EXERCISE #15

SIDE CHANNEL REVIEW

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

Provide an instance of a function with a sensitive argument v and leaks a bit of v via a timing side channel

```
void f(int v){
    printf("enter\n");
    if( v%2 ==1) {

        sleep(100000);

    }
    printf("hello\n");
    return;

}
```

ADMINISTRIVIA AND ANNOUNCEMENTS

Paper review due Sunday at 11:59 PM

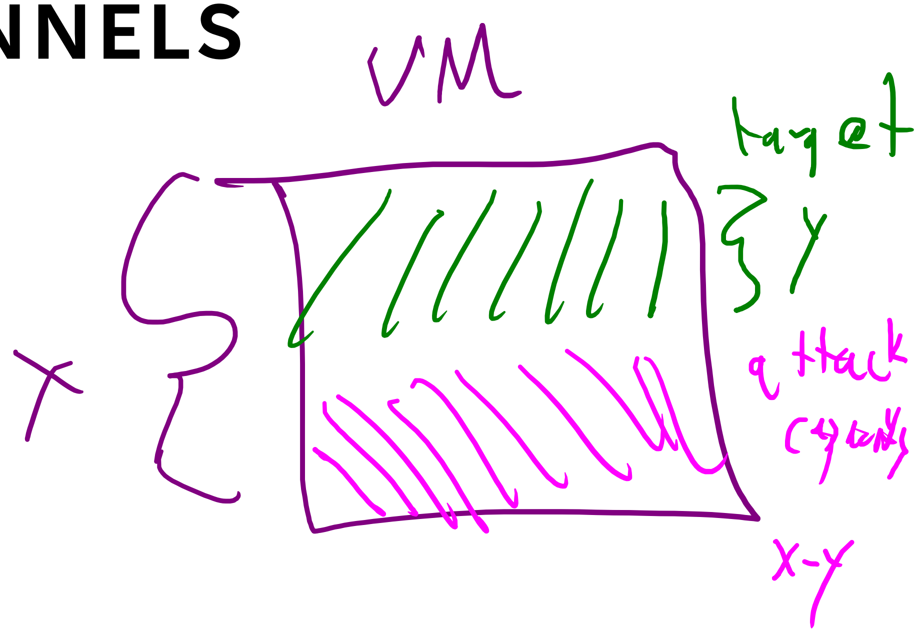
CLASS PROGRESS

SHOWING SOME APPLICATIONS OF STATIC DATAFLOW

- DESCRIBED A PARTICULAR TYPE OF EVASION AGAINST EXPLICIT DATAFLOW: SIDE CHANNELS
- BEGAN TO CONSIDER WHAT WE COULD DO ABOUT IT

