EXERCISE #23

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?

EXERCISE #23 SOLUTION

REFERENCE MONITORS REVIEW

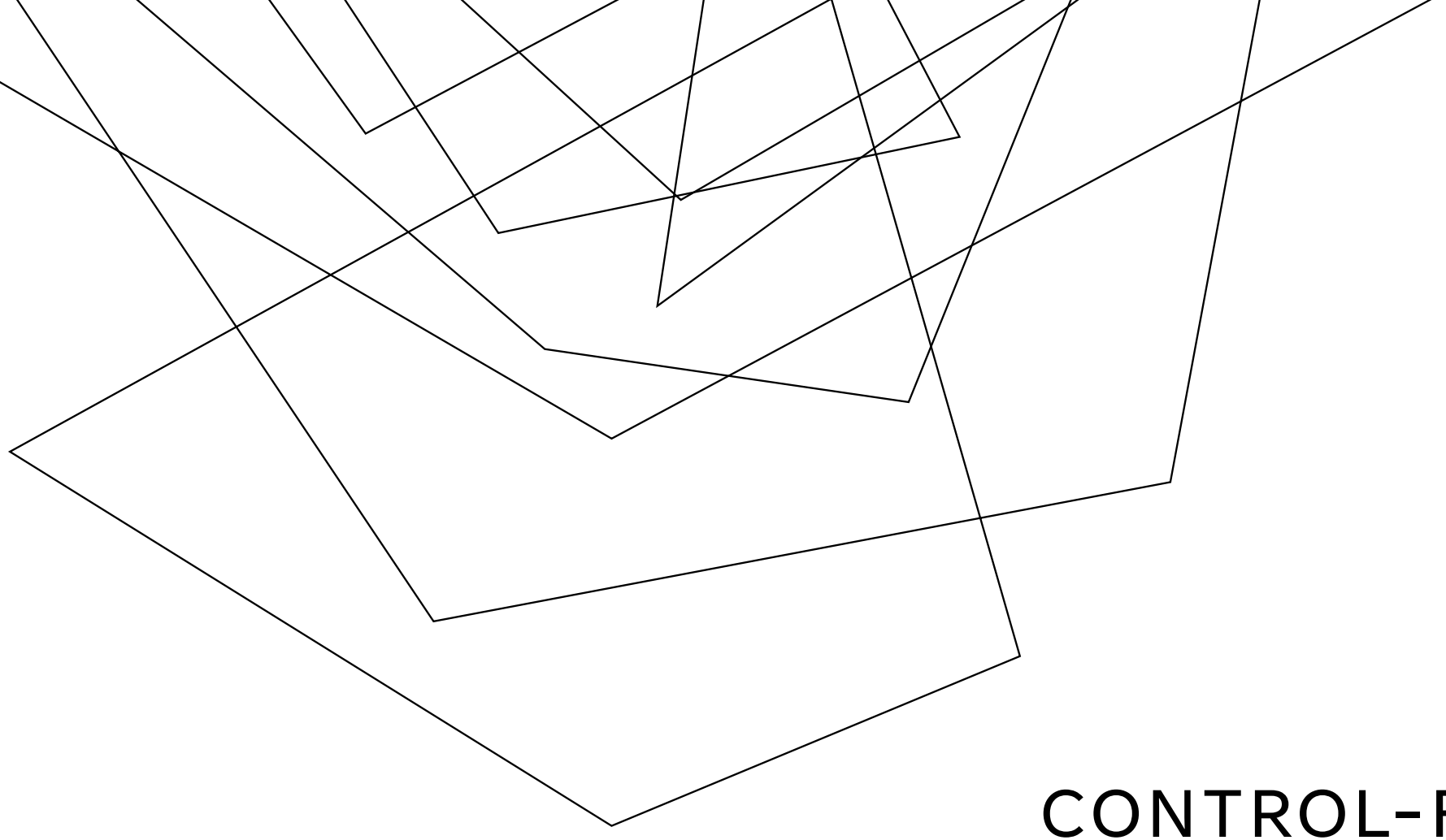
ADMINISTRIVIA AND ANNOUNCEMENTS

