Chapter 1: Intro to "*Intro to CyberSecurity*"

Xianghang Mi

中国科学技术大学
University of Science and Technology of China

Outline

- Course FAQs (frequently asked questions)
- Motivations of CyberSecurity (Why we need cybersecurity)
- Overview of This Course

Outline

- Course FAQs
- Motivations of CyberSecurity
- Overview of This Course

Course Staff: Instructor

- 糜相行(Xianghang Mi) https://xianghang.me
- 计算机学院，特任教授
  - Assistant Professor, CSE at SUNY Buffalo
  - Research Scientist at Meta (Formerly Facebook)
- Research Interests: Network Security, IoT security, Cybercrime, AI for Security
- Email: xmi@ustc.edu.cn
- Reach out to me if you are interested in security research

Time and Location

- 学时：40理论 + 40实验
- Lectures
  - 9:45 AM – 12:10 PM, Thursday
    - 09:45~10:30，10:35~11:20，11:25~12:10
  - 西区3C202 (西区第三教学楼C楼202)
- Q&A
  - Office hours: 1:30pm ~ 4:30pm, Thursday
    - Virtual access: Blackboard-> Questions & Answers
    - Physical access: 西区科技实验西楼1404
  - Online Discussion： 飞书

Course Staff: Teaching Assistants

- Yekai Li
  - Email: yekaili@mail.ustc.edu.cn
- Ronghong Huang
  - Email: ronghonghuang@mail.ustc.edu.cn

Course Objectives

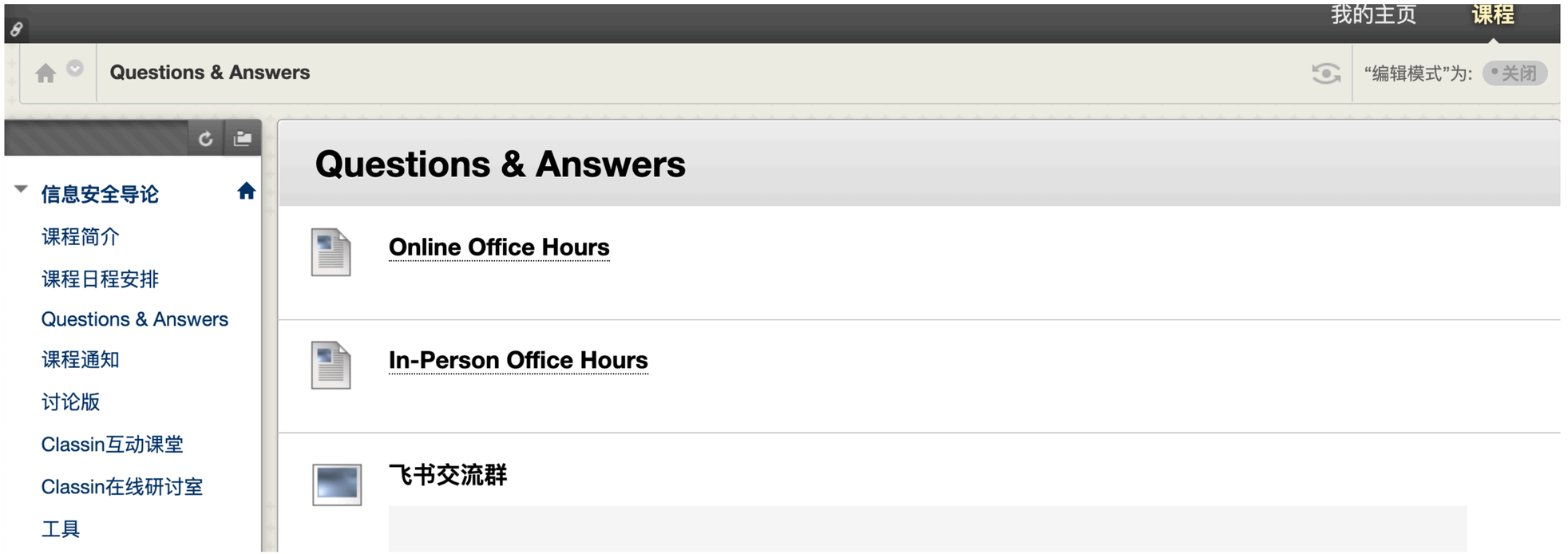
- Learn the fundamentals of cybersecurity
  - Problems, methodologies, and tools
  - Research frontiers
  - Challenges and future directions
- Practice various security tools and do hacking
- Have fun!

What you Will not Learn From This Course

- A lot!!!
- Why?
  - There are many things I don’t know
  - The field is enormous
  - Cybersecurity evolves super-fast

Important URLs

- Course website (assignments, notes, readings, announcements)
  - https://www.bb.ustc.edu.cn/



Important URLs

中科大信息安全导论2023

企业邀请码

XLJE JMDK

Course Material

- Recommended Materials
  - CyBok1.1, https://www.cybok.org/knowledgebase1_1/
  - Computer Security by David Wagner: https://textbook.cs161.org/
  - Computer Security: Principles and Practice, by William Stallings and Lawrie Brown, 4th version in 2018, 3th version in 2014
  - Security Engineering: A Guide to Building Dependable Distributed Systems, 3th version, 2020
  - A Graduate Course in Applied Cryptography, D. Boneh and V. Shoup http://toc.cryptobook.us/#toc

Course Workload

- **Start early, and try to enjoy it!!**
  - 5 security Labs/Homeworks
  - 1 programming assignment
  - 1 research project
  - 1 final exam
- You can work as a team of up to 5 members for **all tasks except for the final**
  - You may **work as different teams** for separate assignments

Grading (Tentative)

- Final 30%
- Security Labs/HWs 30%
- 1 Programming Assignments 20%
- 1 Research Assignment 20%
- Bonus Points:
  - Positive/negative course participation ±5%
  - High-impact and novel security findings 10%

Lectures

- Understand material in class (**ask questions!**)
  - Read relevant chapters，and the recommended reading materials
- Will also post extra readings for better understanding and additional concepts

Assignment Submission

- Done individually or (highly recommended) in teams of up to 5 students
  - One submission per team
  - All team members get the same grade
- Late Policy: 0.9^X if X <=7 else 0
  - late submissions of X days: scored out of 100 * 0.9^X points instead of 100
  - For example, X=1, the scoring scale will be 90 instead of 100
- Handwritten work and legibility
  - Written assignments should be neat and legible
  - The TAs **reserve the right to assign a zero grade** to illegible handwritten homeworks/reports

The Final Exam

- No make up exam will be given without a valid excuse
- **No lame excuses please!!!**
  - I have to go home/start an internship early, can I take the exam earlier?
  - I had a fight with my girlfriend/boyfriend
  - I need a B+ to graduate this semester
  - I have a job interview on/before the exam/project due day
  - ….

The Final Exam

Time: 10:45AM - 12:10PM, Thursday, June 8, 2023

- Policies
  - This exam is open-book, but seeking help from any other individuals is NOT ALLOWED.
  - You must not give or receive any unauthorized help on this exam. All work must be done on your own
- Handwritten answering should be human readable.

