

Introduction to Cybersecurity and Security Mindset

Shuai Wang



香港科技大學

THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Agenda

- Motivation & definition about “Security”
- Some interesting topics in security research
- Mindset

Learn to become a “hacker”, an ethic one 😊

Cybersecurity is Real-World Problem-Driven

- Many (research) topics are indeed driven by security breaches in the real world!
 - That's **one key reason** I decide to work in this field

A Sad (?) Story

aws

Amazon Web Services

Mon May 18 2020
10:51:26

wangshuai

Mon May 18 2020

Translate ▼

Hi there,

Hello,

Was this response helpful? Click here to rate:



aws

Amazon Web Services

Fri Jun 05 2020
15:00:49

GMT+0800 (Hong Kong Standard Time)

Translate ▼

Hello there,

Martin here from AWS.

I'm happy to advise that 100% of the charges for the compromised activity on your account have been waived. Rest assured, you no longer have to worry about the charges.

To avoid similar compromises in the future, please consider the following to help improve the security of your account.

Was this response helpful? Click here to rate:



What's Security?

Confidentiality

Information is secret

Integrity

Information/System is correct

Availability

System is usable

You will again and again come to this part when doing security research...

Typical Topics Covered in Cybersecurity Studies

- **Security basics and principles :**
 - Confidentiality, integrity, availability, attack models
- **Cryptography:**
 - Basic crypto primitives, public key crypto, signatures, authentication, symmetric crypto
- **Software security:**
 - Memory errors, buffer overflow, obfuscation, malware, security testing
- **System & web security:**
 - Authentication, access control, protocols, browser security, side channel attacks
- **Security on emerging platforms:**
 - blockchain; IoT; AI;

Reverse Engineering

- How to break the password protection of a Windows software?

```
call sub_401760
call sub_401400
mov [esp+98h+var_98], offset aPleaseEnterYou ; "Please enter your password\n\n"
call printf
lea eax, [ebp+var_78]
mov [esp+98h+var_94], eax
mov [esp+98h+var_98], offset aS ; "s"
call scanf
lea eax, [ebp+var_78]
mov [esp+98h+var_94], offset aFindneifyoucan ; "FindNeIfYouCan"
mov [esp+98h+var_98], eax
call strcmp
test eax, eax
jnz short loc_401301

loc_401300:
mov [esp+98h+var_98], offset aCongratsCorrec ; "Congrats!! Correct Pass\n\n"
call printf
jmp short loc_401300

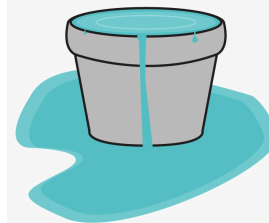
loc_401301:
mov [esp+98h+var_98], offset aWrongPass ; "Wrong Pass\n\n"
call printf
```

```
D:\temp\binaryCrack\Debug\bin
Please enter the password
dfrgdrtrgrtdfg
Wrong password
```

How to find vulnerabilities in software?

```
#include <stdio.h>
#define MAX_IP_LENGTH 15
int main(void) {
    char file_name[] = "ip.txt";
    FILE *fp;
    fp = fopen(file_name, "r");
    char ch;
    int counter = 0;
    char buf[MAX_IP_LENGTH];
    while((ch = fgetc(fp)) != EOF) {
        buf[counter++] = ch;
    }
}
```

Buffer overflow if "ip.txt" has more than 15 bytes.



**BUFFER
OVERFLOW
ATTACKS**

Vulnerability Finding Today

- Security bugs can bring \$500-\$100,000 on the open market
- Good bug finders make \$180-\$250/hr consulting
- Few companies can find good people, many don't even realize this is possible.
 - Google: Team Zero; Tencent: Keen Lab; ...
- Still largely a black art



Automatic Vulnerability Detection



Find a needle in a haystack

OSS-Fuzz - continuous fuzzing for open source software.

 [google.github.io/oss-fuzz](https://github.com/google/oss-fuzz)

Fuzz Testing

CodeQL

Discover vulnerabilities across a codebase with CodeQL, our industry-leading semantic code analysis engine. CodeQL lets you query code as though it were data. Write a query to find all variants of a vulnerability, eradicating it forever. Then share your query to help others do the same.