Second reading assigned

- The original paper on CFI

Basically halfway through the semester

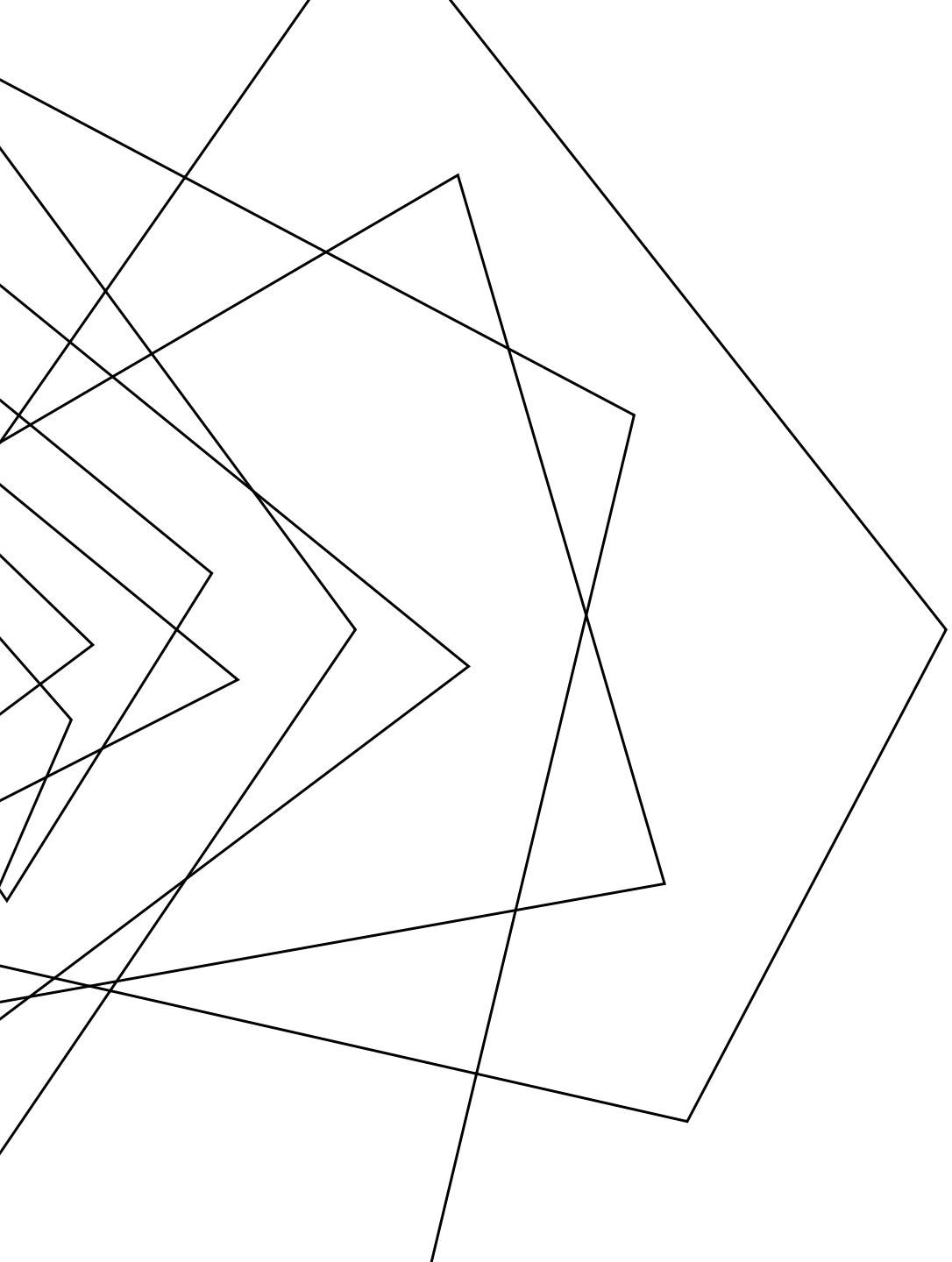
- Time to check in on how things are going



