

# EXERCISE #5

## LLVM CALLS REVIEW

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

Write LLVM IR corresponding to the following C snippet:

```
extern int rand();
int foo() {
    return rand();
}
```

```
define i32 @main() {
    %2 = call i32 (...) @rand()
    ret i32 %2
}

declare i32 @rand(...)
```

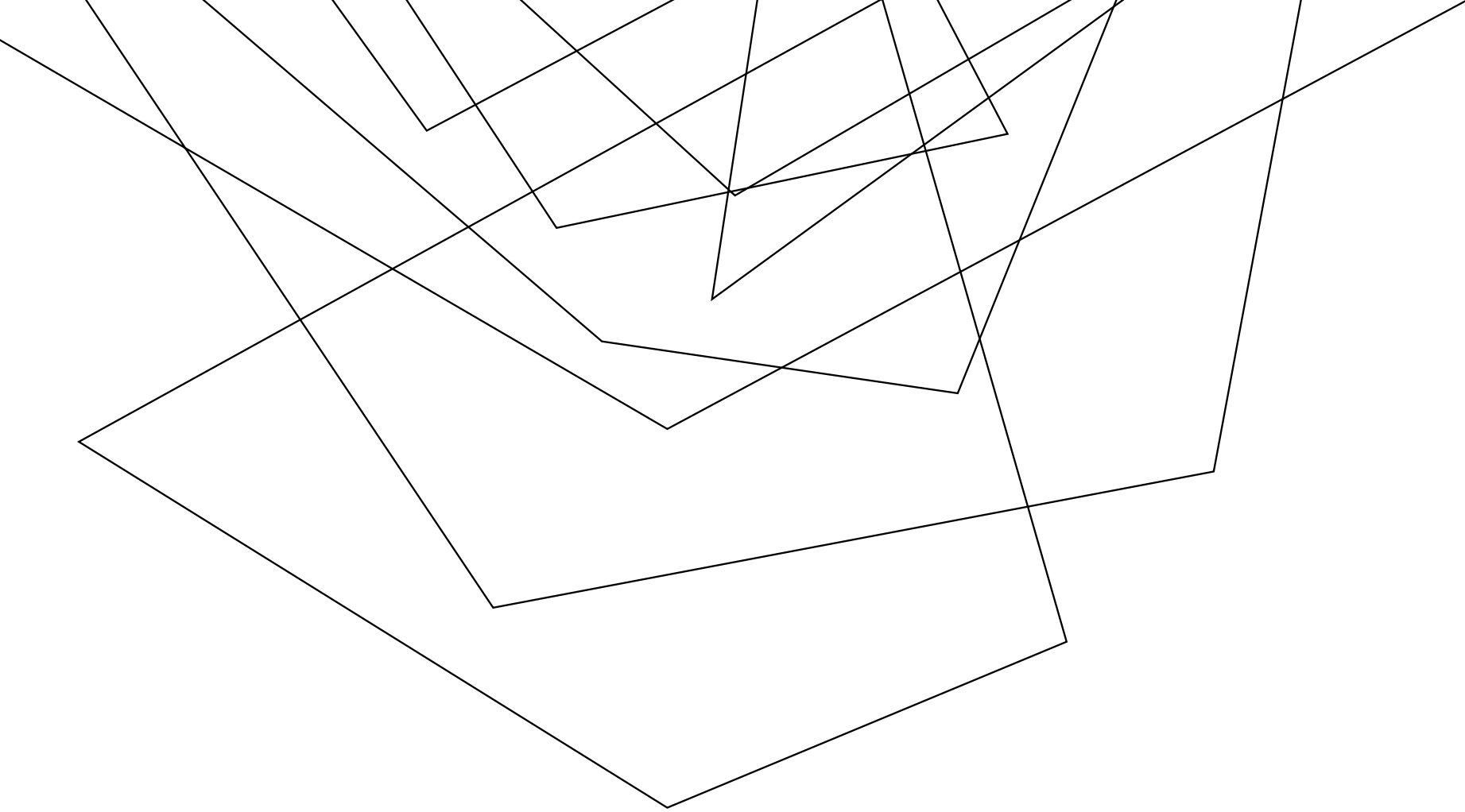


Quiz 1 on Friday

Review Session on Thursday 6:30 PM

https://drew.hquidson.  
cool/where

**ADMINISTRIVIA  
AND  
ANNOUNCEMENTS**



# MALWARE

EECS 677: Software Security Evaluation

Drew Davidson

# LAST TIME: LLVM CALLS

REVIEW: LAST LECTURE

HOW TO INVOKE YOUR OWN FUNCTIONS

HOW TO DECLARE THE EXISTENCE OF EXTERNAL FUNCTIONS

~~@~~ declare i32 @abs (...)