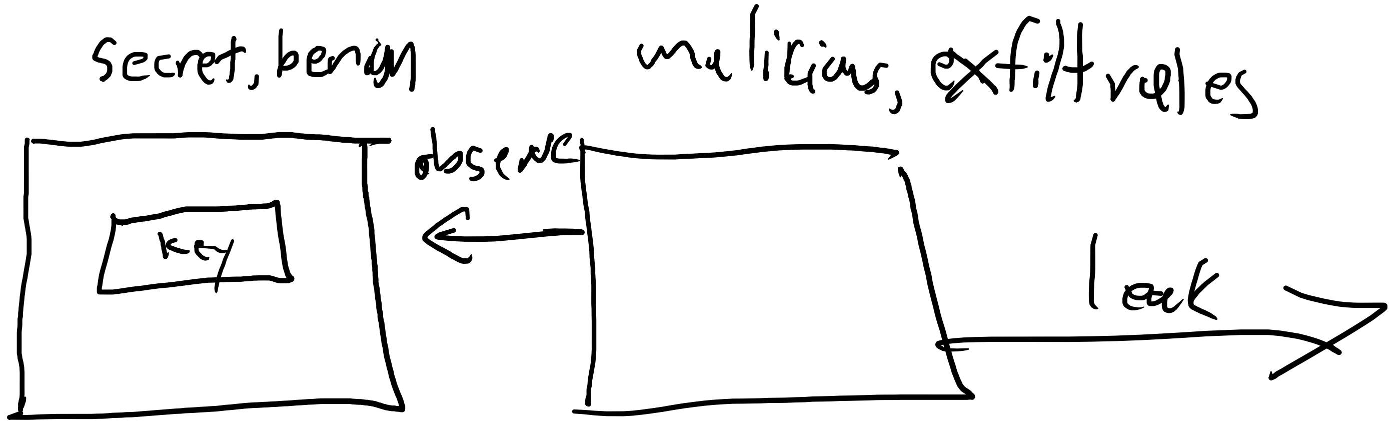
LAST TIME: SIDE CHANNELS

REVIEW: LAST LECTURE



UNDETECTABLE VIA (TYPICAL) STATIC DATAFLOW

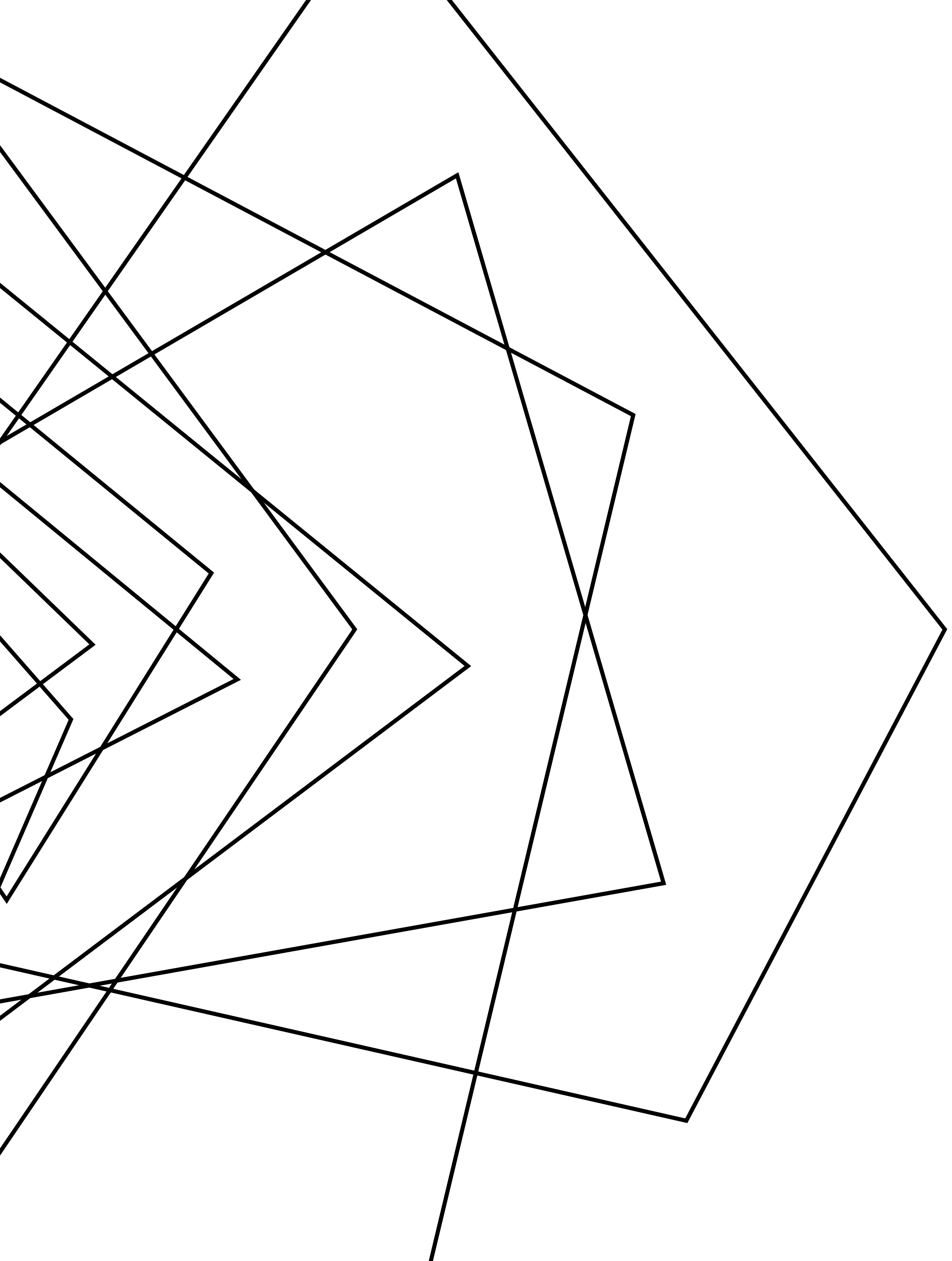
- General side-channel: using a predictable phenomenon outside of the semantics of the program
- Covert channel: special instance of a side channel that is used intentionally by the program