Academic Integrity

- **Your honor is most valuable, don't take risk losing it!**
- **No tolerance on cheating/plagiarism!!!**
  - All academic integrity violation cases will be reported to the department, school, and university, and recorded
  - **0** on the particular assignment/exam on 1st offense in **any course**
  - Fail the course on 2nd offense
- Team members are equally responsible!
- Students who share the work with others are as responsible for academic dishonesty as those receiving the material
- Again, no lame excuses!
  - I did not know/I was not sure/I forgot

Academic Integrity

Your honor is most valuable, don’t take risk losing it!

Academic Integrity

No tolerance on cheating/plagiarism

More on Academic Integrity

- Group study/discussions are encouraged but submission must be your own work
- Homeworks/Labs
  - No collaboration among individuals/teams
  - Use of reference material is allowed as long as you explicitly state the references
- Programming Assignments:
  - Discussion of ideas is welcome but **no sharing of code!**
  - Use of code found online is allowed as long as you explicitly state the reference

Class Participation

- Very important!
- Attend classes, participate in discussions, express your opinion, ask questions
- Feel free to share ideas, questions, articles on cybersecurity, etc. with whole class

How to make it interesting? How to do well?

- Participate
- Give suggestions
  - I will consider them seriously
- Do the assigned readings and surf the web to read related things
- Start early on homeworks/labs/programming assignments

Where Do I Ask Questions About

- Lectures, homework?
  - **Feishu**
  - Office hours
  - Email to Xianghang (email)
- Programming assignments, labs?
  - **Feishu**
  - Email to Xianghang (email) or the TAs

The Research Project

- Explore a cybersecurity topic that interests you the most
- Process
  - Pre-Proposal
  - Proposal
  - Mid-term report
  - Final report

The Research Project

- Recommendations **(Not Mandatory)**
  - Write in English and through Latex
  - Latex Templates
    - ACM SIG templates
    - IEEE Conf Template

Outline

- Course FAQs
- Motivations of CyberSecurity
- Overview of This Course

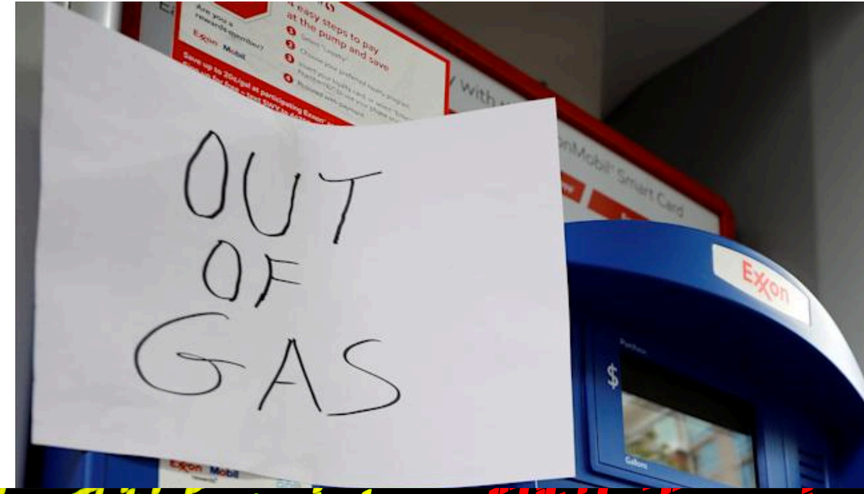
Weak VPN Password

VPN

Colonial Pipeline Company

Customer Billing

Pipeline Operation



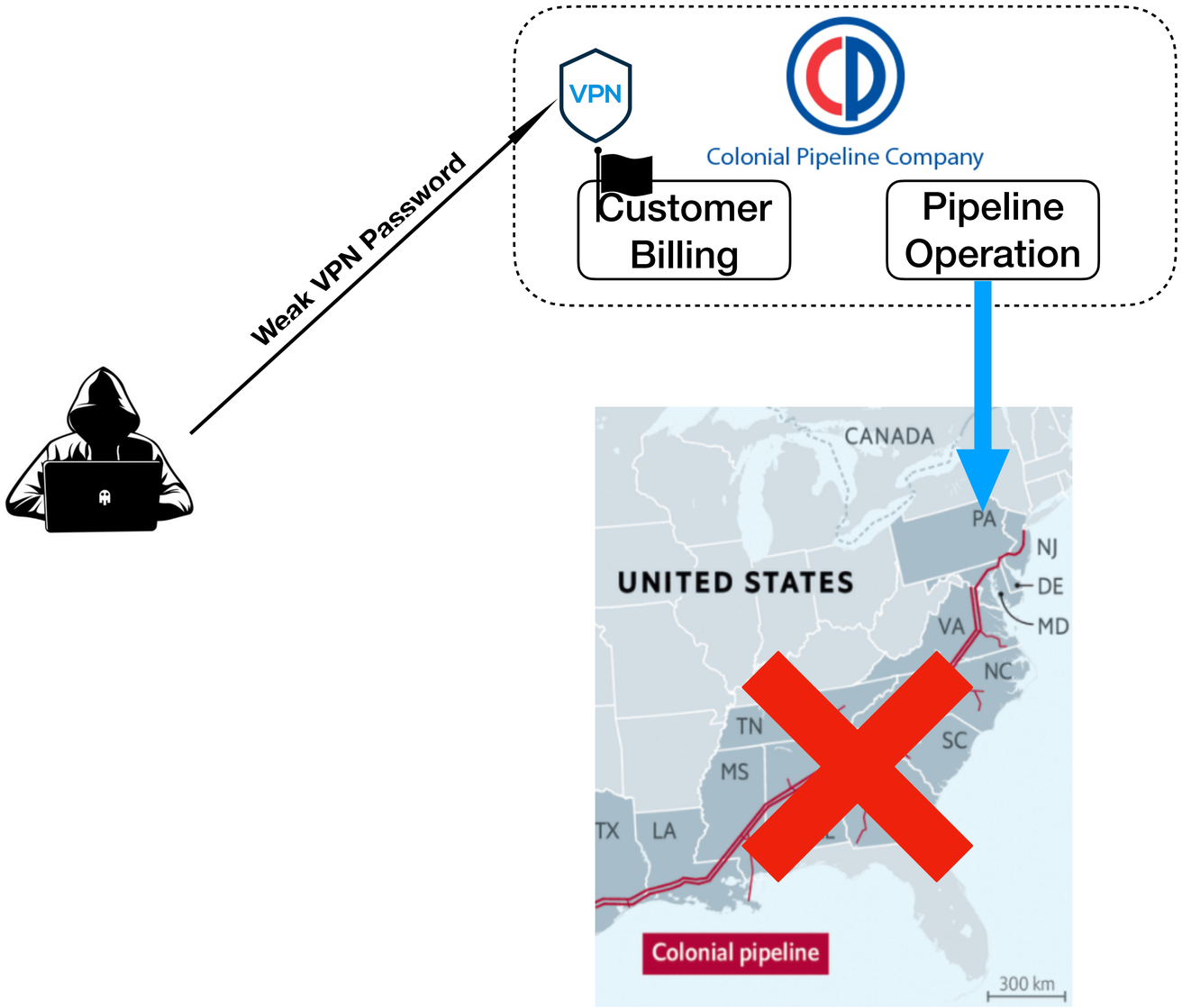
The FBI confirms that the Darkside ransomware is responsible for the compromise of the Colonial Pipeline networks. We continue to work with the company and our government partners on the investigation.