Information flow analysis

Formal Verification

- Formal verification can (ideally) **completely eliminate vulnerabilities**.
 - **Mathematically prove the absence of bugs.**
- How to I know the **insertion sort** will always return a sorted list?

```
# Python program for implementation of Insertion Sort

# Function to do insertion sort
def insertionSort(arr):

    # Traverse through 1 to len(arr)
    for i in range(1, len(arr)):

        key = arr[i]

        # Move elements of arr[0..i-1], that are
        # greater than key, to one position ahead
        # of their current position
        j = i-1
        while j >=0 and key < arr[j] :
            arr[j+1] = arr[j]
            j -= 1
        arr[j+1] = key

# Driver code to test above
arr = [12, 11, 13, 5, 6]
insertionSort(arr)
print ("Sorted array is:")
for i in range(len(arr)):
    print ("%d" %arr[i])
```

Formal Verification

- You can **prove** it, as how you prove some Euclidean geometry properties.

```
# Python program for implementation of Insertion Sort
```

```
# Function to do insertion sort
```

```
def insertionSort(arr):
```

```
    # Traverse through 1 to len(arr)
```

```
    for i in range(1, len(arr)):
```

```
        key = arr[i]
```

```
        # Move elements of arr[0..i-1], that are  
        # greater than key, to one position ahead  
        # of their current position
```

```
        j = i-1
```

```
        while j >=0 and key < arr[j] :
```

```
            arr[j+1] = arr[j]
```

```
            j -= 1
```

```
        arr[j+1] = key
```

```
# Driver code to test above
```

```
arr = [12, 11, 13, 5, 6]
```

```
insertionSort(arr)
```

```
print ("Sorted array is:")
```

```
for i in range(len(arr)):
```

```
    print ("%d" %arr[i])
```