REFERENCE MONITORS

EECS 677: Software Security Evaluation

Drew Davidson



OVERVIEW

PREVENTING BAD STUFF FROM HAPPENING IN A PROGRAM

LECTURE OUTLINE

- Overview
- Details
- Instances

LIMITATIONS OF ANALYSIS
REFERENCE MONITORS: OVERVIEW

SO FAR, OUR FOCUS HAS BEEN LARGELY ON DETECTING UNDESIRABLE BEHAVIOR

- That's valuable!
  - Ask developers to correct their own mistakes
  - Empower users to forgo running bad software

LIMITATIONS OF ANALYSIS

REFERENCE MONITORS: OVERVIEW

Detection might not be enough

- May be in a position where we can't run the analysis

Static Analysis

- False positives
- Scalability issues

Dynamic Analysis

- False negatives
- Run time issues

A HANDS-ON ALTERNATIVE

REFERENCE MONITORS: OVERVIEW

KEEP BAD THINGS FROM HAPPENING DURING SYSTEM EXECUTION

- Requires some sort of specification for “bad things”
- Requires some sort of preventative capabilities

PREVENTATIVE CAPABILITIES

REFERENCE MONITORS: OVERVIEW

SIMPLE FORM

Kill the program

DATAFLOW FORM

Sanitize the data

THE BIG IDEA
REFERENCE MONITORS: OVERVIEW

KEEP PROGRAMS ON THE "STRAIGHT AND NARROW"

- Articulate a policy for allowed behavior
- Keep a running record of security-relevant behavior
- Prevent a violation of the policy

SAFETY POLICIES

REFERENCE MONITORS: INSTANCES

Execution of a process as a sequence of states

Policy is a predicate on sequence prefix

Policy depends only on the past of a particular execution – once violated, never “unviolates”

Incapable of handling liveness policies

“If this server accepts a SYN, it will eventually send a response”

LECTURE OUTLINE

- Overview
- Details
- Instances

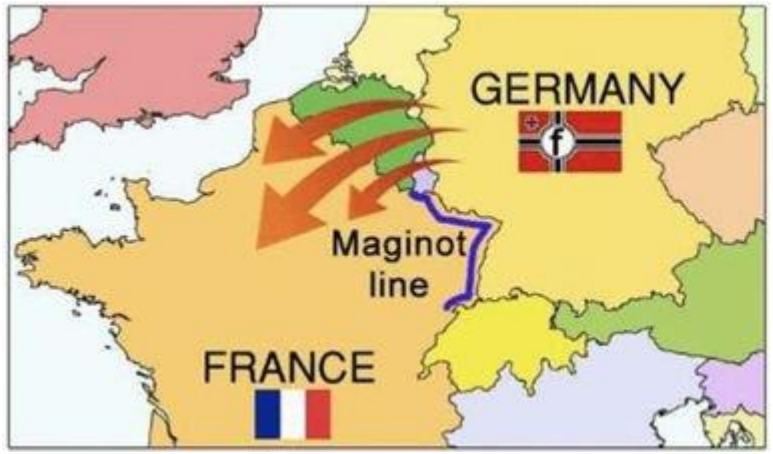
CONSIDER THE REACTIVE ADVERSARY

REFERENCE MONITORS: OVERVIEW

DEFINITION

Reactive Adversary: An adversary with the capability to understand the defense mechanism and an opportunity to avoid it