FBI FEDERAL BUREAU OF INVESTIGATION

STATEMENT

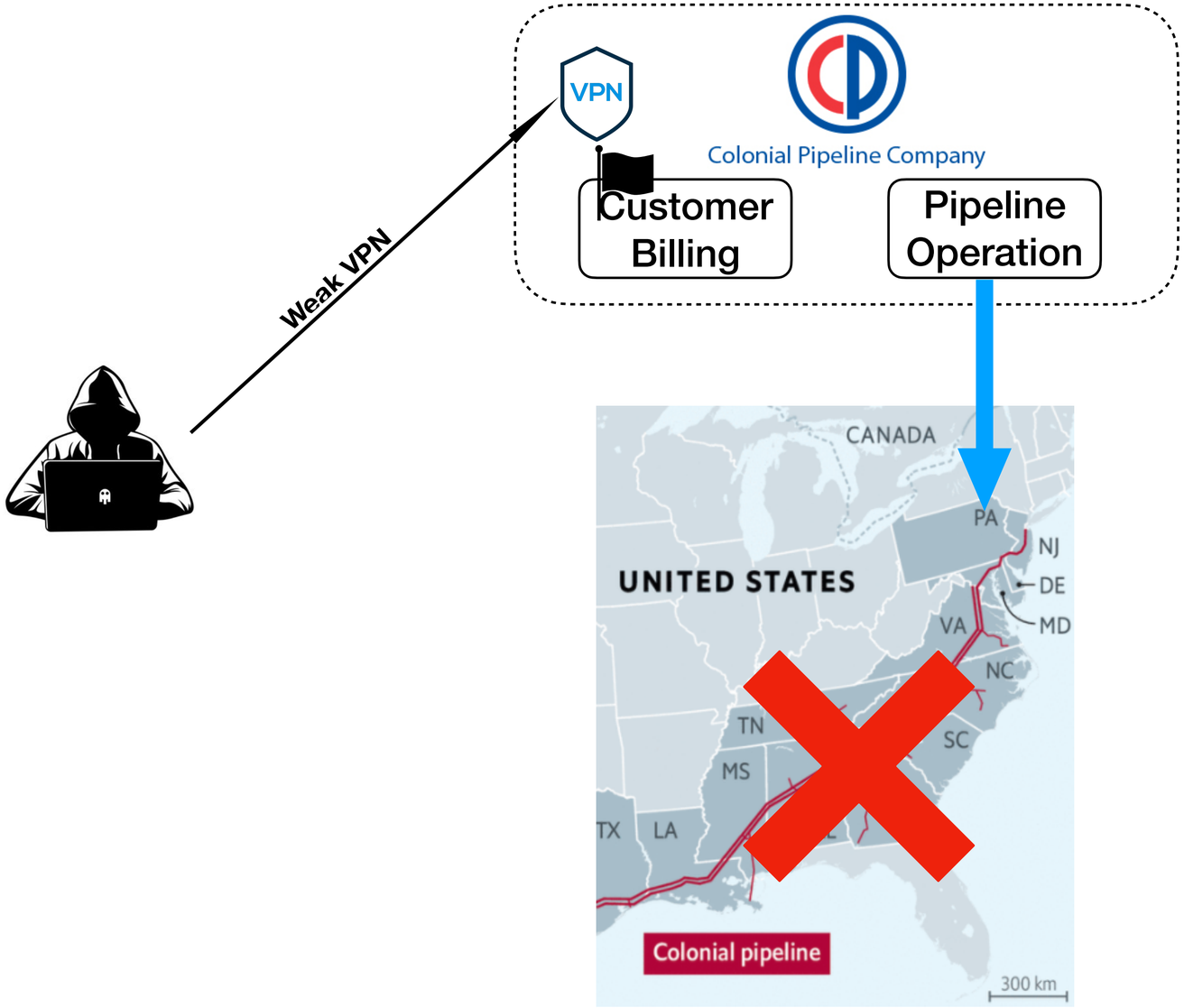
DarkSide Ransomware Creates Emegency State in the U.S.

Colonial Pipeline Hack: timeline



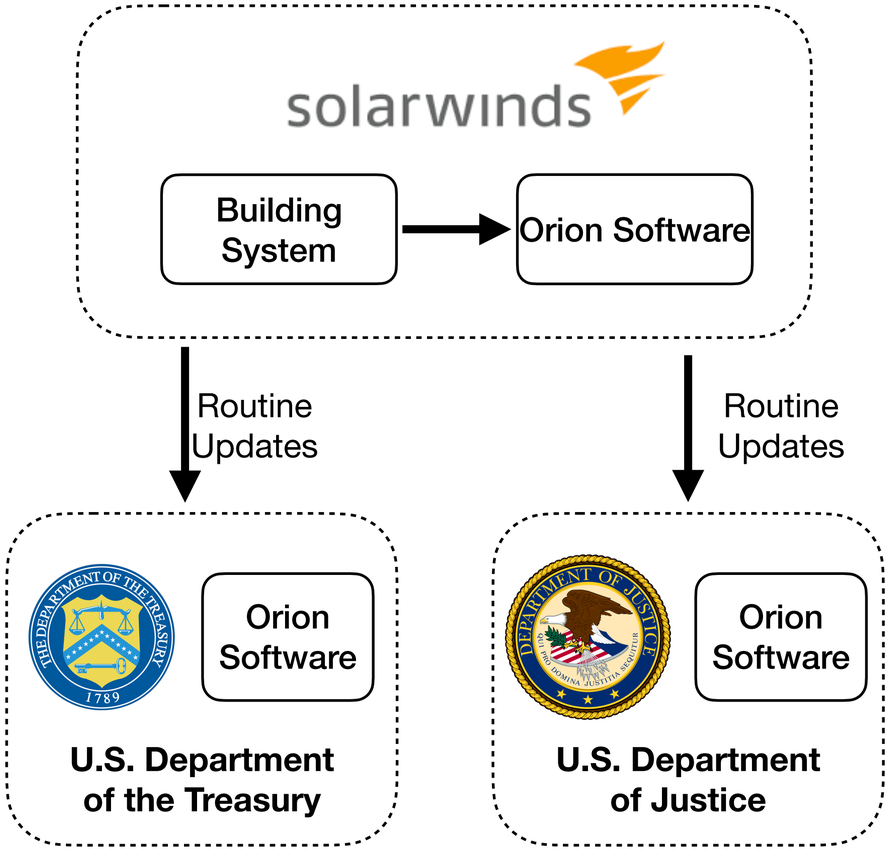
- **April 29, 2021:** the attacker got access to the networks of Colonial Pipeline Co, through a VPN account
- **5 a.m, May 7, 2021:** an employee in Colonial's control room saw a ransom note demanding cryptocurrency appear on a computer
- **6:10 a.m, May 7 2021:** the pipeline has been shut down, which was the first time Colonial had shut down the entirety of its gasoline pipeline system in its 57-year history
- Colonial paid the hackers, a $4.4 million ransom shortly within several hours after the hack, and The hackers also stole nearly 100 gigabytes of data from Colonial Pipeline
- **5 p.m., May 12 2021:** the restart of pipeline operations began, ending a six-day shutdown.