CONTROL-FLOW INTEGRITY

EECS 677: Software Security Evaluation

Drew Davidson



TOPIC CONTEXT

CONTEMPLATED A FORM OF ATTACK,
LEFT WITH A HINT OF DEFENSES

LAST TIME: REFERENCE MONITORS

REVIEW: LAST LECTURE

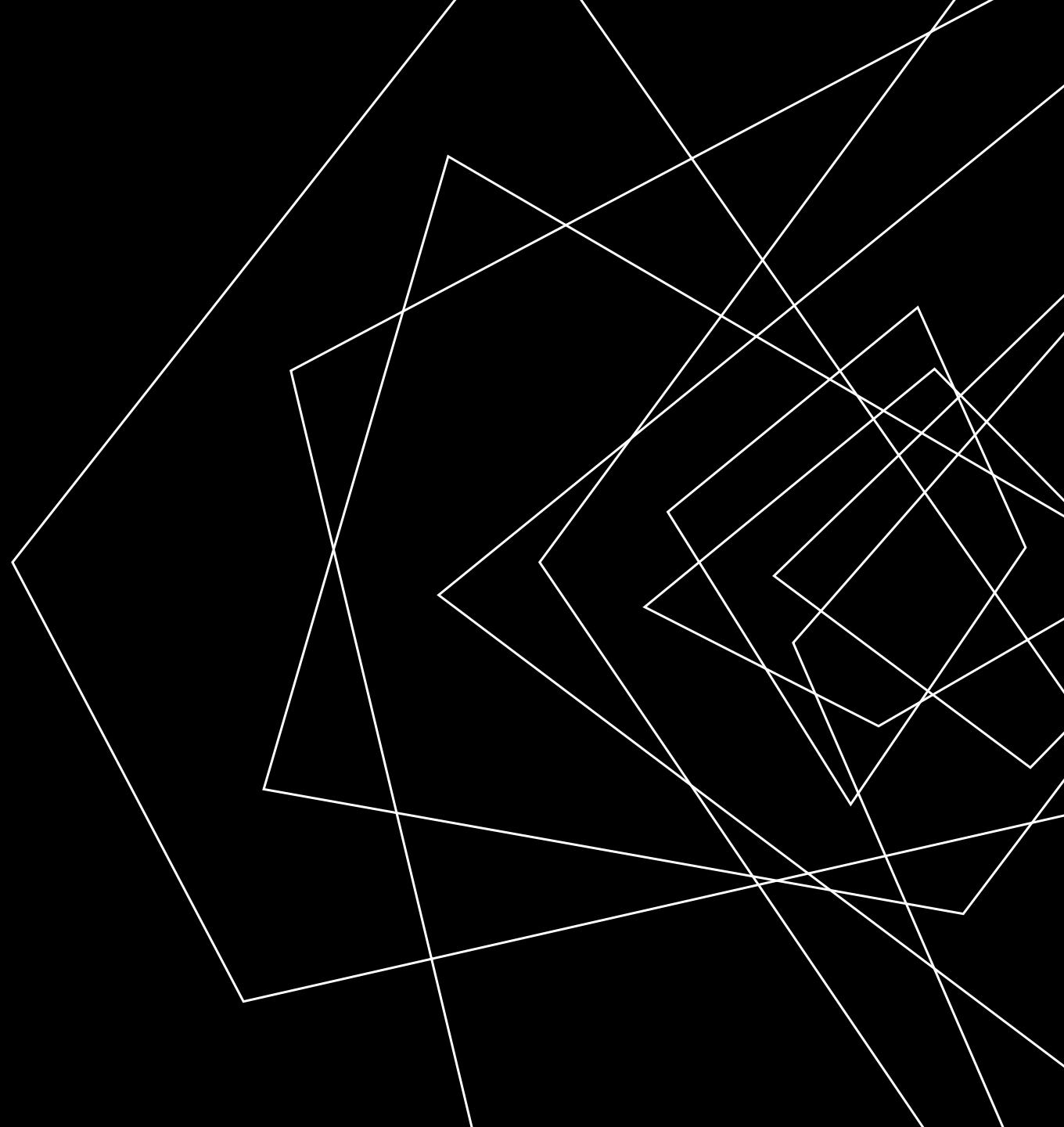
Ensure Adherence to a Safety Policy

Halt the program is an action would violate the policy

Keep the program “on the rails”

LECTURE OUTLINE

- Motivation
- Implementation considerations
- Practical manifestations

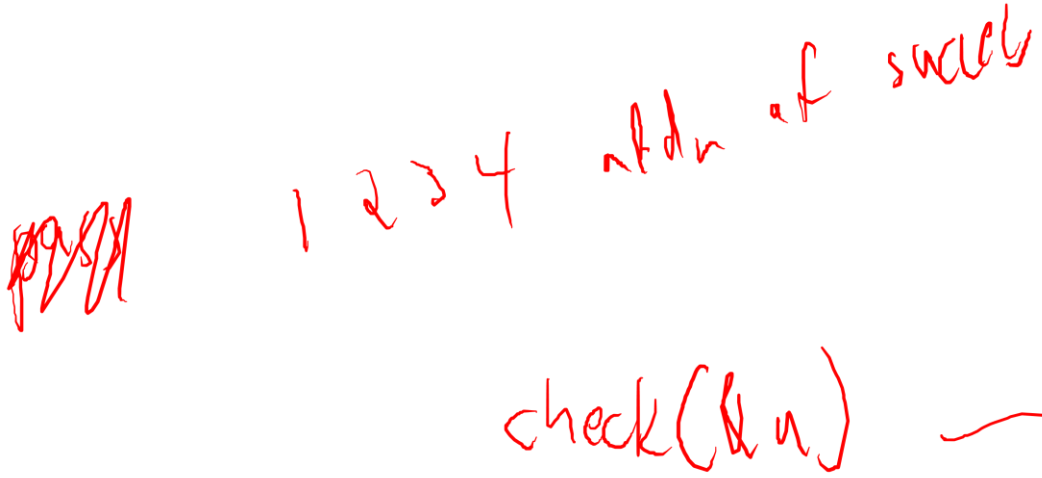


WE KNOW THE PROBLEM

MOTIVATION

JUMPING WHERE YOU SHOULDN'T

- This certainly includes ROP
- Might also involve other attacks




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

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

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

void check(struct auth *a) {
        if (strcmp(a->pass, "pass") == 0)
                a->func = &success;
        else
                a->func = &failure;
}
int main(int argc, char **argv) {
        struct auth a;

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

        a.func(&a);
}
```

WE KNOW THE PROBLEM

MOTIVATION

JUMPING WHERE YOU SHOULDN'T

- This certainly includes ROP
- Might also involve other attacks

LOOK, NO RET OVERWRITE!

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

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

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

void check(struct auth *a) {
        if (strcmp(a->pass, "pass") == 0)
                a->func = &success;
        else
                a->func = &failure;
}
int main(int argc, char **argv) {
        struct auth a;

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

        a.func(&a);
}
```

