CIG) + NoChildProc + NoLowLabel + NoRemoteImage

Arbitrary Code Guard (ACG)

HISTORICAL DETOUR

PRACTICAL MANIFESTATIONS: MS CONTROL-FLOW GUARD

This is an interesting talk!

I’d recommend you watch it: https://www.youtube.com/watch?v=oOqpl-2rMTw

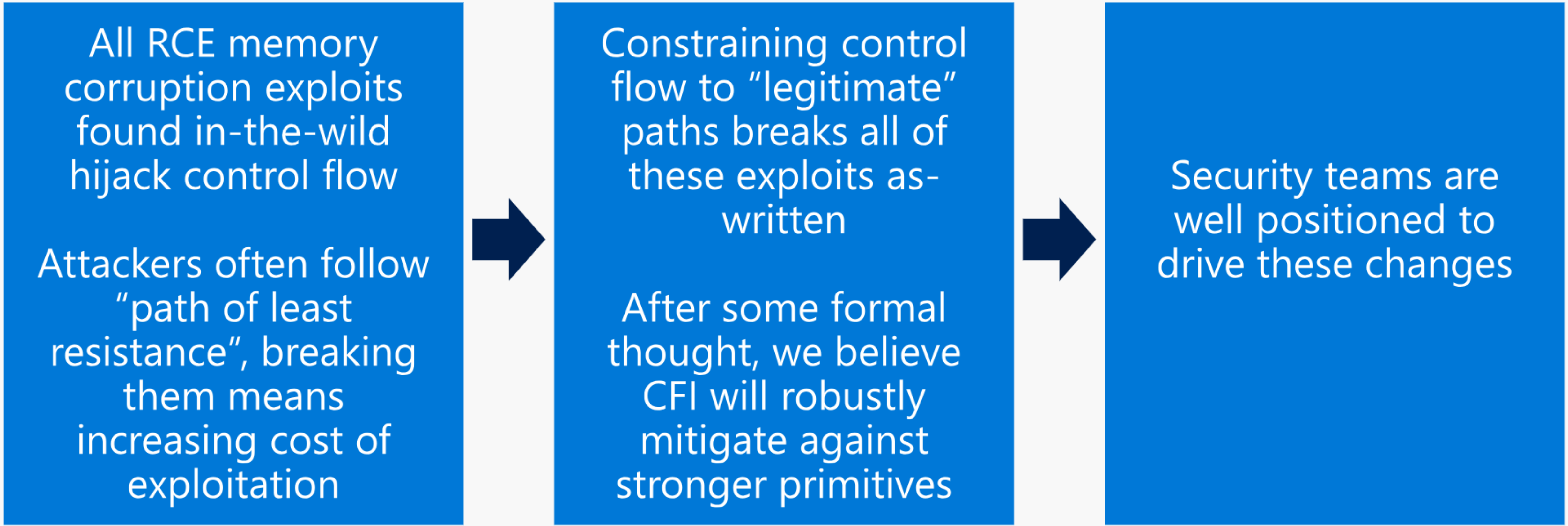
It comes with the historical burden of Control Flow guard

Widely-publicized issue that allowed it to be avoided

Theory

DETOUR

Microsoft’s overarching goal is to make exploitation financially infeasible or impossible



All RCE memory corruption exploits found in-the-wild hijack control flow

Attackers often follow “path of least resistance”, breaking them means increasing cost of exploitation

Constraining control flow to “legitimate” paths breaks all of these exploits as-written

After some formal thought, we believe CFI will robustly mitigate against stronger primitives

Security teams are well positioned to drive these changes

CFG had no formal threat model during very early development. Thought of as a way to kill ROP.

Hindsight is 20/20, but we did have formal thought around future exploit trends. See [1]

HISTORICAL DETOUR

PRACTICAL MANIFESTATIONS: MS CONTROL-FLOW GUARD

Control Flow guard has a historical burden

Widely-publicized issue that allowed it to be avoided

We’ll get to the actual workaround, but let’s talk about its impact

HISTORICAL DETOUR

PRACTICAL MANIFESTATIONS: MS CONTROL-FLOW GUARD

CONTROL FLOW GUARD

PRACTICAL MANIFESTATIONS

DETAILS

Precision: call needs to be a valid function entry point

Enforcement: OS verifies indirect control transfer destinations via a table in protected memory

PROTECTIONS

Protected destinations page in read-only memory

Read-only memory bit can be turned off by attacker ☹

CLANG’S CFI

PRACTICAL MANIFESTATIONS

DETAILS

Precision: call needs to match type signature

Enforcement: compiler-inserted checks

WRAP-UP