Colonial Pipeline Hack: summary



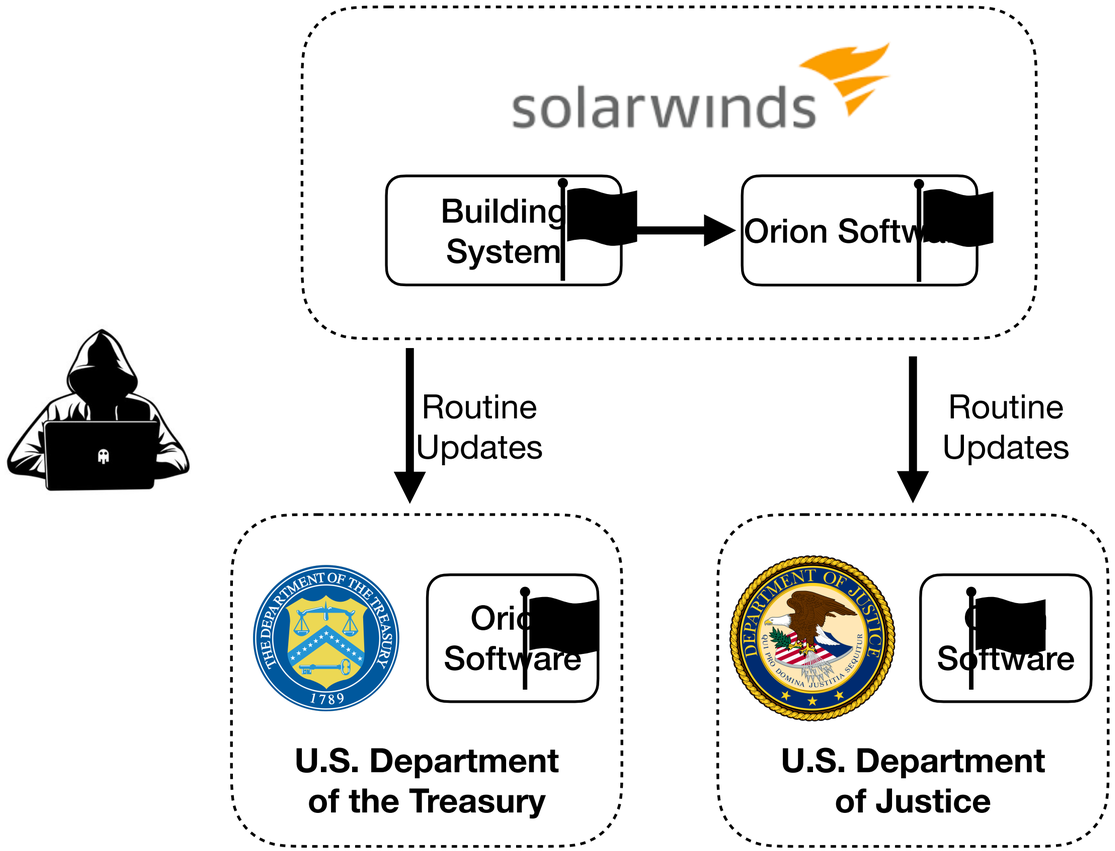
- A ransomware attack
- The largest cyberattack on an oil infrastructure target in the history of the United States
- Weak authentication (no multi-factor authentication) for the VPN access
- Fortunately, attackers compromised only the information technology systems, but not the operational technology systems which are responsible for controlling the flow of gasoline
- **We may see more attacks on industrial control systems (ICS)**

The SolarWinds Hack: background



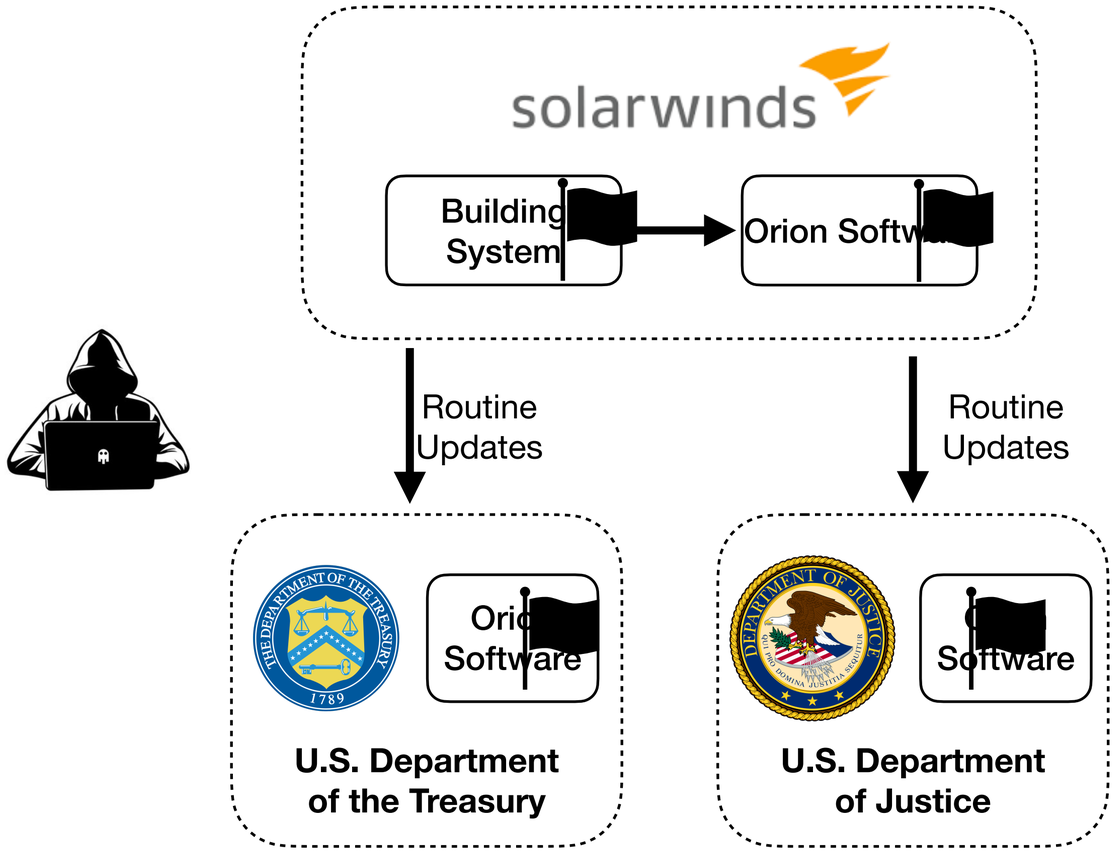
- SolarWinds, a Texas-based Software Company
- One of its major software products is Orion, a network management tool
- Orion is well adopted by more than 30K private and public organizations, e.g., U.S. Department of the Treasury
- Orion is routinely updated for new security patches, and performance enhancements, etc.