WE KNOW THE PROBLEM

MOTIVATION

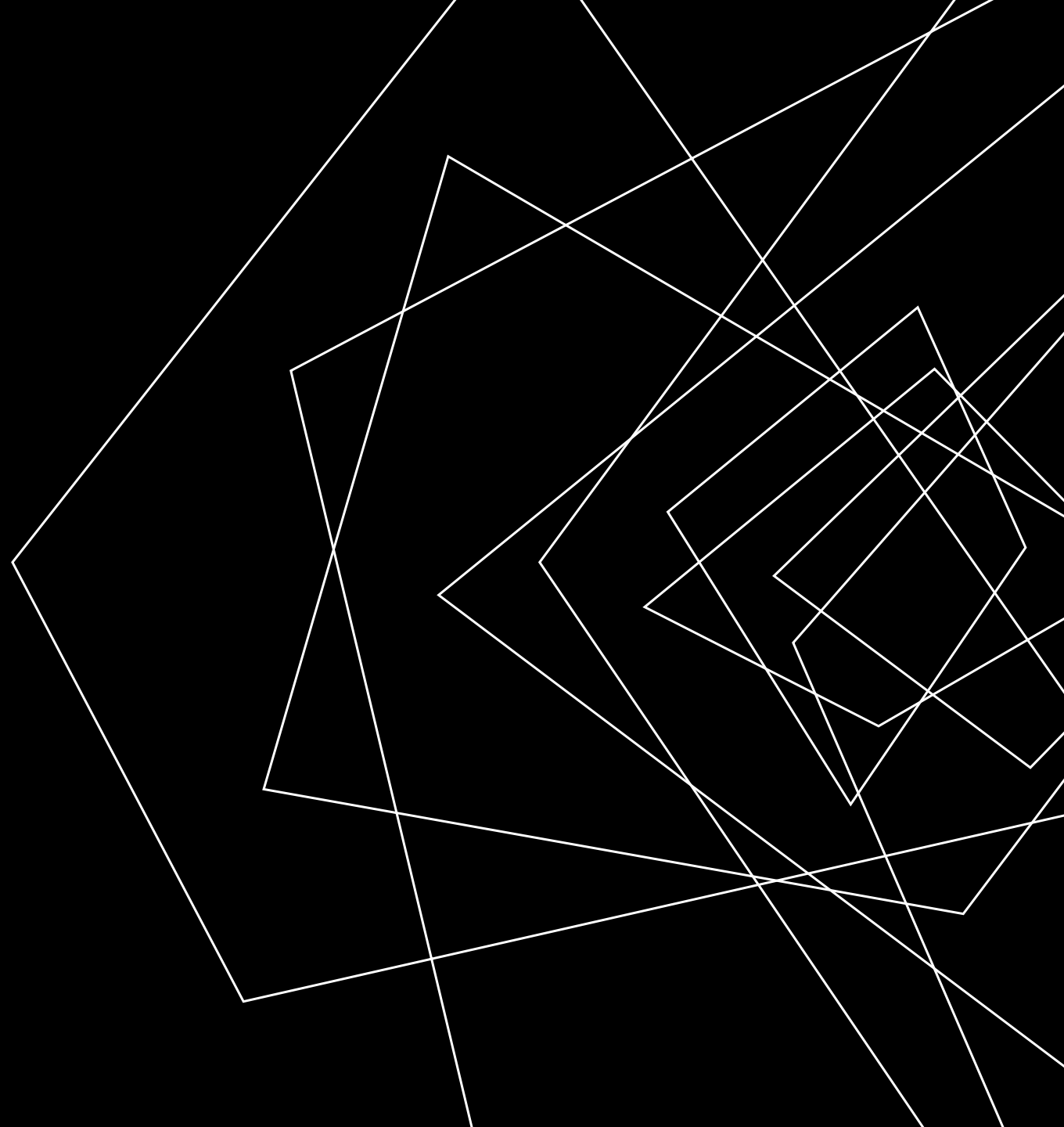
JUMPING WHERE YOU SHOULDN'T

- This certainly includes ROP
- Might also involve other attacks

LOOK, NO RET OVERWRITE!