Proof

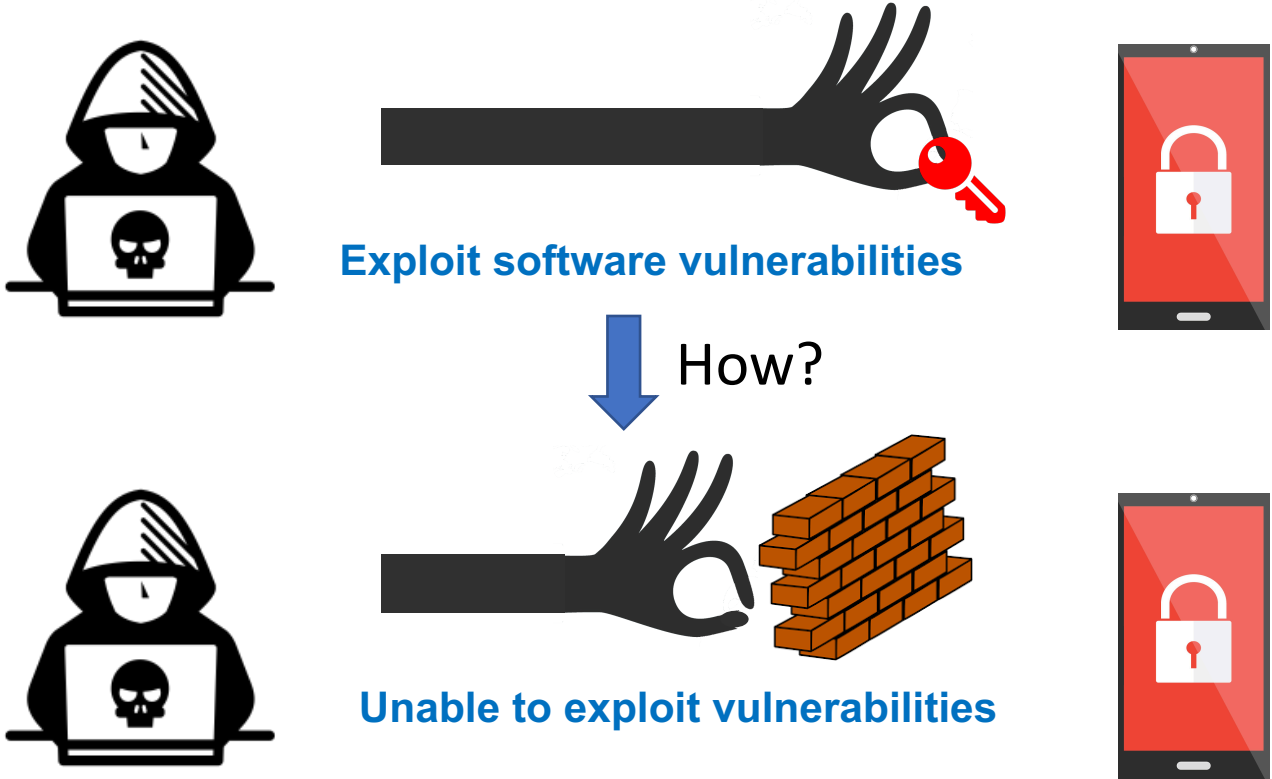


```
1 Require Import List.
2 Require Import ZArith.
3 Require Import Bool.
4 Require Import sort_lectures.
5
6
7
8
9
10 (* Inserts an element in to a sorted list *)
11 Function insert (x : Z) (list : list Z) :=
12   match list with
13   | nil => x :: nil
14   | hd :: tl => if !x <= ? hd then x :: hd :: tl else hd :: insert x tl
15   end.
16
17 (* Insertion sort *)
18 Function insertionSort (list : list Z) :=
19   match list with
20   | nil => nil
21   | hd :: tl => insert hd (insertionSort tl)
22   end.
23
24 (* Insertion of an element keeps the list sorted *)
25 Lemma insert_keeps_list_sorted (x : Z) (list : list Z) :
26   urejen list -> urejen (insert x list).
27 Proof.
28   intros.
29   induction list.
30   - auto.
31   - simpl.
32     case_eq (x <= ? a)hZ.
33     + intros.
34       simpl.
35       firstorder.
36       new apply Z.leb_le.
37     + intros.
38       apply Z.leb_gt in H0.
39       simpl.
40       destruct list in simpl.
41       + firstorder.
42       + firstorder.
43       case_eq (x <= ? a)hZ.
44       + intros.
45         firstorder.
46         new apply Z.leb_le.
47       + intros.
48         replace l1 :: insert x list with (insert x (z :: list)).
49         assumption.
50       + intros.
51         case_eq (x <= ? a)hZ.
52         + intros.
53           absurd (x <= ? a)hZ = false; auto.
54         + now rewrite <- not_false_iff_true in H0.
55           auto.
56   Qed.
57
58
59
60 (* InsertionSort always returns sorted list *)
61 Lemma returns_sorted_List :
62   forall l : list Z, urejen (insertionSort l).
63 Proof.
64   intros.
65   induction l.
66   - now simpl.
67   - simpl.
68     now apply insert_keeps_list_sorted.
69   Qed.
70
71
72
73 (* Number of occurrences of x increases if we insert another x into a list *)
74 Lemma occurrences_of_x (x : Z) (l : list Z) :
75   forall y (insert x l) = S (pojavu x l).
76 Proof.
77   induction l.
78   - now rewrite Z.eq_refl.
79   - simpl.
80     case_eq (x <= ? a)hZ.
81     + intros.
82       case_eq (x = ? a)hZ.
83       + intros.
84         replace a with x.
85         simpl.
86         replace (x = ? a)hZ with true.
87         auto.
88         now rewrite Z.eq_refl.
89       + auto.
90         now apply Z.eq_eq.
91     + intros.
92       simpl.
93       replace (x = ? a)hZ with false.
94       replace (x = ? a)hZ with true.
95       auto.
96     + intros.
97       now rewrite Z.eq_refl.
98   + intros.
99     case_eq (x = ? a)hZ.
100    + intros.
101      replace a with x.
102      simpl.
103      replace (x = ? a)hZ with true.
104      auto.
105      now rewrite Z.eq_refl.
106    + intros.
107      now apply Z.eq_eq.
108    + intros.
109      simpl.
110      now replace (x = ? a)hZ with false.
111    + intros.
112
113 (* Number of occurrences of x does not change if we insert a different element into *)
114 Lemma occurrences_of_y (x y : Z) (l : list Z) :
115   ((x = ? y)hZ = false) -> pojavu x (insert y l) = pojavu x l.
116 Proof.
117   induction l.
118   - simpl.
119   - now replace (x = ? y)hZ with false.
120   - simpl.
121     case_eq (y <= ? a)hZ.
122     + intros.
123       case_eq (x = ? a)hZ.
124     + intros.
125       + intros.
```