The SolarWinds Hack: timeline



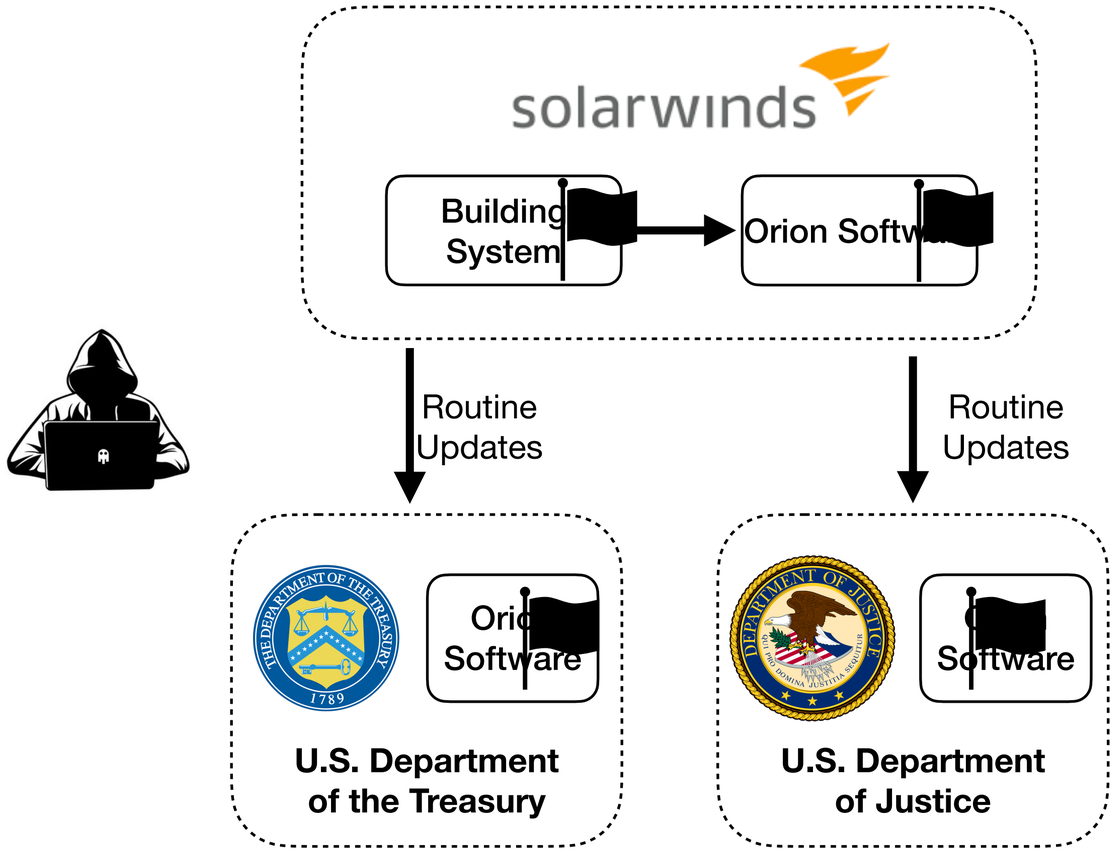
- **Sept 2019:** Attackers gained unauthorized access to SolarWinds's software building system
- **Oct 2019:** a proof-of-concept **tiny** code snippet was injected into the Orion Software, and got distributed to the clients.
- **Feb 20 2020:** the real malicious code, known as Sunburst was injected into Orion
- **Late March 2020**: the tainted Orion was distributed to clients
- **Late Dec 2020**: after being hacked for nine months, the attack was uncovered by FireEye

The SolarWinds Hack: attack techniques



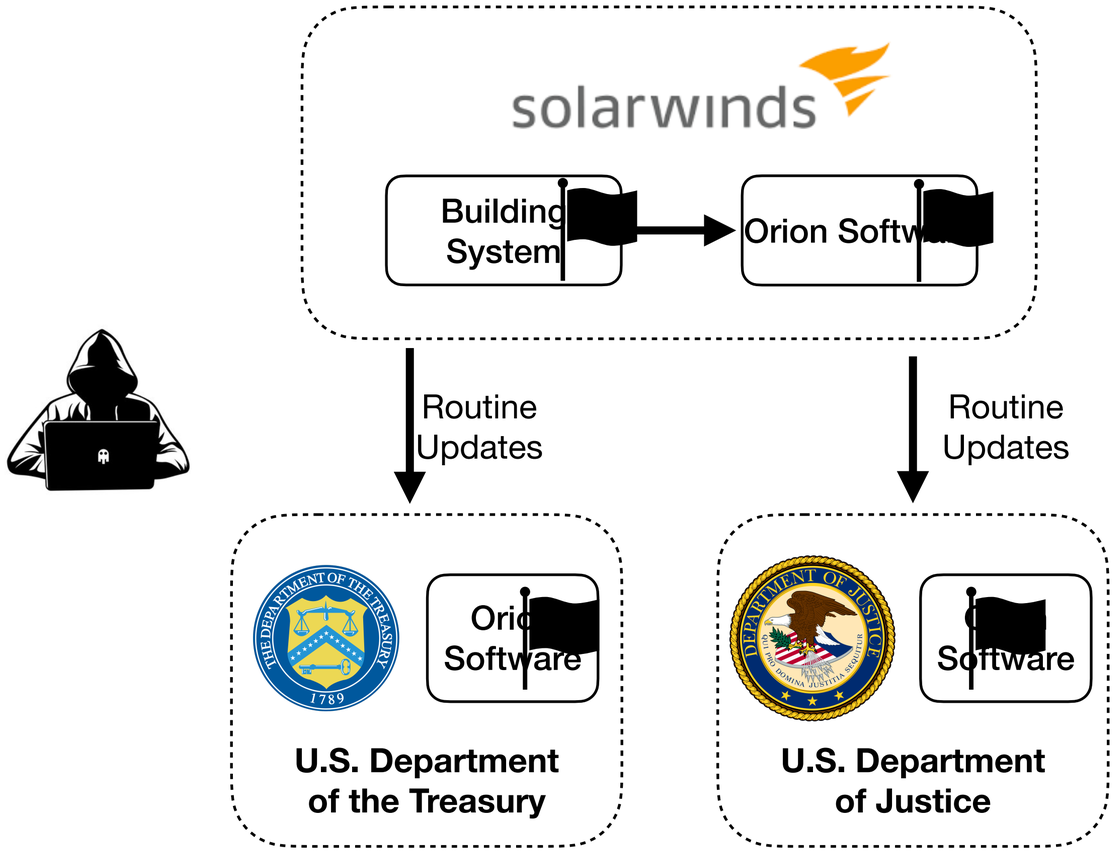
- Orion is a perfect target considering its critical role, and its customers listing on SolarWinds's website
- Mimicked the Orion software communication protocols
- Cleaned the crime scene so thoroughly investigators can't prove definitively who was behind it.

The SolarWinds Hack: results



- The hackers had gained access to the data and emails of at least nine US federal agencies, including the Department of the Treasury and the Department of Justice, and about 100 private companies.
- The hackers also punched a hole into the Cybersecurity and Infrastructure Security Agency (CISA), the office at the Department of Homeland Security
- The attack was there for nine months, it is unclear whether they were just reading emails

The SolarWinds Hack: summary



- This is a typical software supply chain attack
- Only the attackers know the whole picture, of which, many details are either undisclosed or not clear to the public
- There were early warning signs, but ignored or missed for various reasons

Security & Privacy Problems

- Misinformation or Disinformation
  - e.g., fraudulent messages, fake news, or hate speech
- User Tracking or Illicit Surveillance)
- How to make cyberspace activities trustworthy, privacy-reserving, and secure?

- What else?

Outline

- Course FAQs
- Motivations of CyberSecurity
- Overview of This Course
  - Cryptography
  - Access Control
  - Network/Web Security
  - Software/System Security
  - Privacy
  - Trustworthy AI

Cryptography

Symmetric Key vs Public Key Encryption

Symmetric Key Encryption:
The basic idea of symmetric key encryption is:

Message + Secret Key = Ciphertext

Ciphertext + Secret Key = Message

Both parties need **the same secret key** to encrypt and decrypt the message.