LECTURE OUTLINE

- Motivation
- Implementation considerations
- Practical manifestations

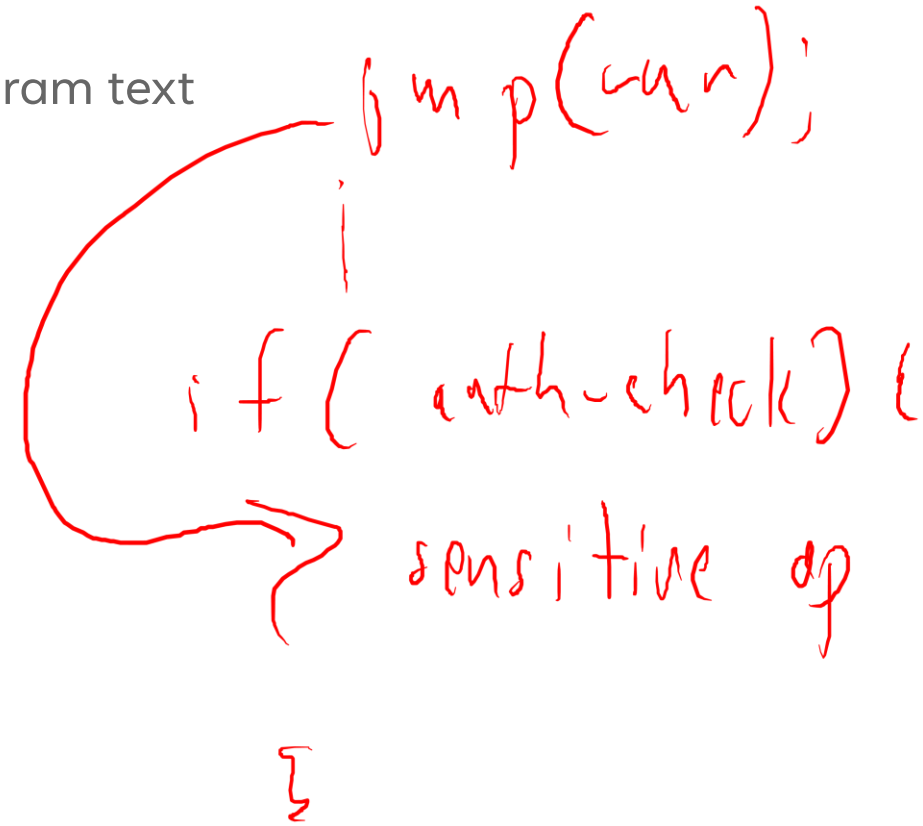


HOW TO IMPLEMENT?

IMPLEMENTATION CONSIDERATIONS

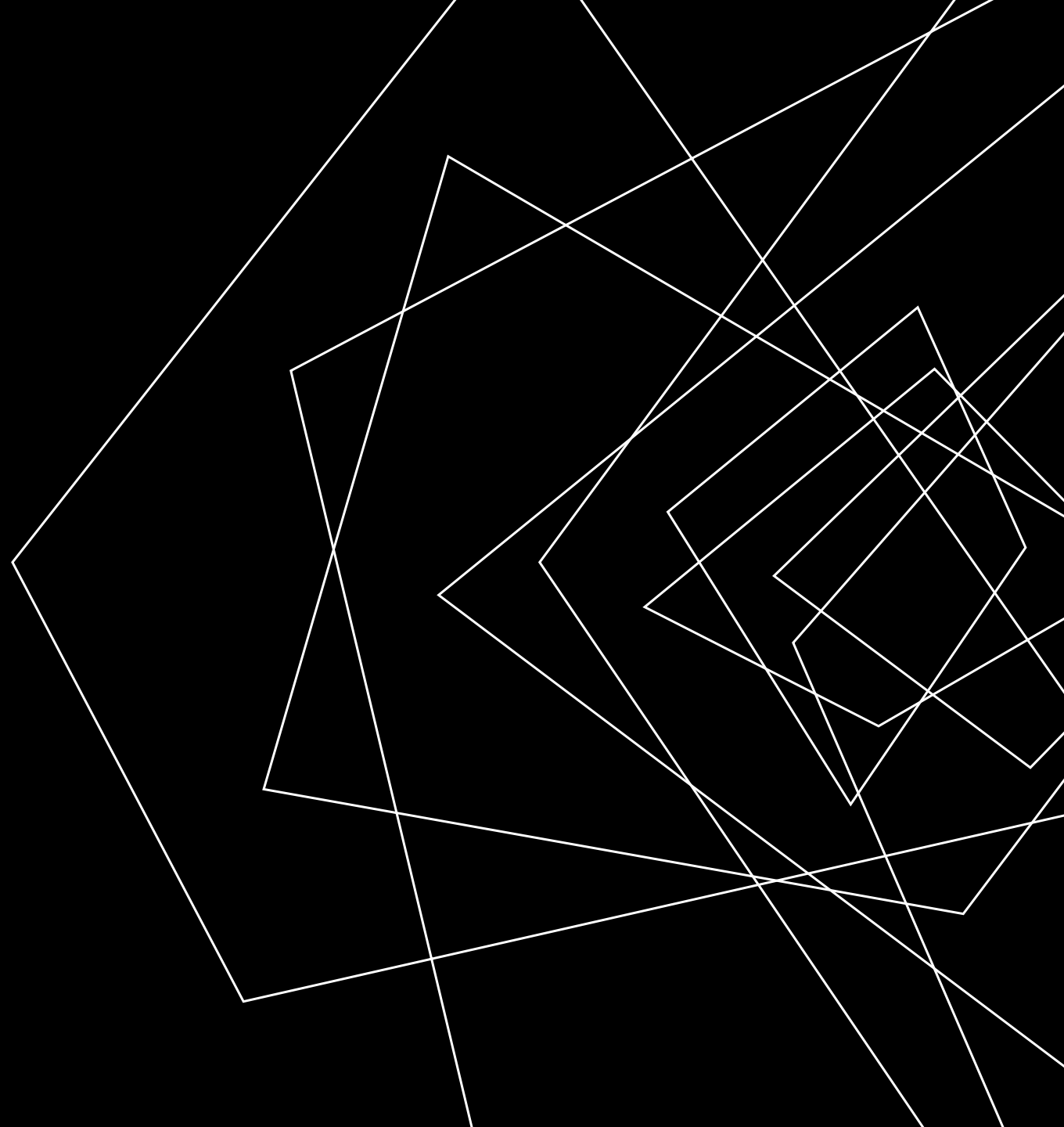
NAÏVE APPROACH:

Encode the entire ICFG into the program text



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) (SHSTK) – Shadow Stack: Prevent ret overwriting with a shadow stack

2) (IBT) – Indirect Branch Tracking: Prevent indirect jumps into gadgets

INTEL CET – USAGE

PRACTICAL MANIFESTATIONS

HW Support

Intel 11th Gen or Later / AMD Ryzen 5000+

OS Support

Windows W10 19H1 (v1903)

Linux: kernel 6.6

Compiler Flags

gcc/llvm: -fcf-protection=full

Visual Studio: /CETCOMPAT

On Linux, possible to check if the program has CET:
readelf –n <binary>
Should include the note

```
Displaying notes found in: .note.gnu.property
  Owner                Data size        Description
  GNU                  0x00000020       NT_GNU_PROPERTY_TYPE_0
      Properties: x86 feature: IBT, SHSTK
        x86 ISA needed: x86-64-baseline
```

INTEL CET
PRACTICAL MANIFESTATIONS

CET Hardware changes

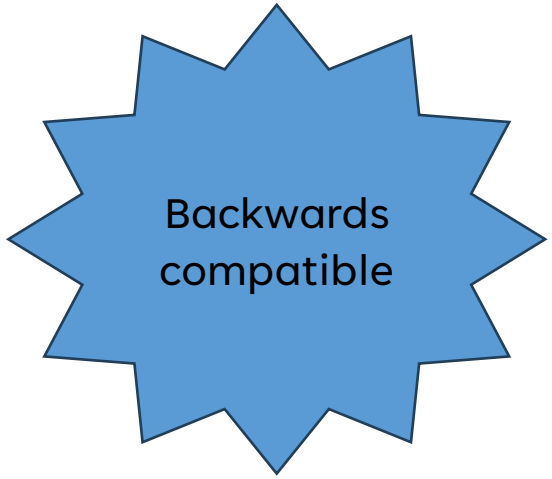
Altered semantics of the CALL and JMP

Moves a processor state machine into the WAIT_FOR_ENDBRANCH state

In WAIT_FOR_ENDBRANCH, next instruction must be the ENDBRANCH instruction

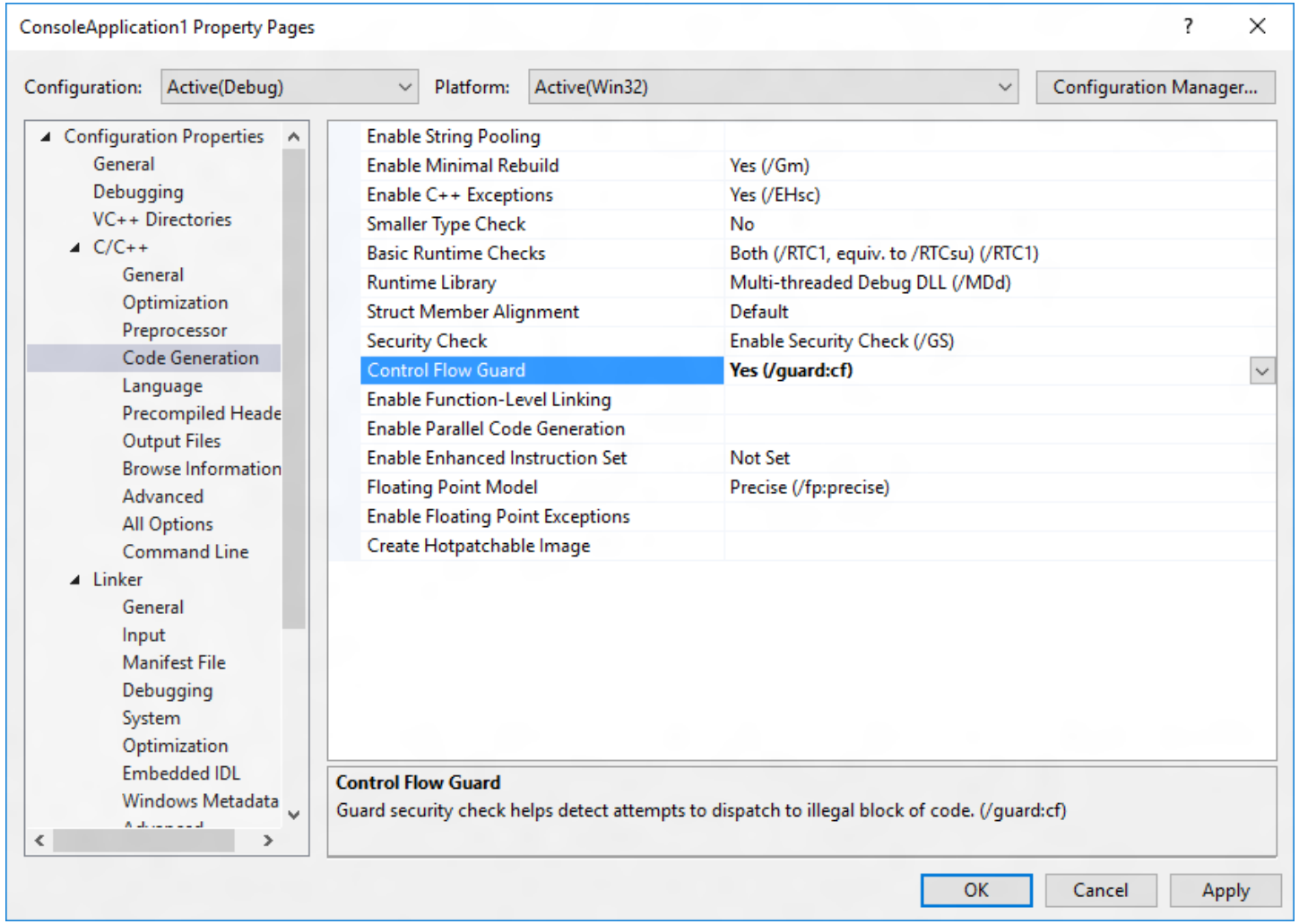
Added a new instruction at control-transfer targets