IF A DEFENSE CAN BE AVOIDED IT HARDLY MATTERS WHAT THE ENFORCEMENT DOES

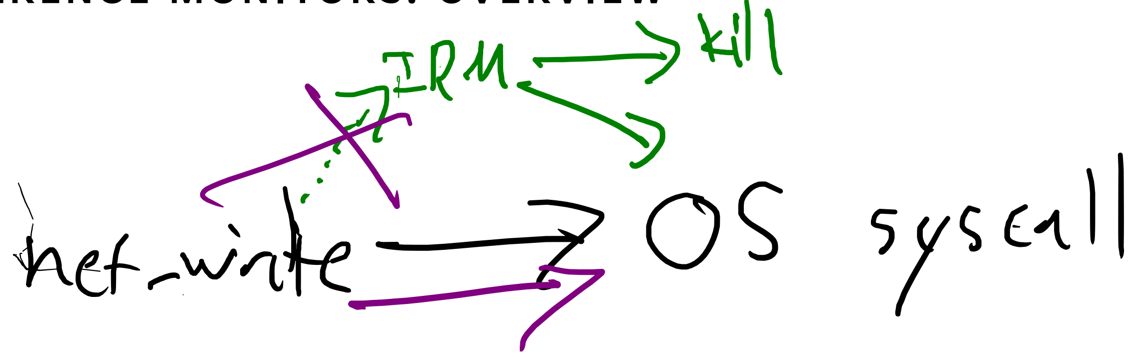


Recall the history of the Maginot Line

SECURITY VS PRECISION

REFERENCE MONITORS: OVERVIEW

PROGRAM PROXIMITY



Close ⟷ Far

Inline reference monitor

Add logging and kill
Stmts into program
(instrument with defense)

External reference monitor

Semantic Gap Bigger

REFERENCE MONITOR DESIGN

REFERENCE MONITORS: INSTANCES

KERNELIZED

Baked into the kernel

- coarse
- secure / hard to avoid

WRAPPER

Specialized execution environment

INLINE

Rewrite the program / hook syscalls

- precise
- less secure / easier to avoid

PROPERTIES WE CARE ABOUT

REFERENCE MONITORS: INSTANCES

Memory Safety

e.g. Programs respect aggregate type sizes, process boundaries, code v data

Type Safety

e.g. Functions and intrinsic operations have arguments that adhere to the type system

Control Flow Safety

e.g. All control transfers are envisioned by the original program

LECTURE OUTLINE

- Overview
- Details
- Instances

OS AS REFERENCE MONITOR

REFERENCE MONITORS: INSTANCES

Collection of running processes and files

Processes are associated with users

Files have ACLs

OS enforces various safety policies

- File access
- Process space write

Same policy for all processes of the same user

SOFTWARE FAULT ISOLATION (SFI)

REFERENCE MONITORS: INSTANCES

ISOLATE PROCESS FAULTS ON SHARED HARDWARE

Each process is a logical fault domain

Ensure all memory references and jump is within the process fault domain

INLINE REFERENCE MONITORS: SASI

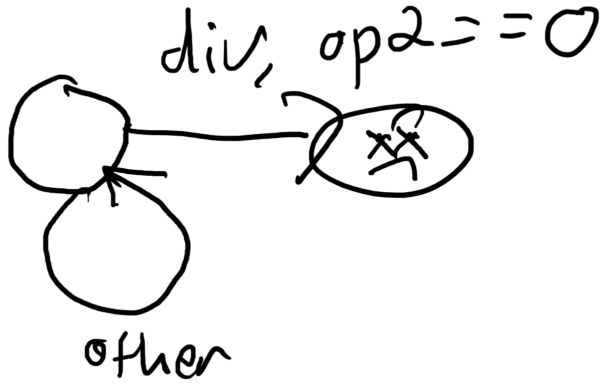
REFERENCE MONITORS: INSTANCES

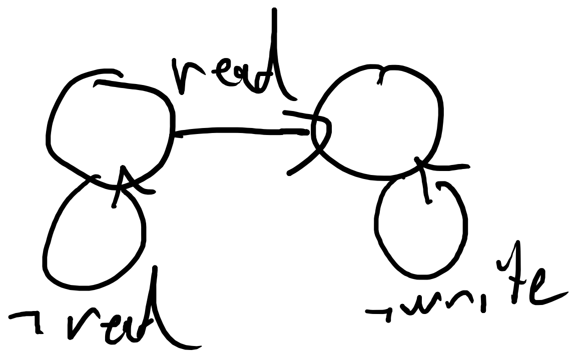
CORNELL PROJECT FOR INLINE POLICY ENFORCEMENT

Change the program to enforce "any" safety policy

Express allowed behavior as an FSM

Examples:
- No division by zero
- No network send after file read





SASI: COST

REFERENCE MONITORS: INSTANCES

Attempts to minimize the number of checks

Looking at every instruction is incredibly expensive

Example: only need to check divide-by-zero before DIV instructions

CONTROL FLOW INTEGRITY: CFI

REFERENCE MONITORS: INSTANCES

Ensure the program control Flow is allowed by the CFG

In a sense, the policy is the control-flow graph

Why would we need to do this?

WRAP-UP