Public Key Encryption:
The basic idea of public key encryption is:

Message + Alice's Public Key = Ciphertext

Ciphertext + Alice's Private Key = Message

Anyone with Alice's public key can send Alice a secret message, but only Alice can decrypt.

Digital Signature and MACs

Digital Signature:

Message + Alice's Private Key = Signature
Message + Signature + Alice's Public Key = YES/NO

Alice can sign a message using her private key, and anyone can verify Alice's signature, since everyone can obtain her public key.

MAC Functions:

Message + Secret Key = Tag
Message + Tag + Secret Key = YES/NO

Need the secret key to verify the tag.

Key Management

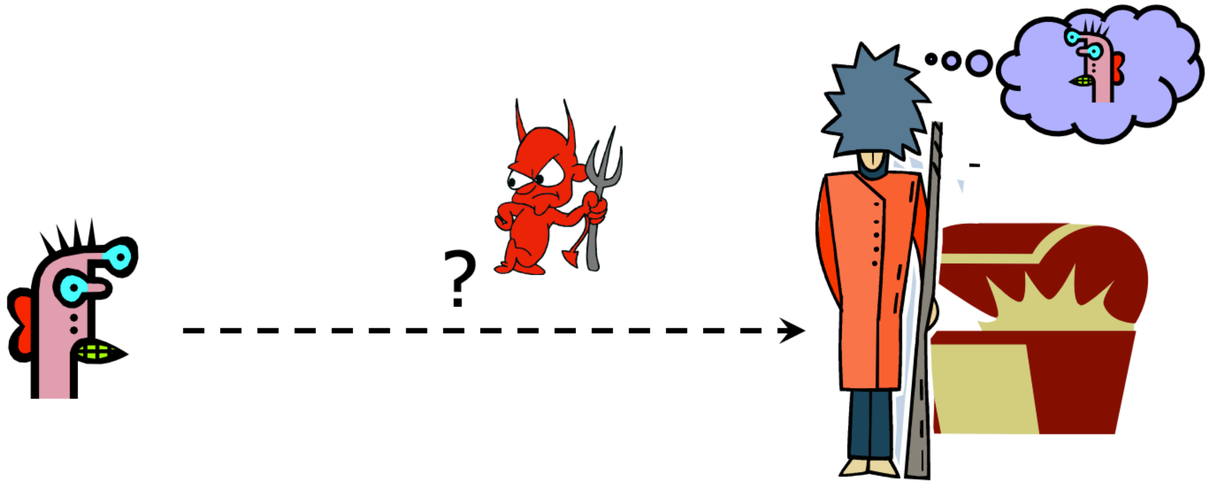
- Key Derivation
- Key Generation
- Key Storage
- Public Key Infrastructure

Cryptographic Protocols

- Zero-knowledge proof
- Secure multi-party computation
- Secure messaging
- Transport Layer Security (TLS)

Access Control

The Authentication Problem



How do you prove to someone that
you are who you claim to be?

Any system with access control must solve this problem.

Authentication is the act of proving an identity assertion, such as the identity of a computer system user.

How to

- Something the individual **knows**
  - Password, PIN, answers to prearranged questions, patterns, etc.
- Something the individual **possesses** (token)
  - Smartcard, electronic keycard, physical key,
- Something the individual **is** (physical/static biometrics)
  - Fingerprint, retina, face, brain wave

How to

- Something the individual **does** (behavioral/dynamic biometrics)
  - Voice pattern, handwriting, typing rhythm (keystroke), gait
- Where you are
  - IP, geolocation, etc.

Biometrics

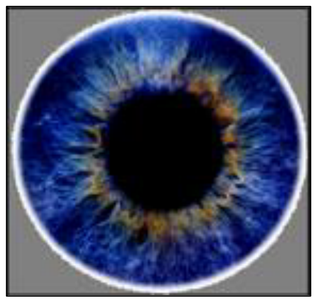
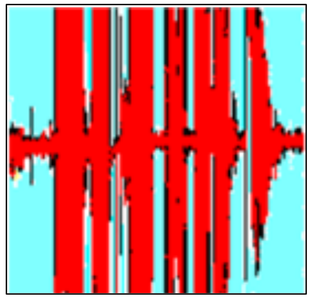
- Authentication leveraging who you are
  - Password: what you know
  - Authentication Token: what you possess

Biometrics

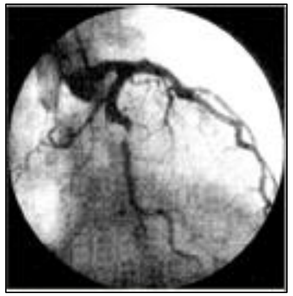
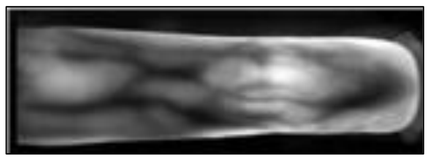
- ***"Automated recognition of individuals based on their behavioral and biological characteristics"*** *ISO/IEC JTC1 2382-37:2012*

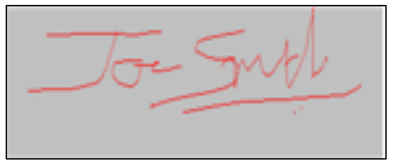
- ***Biometrics: bios (life) + metron (measure)*** *Morris,1875*