The new ENDBRANCH instruction



MICROSOFT CONTROL FLOW GUARD

PRACTICAL MANIFESTATIONS



HISTORICAL DETOUR

PRACTICAL MANIFESTATIONS: MS CONTROL-FLOW GUARD

HISTORICAL DETOUR

PRACTICAL MANIFESTATIONS: MS CONTROL-FLOW GUARD

RECALL FROM LAST TIME...

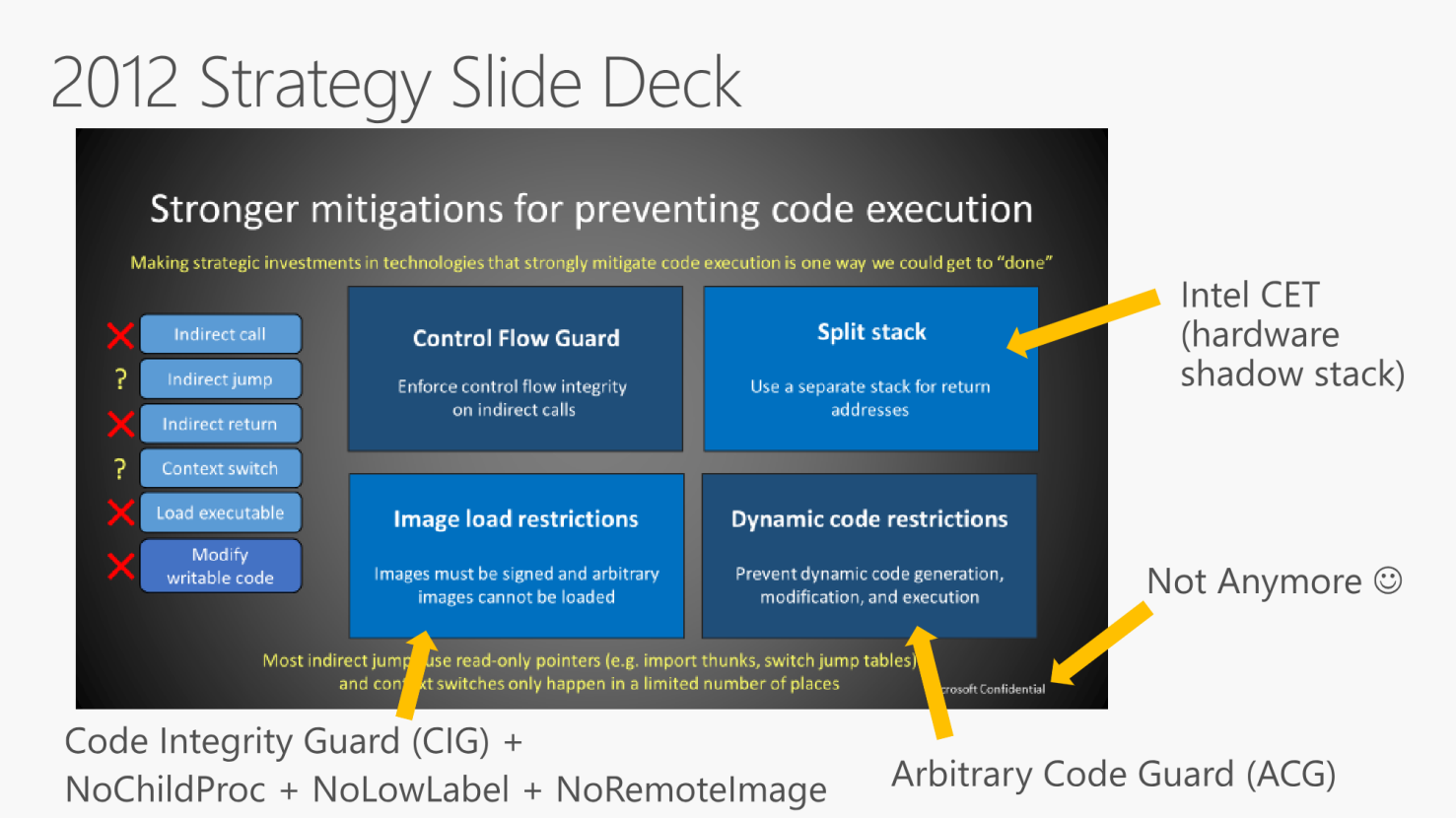
ROP attacks considered harmful

HOW INDUSTRY RESPONDED

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

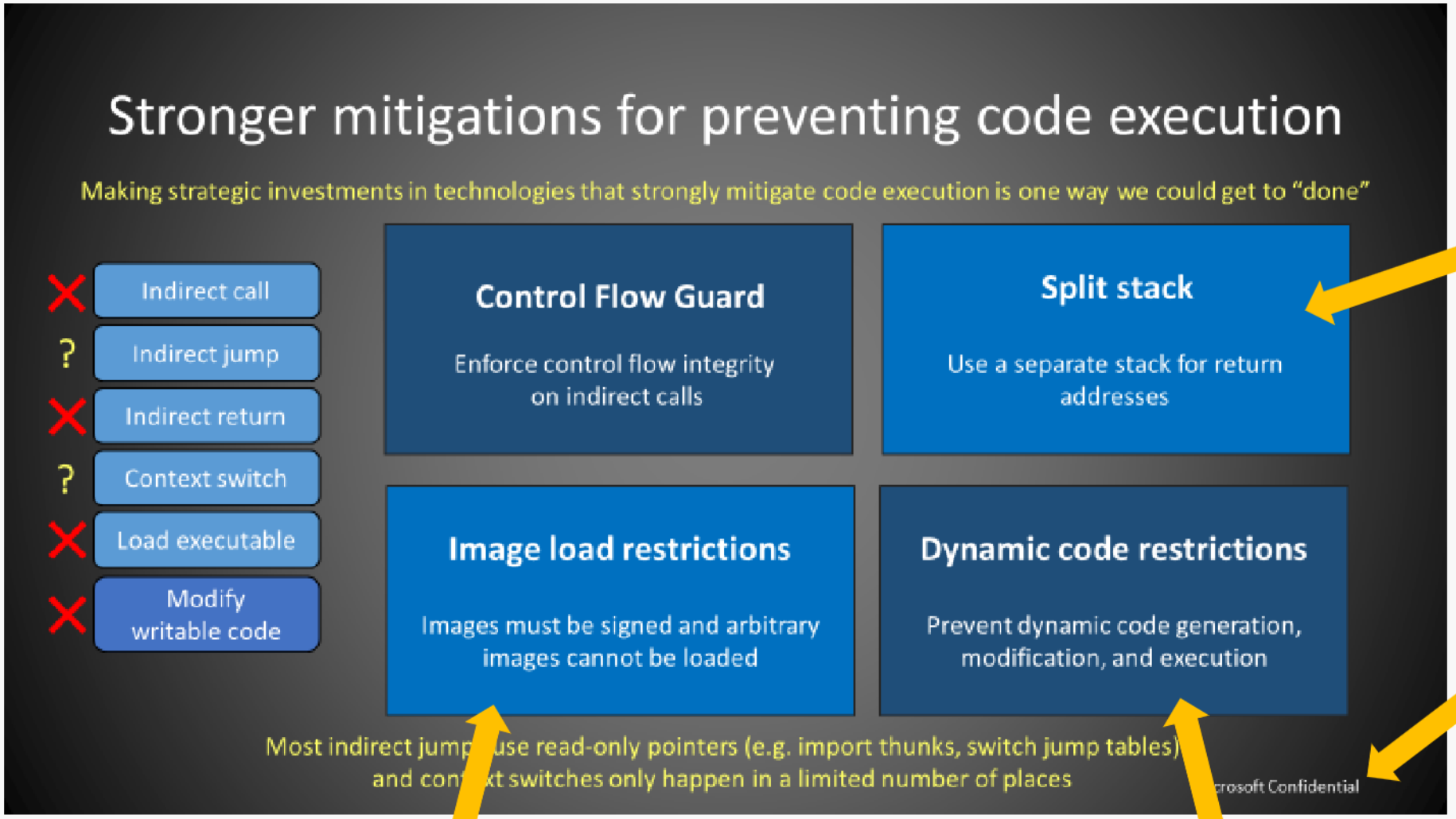
HISTORICAL DETOUR

PRACTICAL MANIFESTATIONS: MS CONTROL-FLOW GUARD



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



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

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

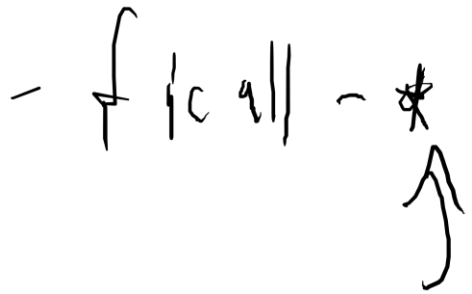
Read-only memory bit can be turned off by attacker ☹

CLANG'S CFI
PRACTICAL MANIFESTATIONS

DETAILS

Precision: call needs to match type signature

Enforcement: compiler-inserted checks



WRAP-UP