*(Handwritten note: An arrow points from the handwritten 'i32' above to the 'i32' in the code snippet.)*

# LAST TIME: LLVM CALLS

REVIEW: LAST LECTURE

## HOW TO AUTOMATICALLY GENERATE LLVM IR

*Clay* *llvm*  
`llvm-emit -S -emit-llvm <prog>`

Creates a <prog>.ll file

# SOME STUFF YOU MIGHT SEE IN PROGRAMS

EXTRA INFO: LLVM

```
#include <stdio.h>
int main(){
    printf("hello world\n");
}
```

delete

keep

delete

```
ModuleID = 'coolprog.c'
source_filename = "coolprog.c"
target datalayout = "e-m:e-p270:32:32-p271:32:32-p272:64:64-i64:64-i128:128-f80:128-n8:16:32:64-S128"
target triple = "x86_64-pc-linux-gnu"

@.str = private unnamed_addr constant [13 x i8] c"hello world\0A\00", align 1

; Function Attrs: noinline nounwind optnone uwtable
define dso_local i32 @main() #0 {
    %1 = call i32 @printf(ptr, ...) @printf(ptr noundef @.str)
    ret i32 0
}

declare i32 @printf(ptr noundef, ...) #1

attributes #0 = { noinline nounwind optnone uwtable "frame-pointer"="all" "min-legal-vector-width"="0" "no-trapping-math"="true" "stack-protector-buffer-size"="8" "target-cpu"="x86-64" "target-features"="+cmov,+cx8,+fxsr,+mmx,+sse,+sse2,+x87" "tune-cpu"="generic" }
attributes #1 = { "frame-pointer"="all" "no-trapping-math"="true" "stack-protector-buffer-size"="8" "target-cpu"="x86-64" "target-features"="+cmov,+cx8,+fxsr,+mmx,+sse,+sse2,+x87" "tune-cpu"="generic" }

!llvm.module.flags = !{!0, !1, !2, !3, !4}
!llvm.ident = !{!5}

!0 = !{i32 1, !"wchar_size", i32 4}
!1 = !{i32 8, !"PIC Level", i32 2}
!2 = !{i32 7, !"PIE Level", i32 2}
!3 = !{i32 7, !"uwtable", i32 2}
!4 = !{i32 7, !"frame-pointer", i32 2}
!5 = !{!"Ubuntu clang version 18.1.3 (1ubuntu1)"}

```

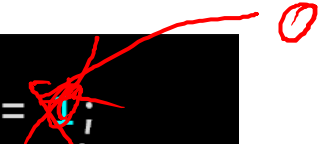
# LLVM CASTING: SIGN EXTENSION

EXTRA INFO: LLVM

## THE (SLIGHTLY UNFORTUNATELY NAMED) SEXT INSTRUCTION

Promotes a sign-extended value of one type to a larger type

```
int main(){  
    int a = 1;  
    long int b = a;  
    return 0;  
}
```



```
define i32 @main() {  
    %reg1 = add i32 0, 0  
    %reg2 = sext i32 %reg1 to i64  
    ret i32 0  
}
```

# LLVM TYPE CONVERSION

EXTRA INFO: LLVM IR

## INTTOPTR

```
#include <stdio.h>
extern int fgetc(FILE *);

int main(){
    fgetc(0);
    return 0;
}
```

```
define i32 @main() {
    %arg = inttoptr i64 0 to ptr
    %res = call i32 @fgetc(ptr %arg)
    ret i32 0
}

declare i32 @fgetc(ptr noundef)
```