Computer will check the correctness

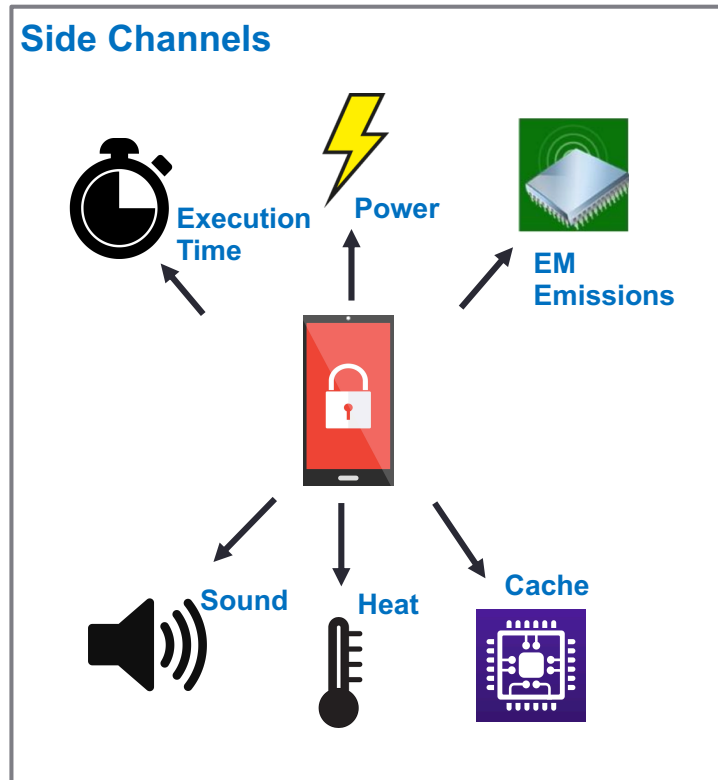
Haven't finished yet..

Side Channel Attacks



Side Channel Attacks

- De-facto exploitations in Cybersecurity

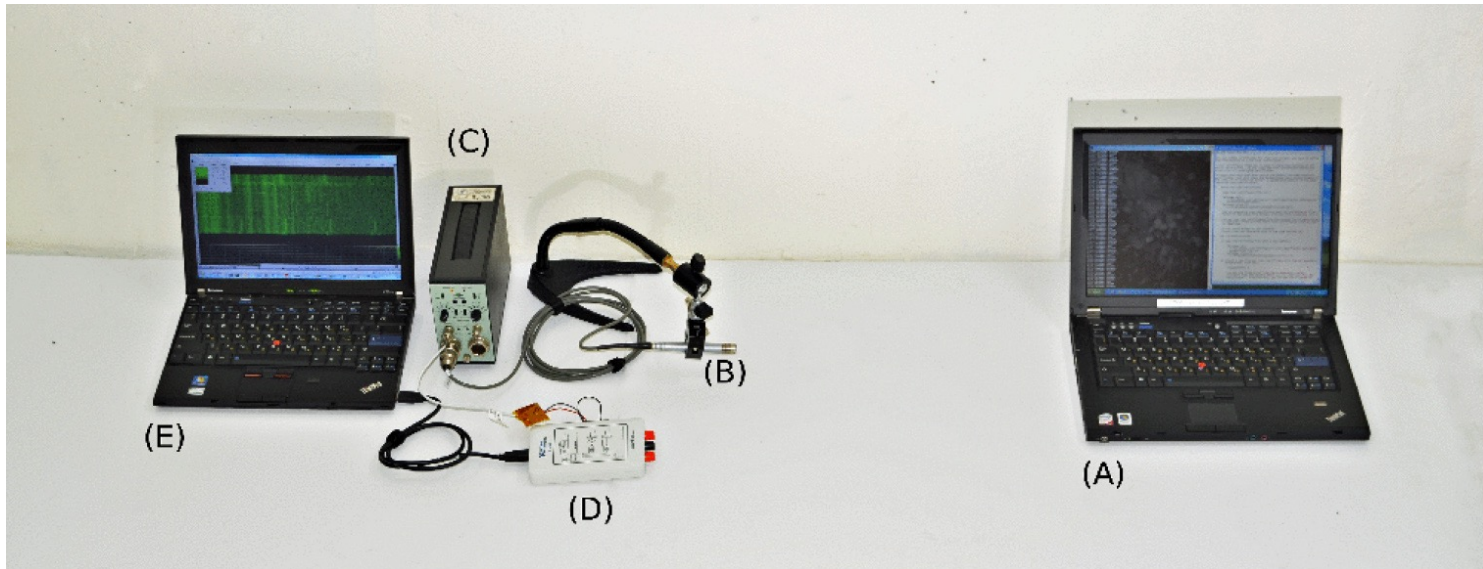


Infer secrets via **secret-dependent** physical information.