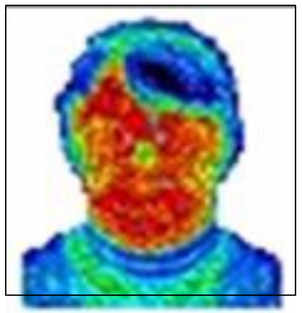




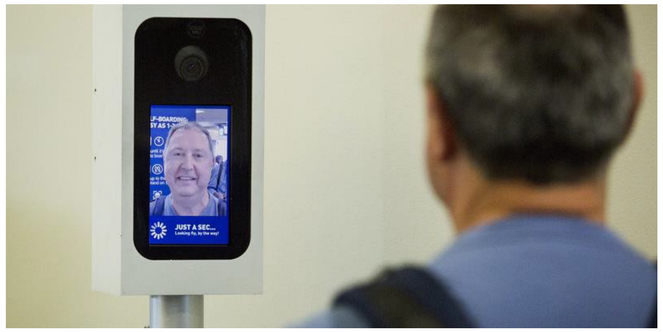



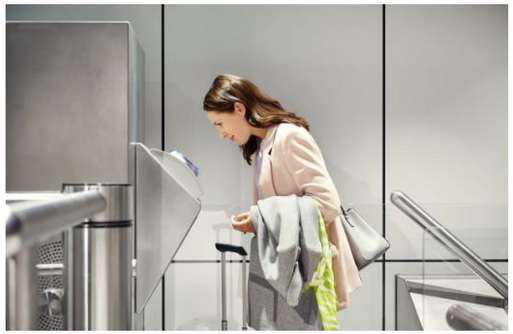



Most Popular Biometric Traits

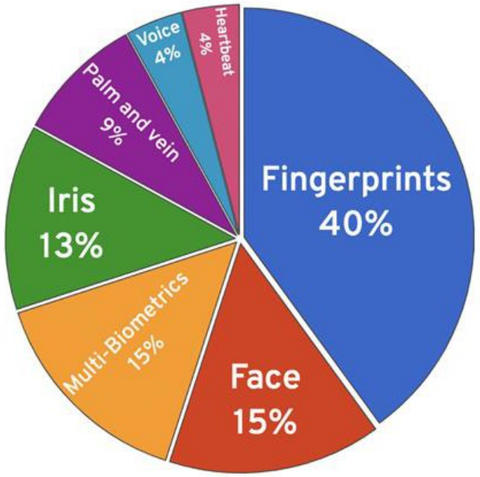
Incheon, South Korea: Smart Entry



Australia: SmartGate



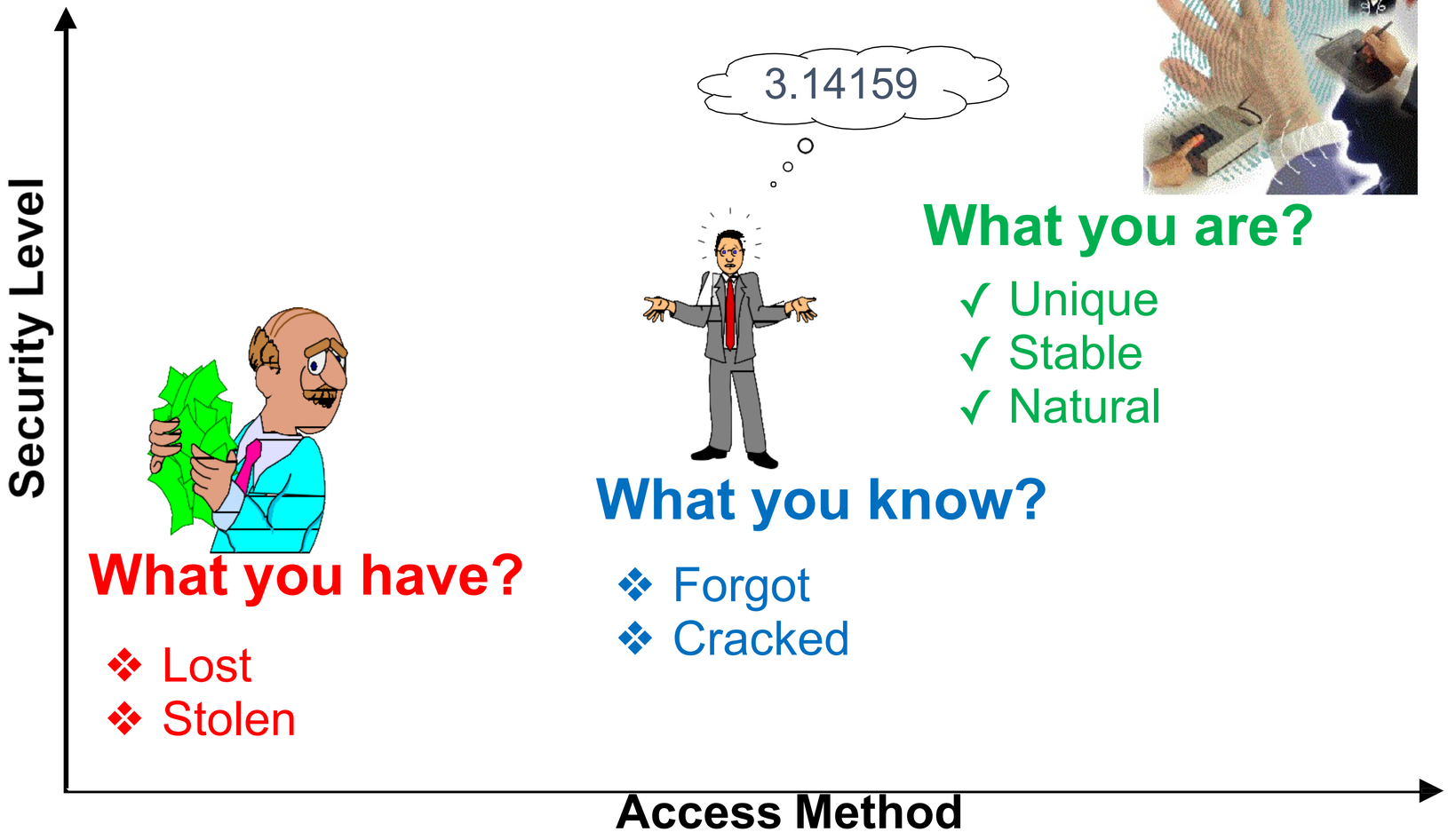
Amsterdam: Privium border passage



Uniqueness, persistence, legacy data, accuracy, fast search (1:N), NIST evaluations

http://www.homestaykorea.com/?document_srl=73667&mid=bbs_koreainfo_news
https://tottnews.com/tag/smart-gates/
https://www.idemia.com/news/multi-biometrics-future-border-control-2016-04-21

Why Biometrics?



Definitions of Access Control

- RFC 4949 defines access control as: “a process by which use of system resources is regulated according to a security policy and is permitted only by authorized entities (users, programs, processes, or other systems) according to that policy”

- NISTIR 7298 defines access control as: “the process of granting or denying specific requests to: (1) obtain and use information and related information processing services; and (2) enter specific physical facilities”

Subjects, Objects, and Access Rights

- Subject
  - An entity capable of accessing objects
  - Three classes
    - Owner
    - Group
    - World
- Object
  - A resource to which access is controlled
  - Entity used to contain and/or receive information
- Access right
  - Describes the way in which a subject may access an object
  - Could include:
    - Read
    - Write
    - Execute
    - Delete
    - Create
    - Search

Unix access control

- Process has user id
  - Inherit from creating process
  - Process can change id
    - Restricted set of options
  - Special “root” id
    - All access allowed
- File has access control list (ACL)
  - Grants permission to users
  - Three “roles”: owner, group, other

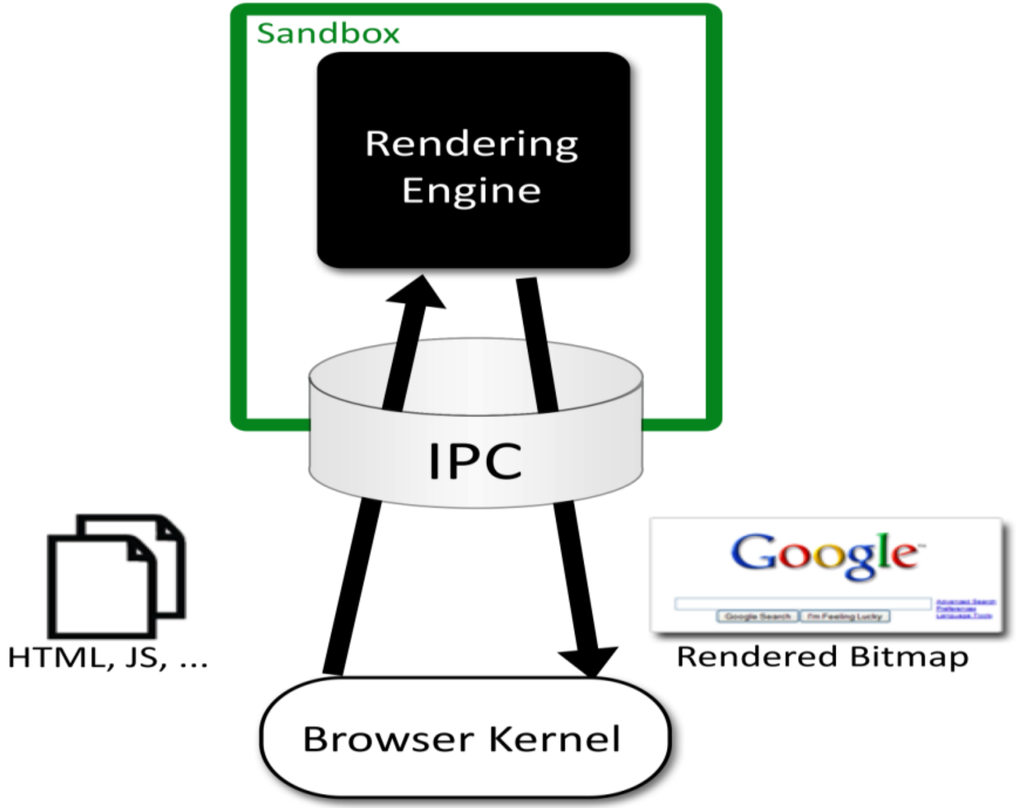
	File 1	File 2	…
Owner	read	write	-
Group	write	write	-
Other	-	-	read