Side note: LLVM does actually have the concept of the null pointer, so another program could be

```
define i32 @main() {
    %2 = call i32 @fgetc(ptr null)
    ret i32 0
}

declare i32 @fgetc(ptr)
```



# LLVM TYPE CONVERSION

EXTRA INFO: LLVM IR

INTTOPTR (AND PTRTOINT) AS OPERAND

```
int main(){
    int * var;
    var = (int *)5;
    return (int)var;
}
```

```
define i32 @main() #0 {
    %reg1 = alloca ptr
    store ptr inttoptr (i64 5 to ptr), ptr %reg1
    %reg2 = load ptr, ptr %reg1
    %reg3 = ptrtoint ptr %reg2 to i32
    ret i32 %reg3
}
```



## **CLASS PROGRESS**

WE'VE GOT OURSELVES A WAY TO  
REPRESENT PROGRAMS

WHAT ARE WE TRYING TO PREVENT?

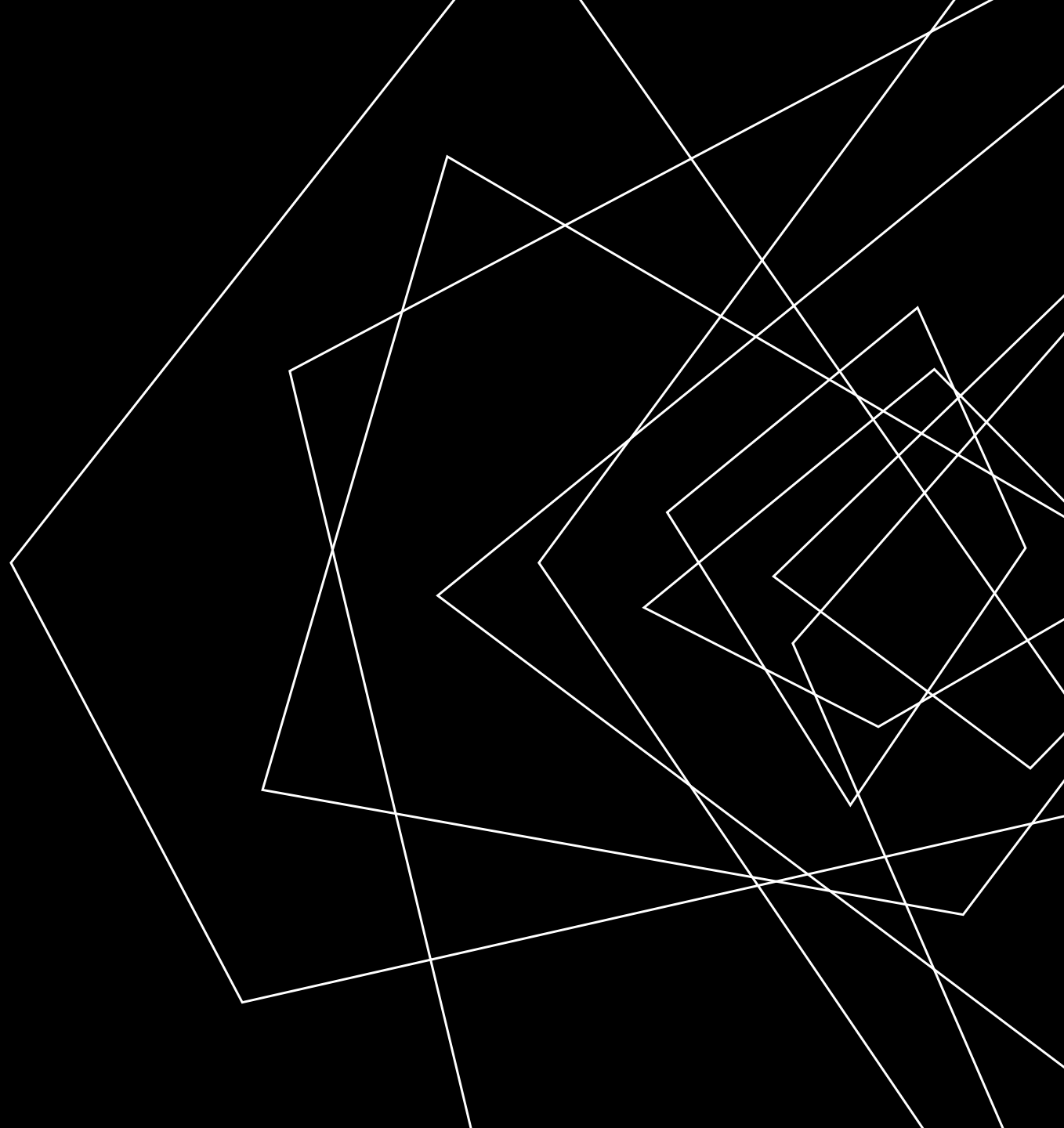
# OVERVIEW

PREVENTING BAD STUFF FROM  
HAPPENING IN A PROGRAM



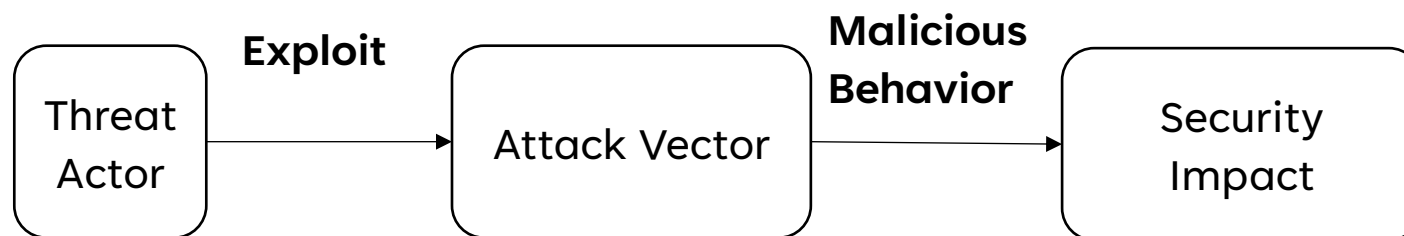
# LECTURE OUTLINE

- Terms/Concepts
- What can go wrong?
- How incidents happen



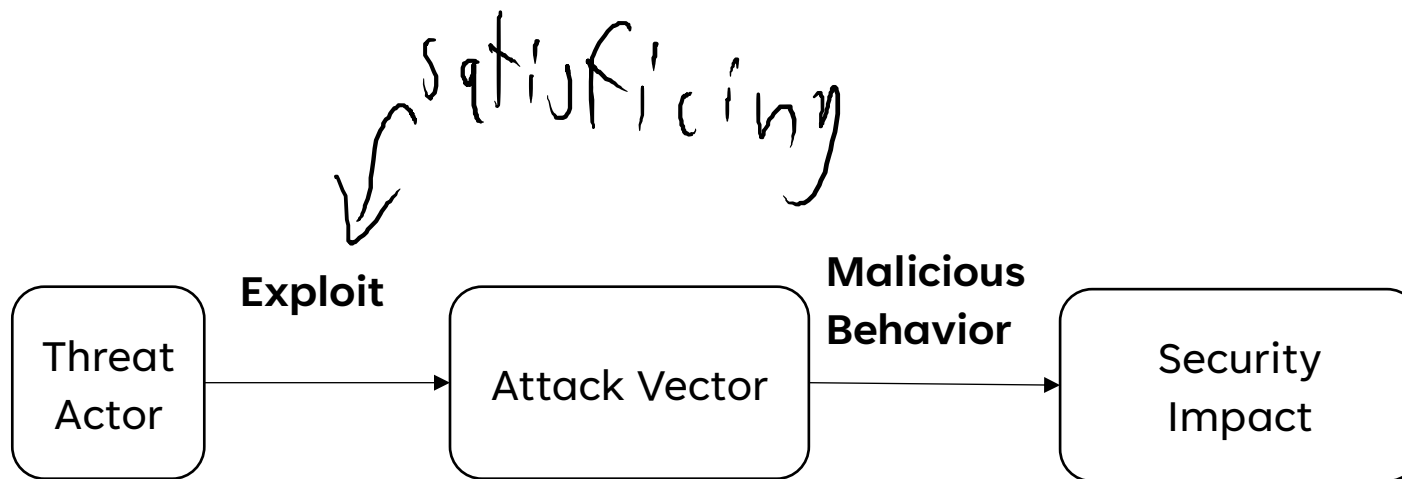