Side Channel Attacks

- Infer your secrets (password; private key) via acoustic side channel attack

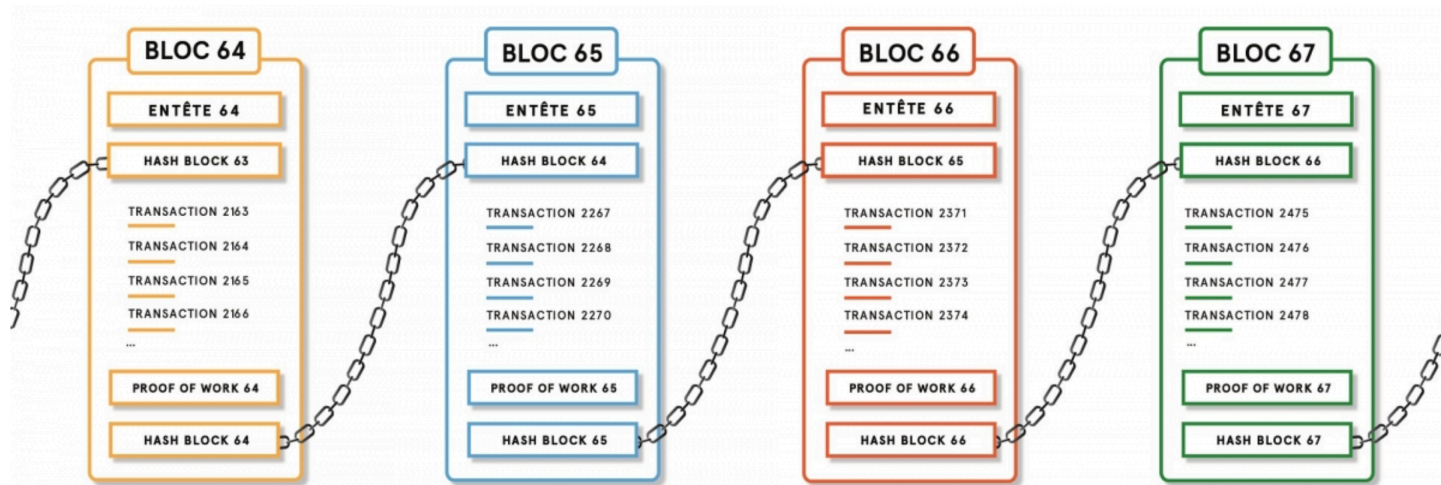


Attacker's

Victim's

Blockchain

The best real-world crypto application and have made many millionaires?



Bitcoin

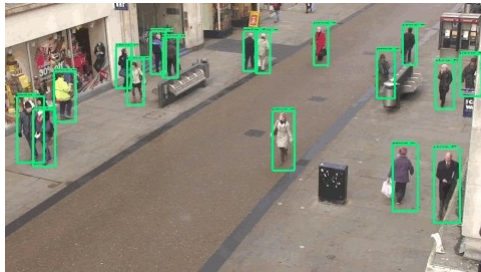
- Unregulated digital currency
- Bitcoin transactions are stored on Blockchain
- Each anonymous address on the blockchain acted as a simple bank account.

Ethereum

- Unregulated digital currency and **computing system**
- **Smart contracts:** programs executed on the blockchain
- Each anonymous address on the blockchain could be a user or a **smart contract**.

Artificial Intelligence

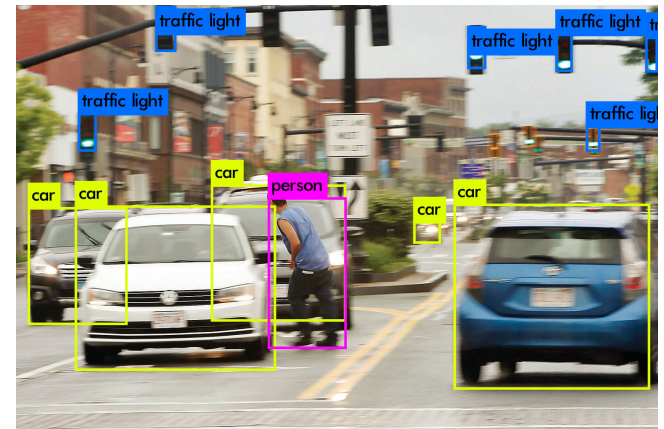
- AI techniques have been used for security purposes.