# A SECURITY INCIDENT WORKFLOW

## MALWARE – TERMS AND CONCEPTS



# A SECURITY INCIDENT WORKFLOW

## MALWARE - TERMS AND CONCEPTS



*satisficing*

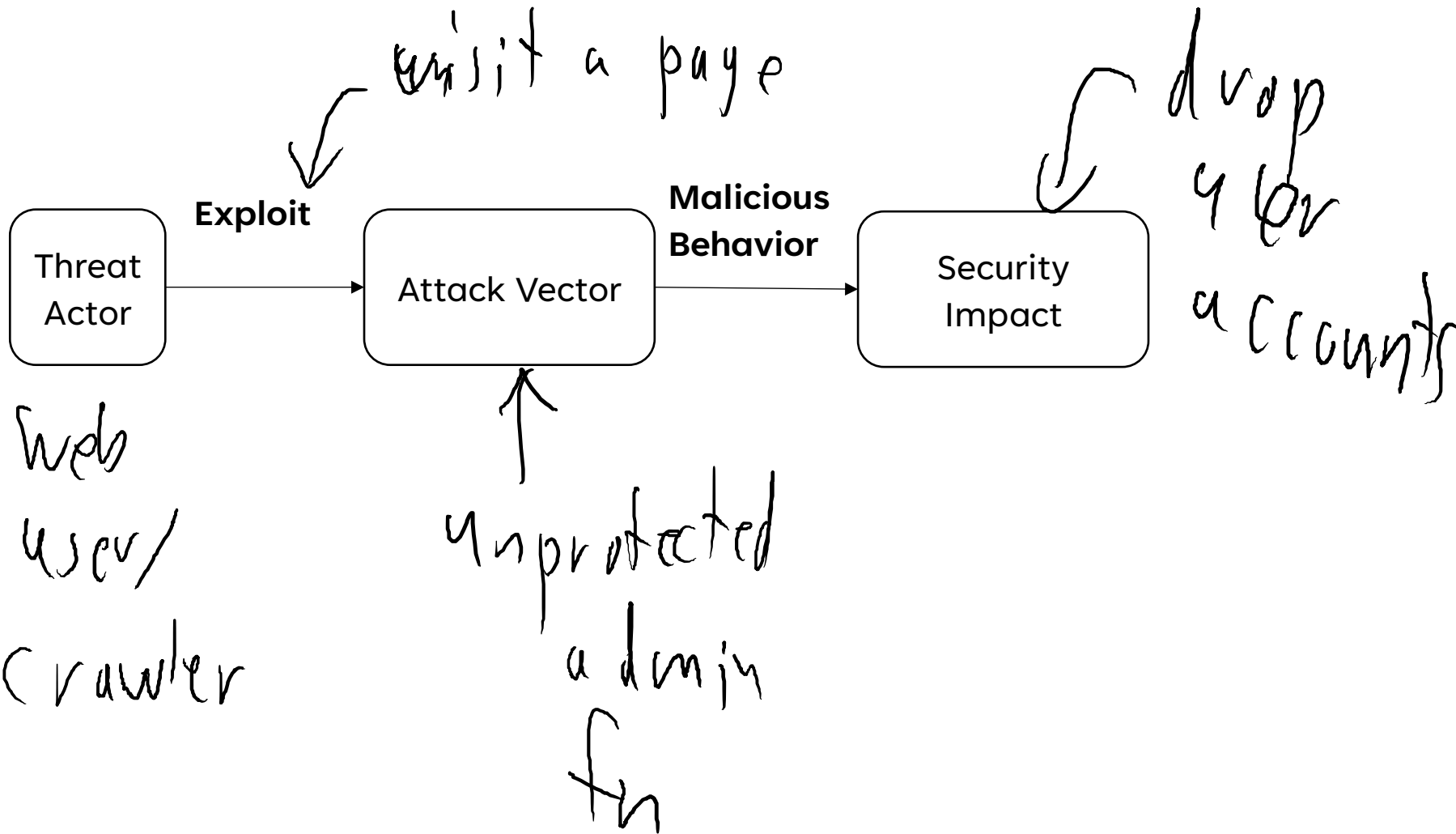
*data leak*

Scenario #1

*Malicious  
App  
Published*

# A SECURITY INCIDENT WORKFLOW

## MALWARE - TERMS AND CONCEPTS



Scenario #2

web  
user/  
crawler

# ADVERSARIES

## MALWARE – TERMS AND CONCEPTS

Any one  
attempting  
to

~~cause~~

cause

Security incidents





# THREAT MODELS

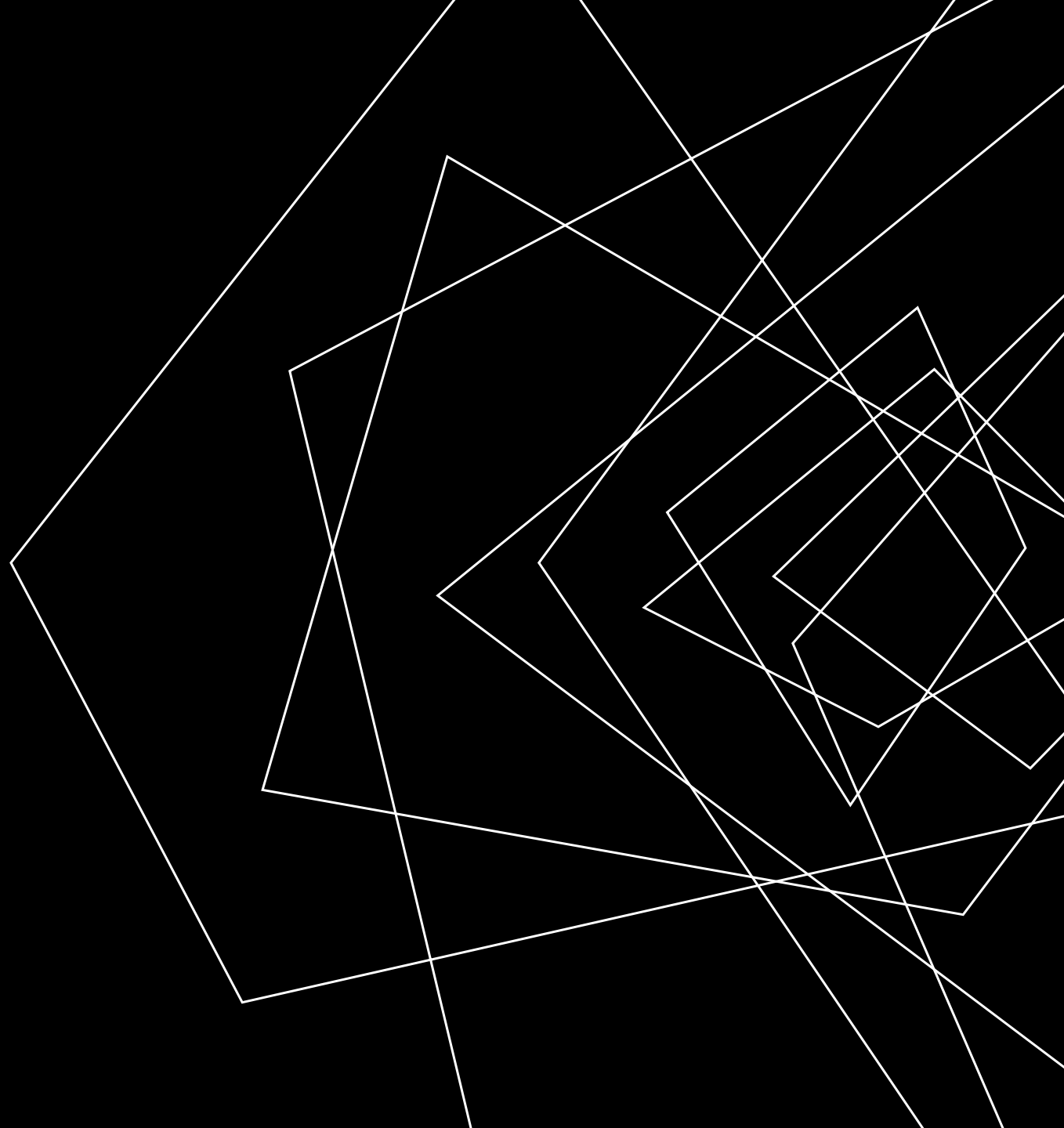