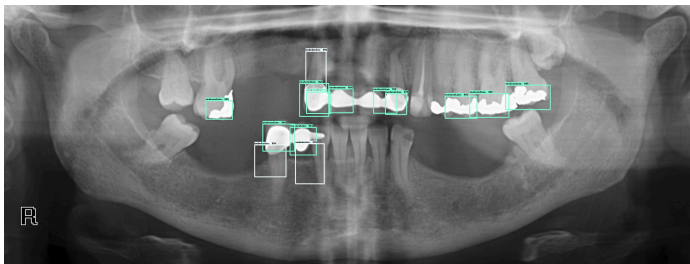
Surveillance Camera



Surveillance Camera



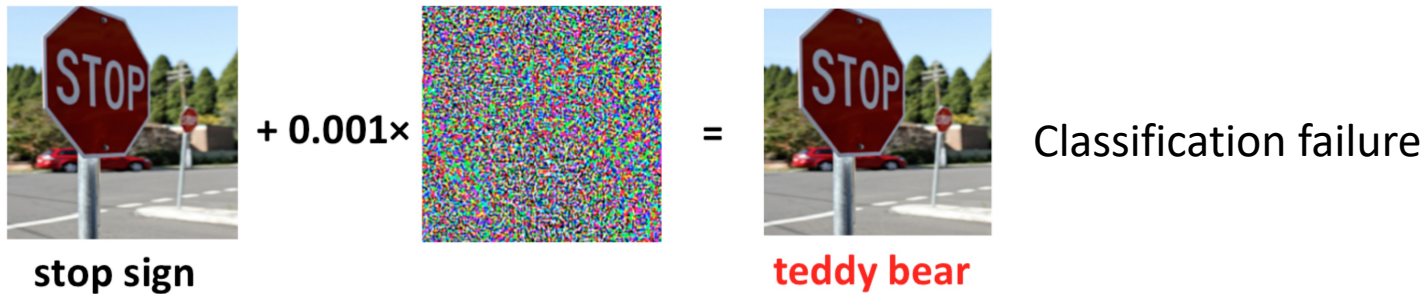
Auto-Driving Systems



Medical Image Processing

Artificial Intelligence

- Adversarial attacks are popular...



Object detection failure

We will talk more cases on AI security.

The Security Mindset

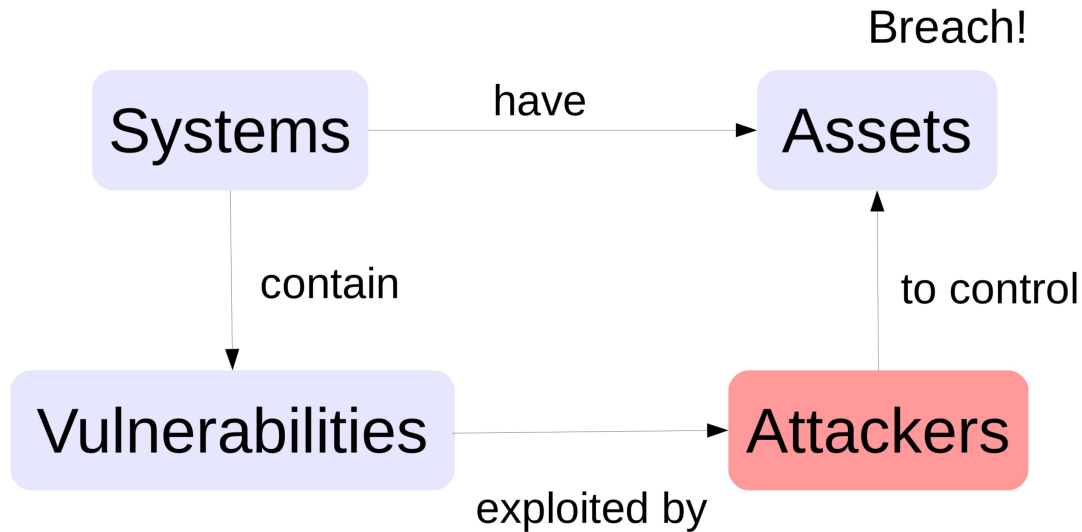


Attacker vs. defender

The Security Mindset

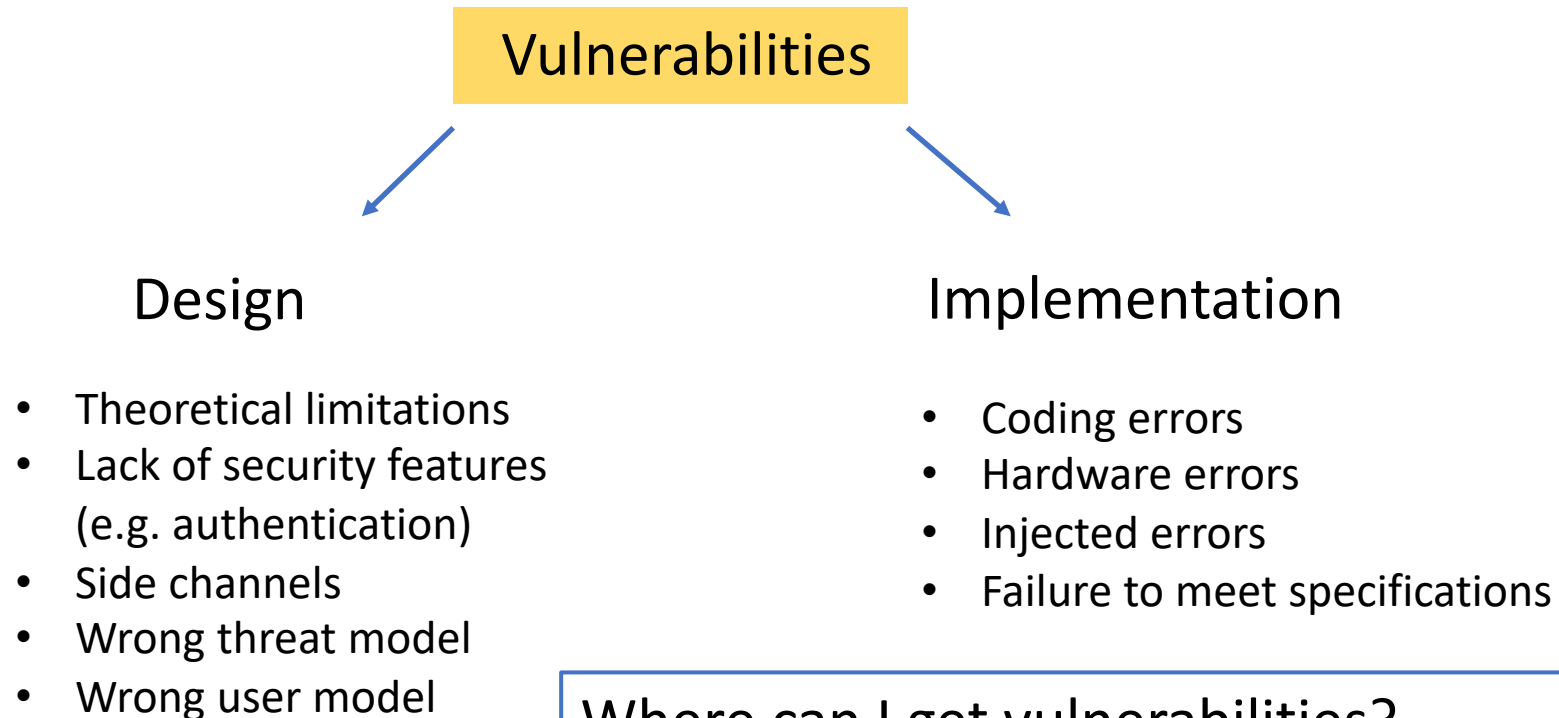
- Think like a cyber attacker
 - Understand **techniques** and **opportunities** for exploiting security. → next two slides
- Think like a cyber defender
 - Know yourself: **security policy**
 - Know yourself: **risk assessment**
 - Know your enemy: **threat model**
 - Benefits vs. costs:
 - Some security defenses are just too expensive

Think Like an Attacker



Think Like an Attacker

Where do vulnerabilities come from?



Where can I get vulnerabilities?