## MALWARE - TERMS AND CONCEPTS

- 1) Set of adversary capabilities
- 2) Set of goals



# LECTURE OUTLINE

- Terms/Concepts
- What can go wrong?
- How incidents happen



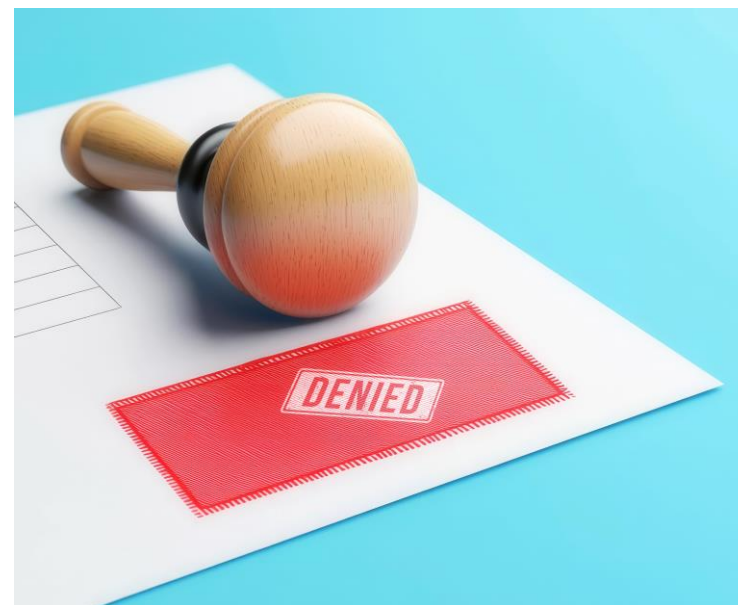
# THE CIA TRIAD

## MALWARE – WHAT CAN GO WRONG?

C onfidentiality  
I ntegrity  
A vailability

# DENIAL OF SERVICE

## MALWARE – WHAT CAN GO WRONG



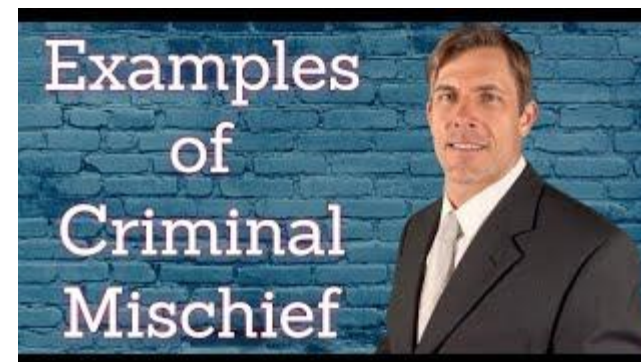
# INFORMATION LEAKAGE

## MALWARE – WHAT CAN GO WRONG



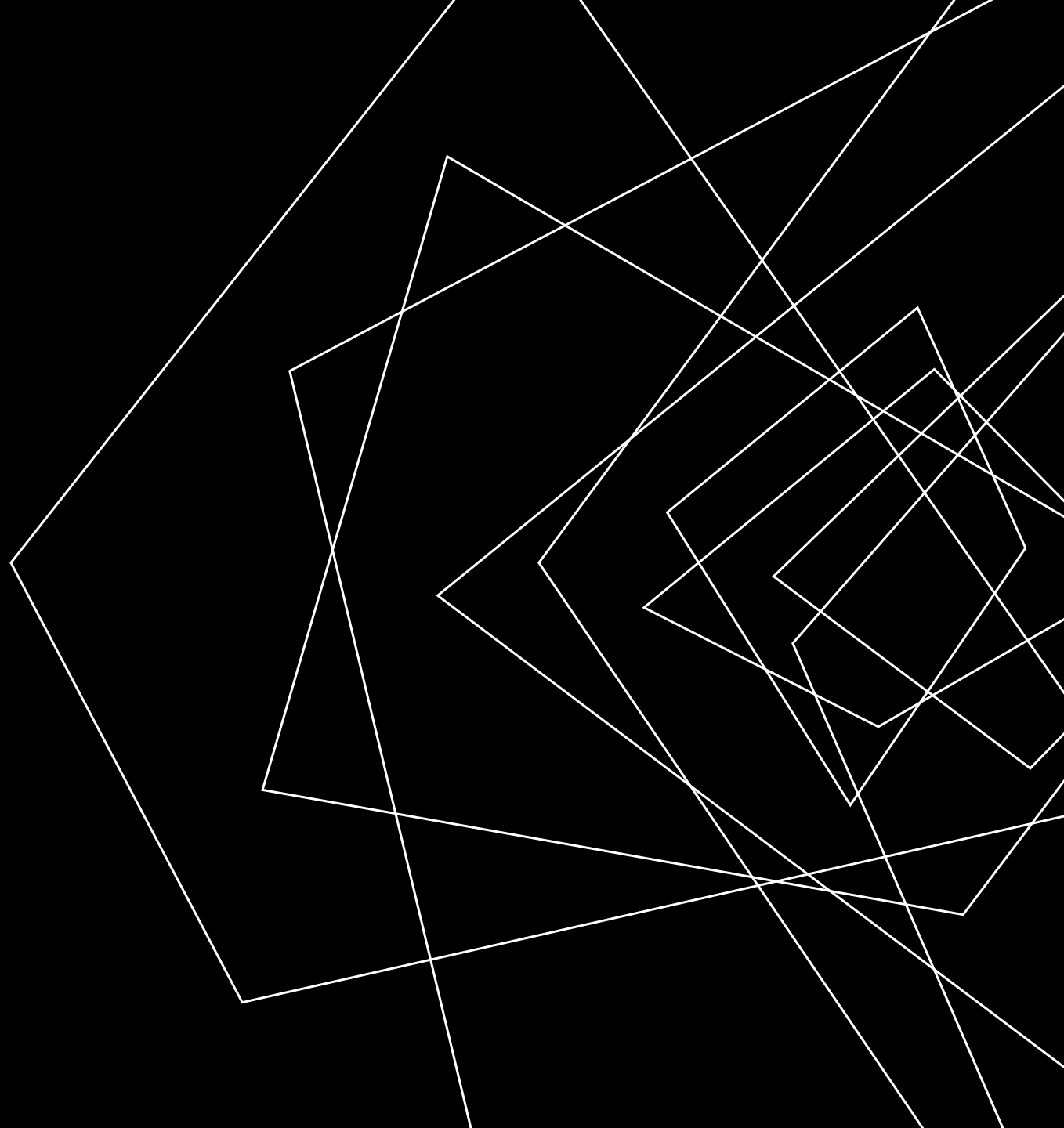
# PROGRAM MISBEHAVIOR

MALWARE – WHAT CAN GO WRONG



# LECTURE OUTLINE

- Terms/Concepts
- What can go wrong?
- How incidents happen



# HOW DO “BAD” PROGRAMS RUN?

## MALWARE – TERMS AND CONCEPTS

### REACTIVE CONCERNS

- Social engineering
- “Flaws” in system installation policies

← other programs

### PROACTIVE CONCERNS

- The program accidentally does damage
- The program contains a vulnerability

← own programs



# HOW DO “BAD” PROGRAMS RUN?

## MALWARE – TERMS AND CONCEPTS

### REACTIVE CONCERNS

- Social engineering
- “Flaws” in system installation policies

### PROACTIVE CONCERNS

- The program accidentally does damage
- The program contains a vulnerability



*We're concerned about all these threats*

# REMOTE CODE EXECUTION (RCE)

MALWARE - TERMS AND CONCEPTS

```
print "enter your name" ; alert();  
var = name;  
exec("print" + name);
```

# DRIVE BY DOWNLOADS

## MALWARE – TERMS AND CONCEPTS

# BEHAVIORAL OPACITY

MALWARE – TERMS AND CONCEPTS

**WRAP-UP**