- Find unknown vulnerabilities → hard
- Buy zero-day vulnerabilities → well, you can do that..?
- Reuse known vulnerabilities → easy?

But Why Good Citizens Need to Know How to Attack?

To understand this, think about why biologists would study (unknown) virus...



White hat wizards!

- Identify vulnerabilities so they can be fixed.
- Learn about unknown threats.
- Help vendors to build more secure systems.
- ~~And get lots of bonus~~ from vendors

Think Like a Defender

- Security policy
 - What **property** we are trying to enforce?
 - E.g., **password** can only be stored within my phone.
 - E.g., data pointers in your C code can only access certain memory region.
 - Could be **difficult** to even define the policy/specification
- Risk assessment
 - Identify assets (e.g., network, servers, applications, data centers, etc.) within the organization.
 - Asset criticality.
 - Measure the risk ranking for assets and prioritize them for assessment.

Think Like a Defender

- **Threat model**
 - Who are the **attackers**?
 - What kind of capability they have?
 - What kind of information/data they try to steal?

Think Like a Defender

- Threat model for a (simplified) cloud computing platform
 - Attacker; capability; assets

Think Like a Defender

- **Threat model**

- Who are the **attackers**?
 - Service provider, and other users
- What kind of capability they have?
 - Service provider can control anything
 - Attackers on the cloud VM can share the same hardware with you
 - Common threat model for side channels
- What kind of assets they try to steal?
 - Anything valuable!

Think Like a Defender

- Costs vs. benefits?
 - For example, to protect an OS kernel from being exploited, you can have two options:
 - Online monitoring:
 - **easy** to do.
 - **slow down** the performance
 - Offline formal verification:
 - **very difficult** to conduct for commercial OS.
 - But **no penalty** for online performance.
- Saltzer and Schroeder's Principles of Secure Design
 - A series of design principles for secure systems
 - Extensions for reading after the class.
 - Some of the rules may **not** be applicable nowadays.

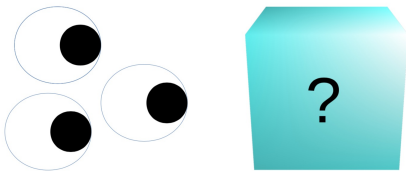
Saltzer and Schroeder's Principles of Secure Design

- 1) Open Design vs. Obscure Design

The system's design should be openly available to everyone.

"Given enough eyeballs, all bugs are shallow"

-- Linus Torvalds



"The eyeballs weren't looking"

Saltzer and Schroeder's Principles of Secure Design

- 2) Economy of Mechanism

The system should be simple enough to understand and analyze.

Helpful for security analysis:

- Debugging/code audit
- Static/dynamic analysis
- Formal verification

Clean interfaces between modules, avoid global state, etc.

Saltzer and Schroeder's Principles of Secure Design

- 3) Least Privilege

A subject should only be given the minimum necessary privileges for completing its task.

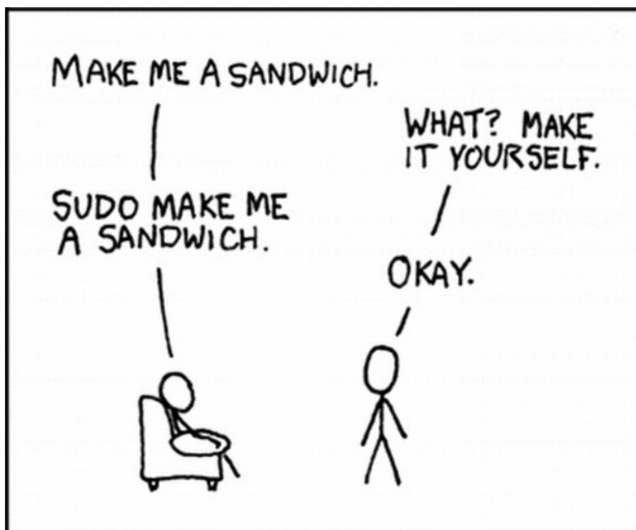


Figure out exactly what capabilities a program requires in order to run, and grant exactly those

- This is not easy. One approach is to start with granting **none**, and see where errors occur.

Summary

- The **endless arms race** between cyber attackers and defenders lead to many interesting problems
 - For doing research & engineering
- Be a **happy** and **ethic** hacker!
 - Otherwise, you (and your teacher) might run into trouble ...