Android process isolation

- Android application sandbox
  - Isolation: Each application runs with its own UID in own VM
    - Provides memory protection
    - Communication limited to using Unix domain sockets
    - Only ping, zygote (spawn another process) run as root
  - Interaction: reference monitor checks permissions on inter-component communication
  - Least Privilege: Applications announces permission
    - User grants access at install time (request permissions during runtime in latest Android versions)

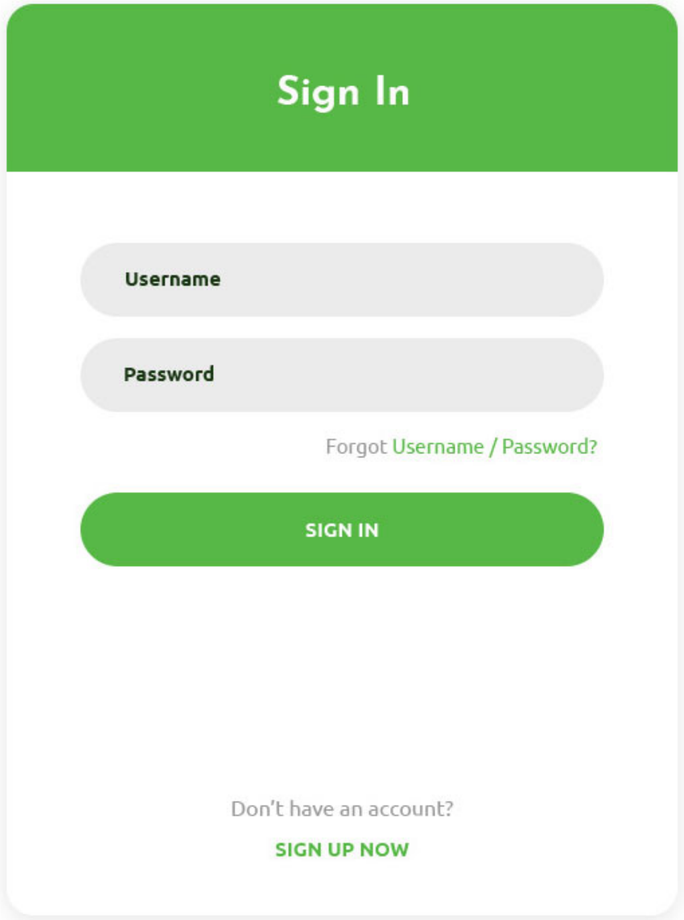
Chromium Security Architecture

- Browser ("kernel")
  - Full privileges (file system, networking)
- Rendering engine
  - Can have multiple processes
  - Sandboxed
- Browser plugin
  - Can request cross-origin permissions



Network/Web Security

SQL Injection Example




```
$login = $_POST['login'];
$pass = $_POST['password'];
$sql = "SELECT id FROM users
        WHERE username = '$login'
        AND password = '$password'";

$rs = $db->executeQuery($sql);
if $rs.count > 0 {
   // success
}
```

Non-Malicious Input

```
$u = $_POST['login’]; // xmi
$pp = $_POST['password']; // imx

$sql = "SELECT id FROM users WHERE uid = '$u' AND pwd = '$p'”;

$rs = $db->executeQuery($sql);
if $rs.count > 0 {
   // success
}
```

Non-Malicious Input

```
$u = $_POST['login’]; // xmi
$pp = $_POST['password']; // imx

$sql = "SELECT id FROM users WHERE uid = '$u' AND pwd = '$p'”;
//      "SELECT id FROM users WHERE uid = 'xmi' AND pwd = ‘imx'”
$rs = $db->executeQuery($sql);
if $rs.count > 0 {
   // success
}
```

Bad Input

```
$u = $_POST['login’]; // xmi
$pp = $_POST['password']; // imx'

$sql = "SELECT id FROM users WHERE uid = '$u' AND pwd = '$p'”;
//      "SELECT id FROM users WHERE uid = 'xmi' AND pwd = 'imx''”
$rs = $db->executeQuery($sql);
//     SQL Syntax Error
if $rs.count > 0 {
    // success
}
```

Malicious Input

```
$u = $_POST['login']; // xmi'--
$pp = $_POST['password']; // 123

$sql = "SELECT id FROM users WHERE uid = '$u' AND pwd = '$p'";
//      "SELECT id FROM users WHERE uid = 'xmi'-- AND pwd…"
$rs = $db->executeQuery($sql);
//     (No Error)
if $rs.count > 0 {
   // Success!
}
```

No Username Needed!

```
$u = $_POST['login’]; // 'or 1=1 --
$pp = $_POST['password']; // 123

$sql = "SELECT id FROM users WHERE uid = '$u' AND pwd = '$p'”;
//      "SELECT id FROM users WHERE uid = ''or 1=1 -- AND pwd…”
$rs = $db->executeQuery($sql);
//     (No Error)
if $rs.count > 0 {
    // Success!
}
```

Causing Damage

```
$u = $_POST[‘login’]; // '; DROP TABLE [users] --
$pp = $_POST['password']; // 123

$sql = "SELECT id FROM users WHERE uid = '$u' AND pwd = '$p'”;
//      "SELECT id FROM users WHERE uid = ''DROP TABLE [users]--”
$rs = $db->executeQuery($sql);
// No Error…(and no more users table)
```

Cross Site Scripting (XSS)

Cross Site Scripting: Attack occurs when application takes untrusted data and sends it to a web browser without proper validation or sanitization.

Command/SQL Injection

attacker's malicious code is executed on app's server

Cross Site Scripting

attacker's malicious code is executed on victim's browser

Session hijacking

Attacker waits for user to login

then attacker steals user's Session Token
and "hijacks" session

⇒ attacker can issue arbitrary requests on behalf of user

Example: **FireSheep**

Firefox extension: hijacks HTTP session tokens over WiFi

Solution: always send session tokens over HTTPS!

DNS Spoofing

Scenario: DNS client issues query to server

Attacker would like to inject a fake reply
  Attacker does not see query or real response

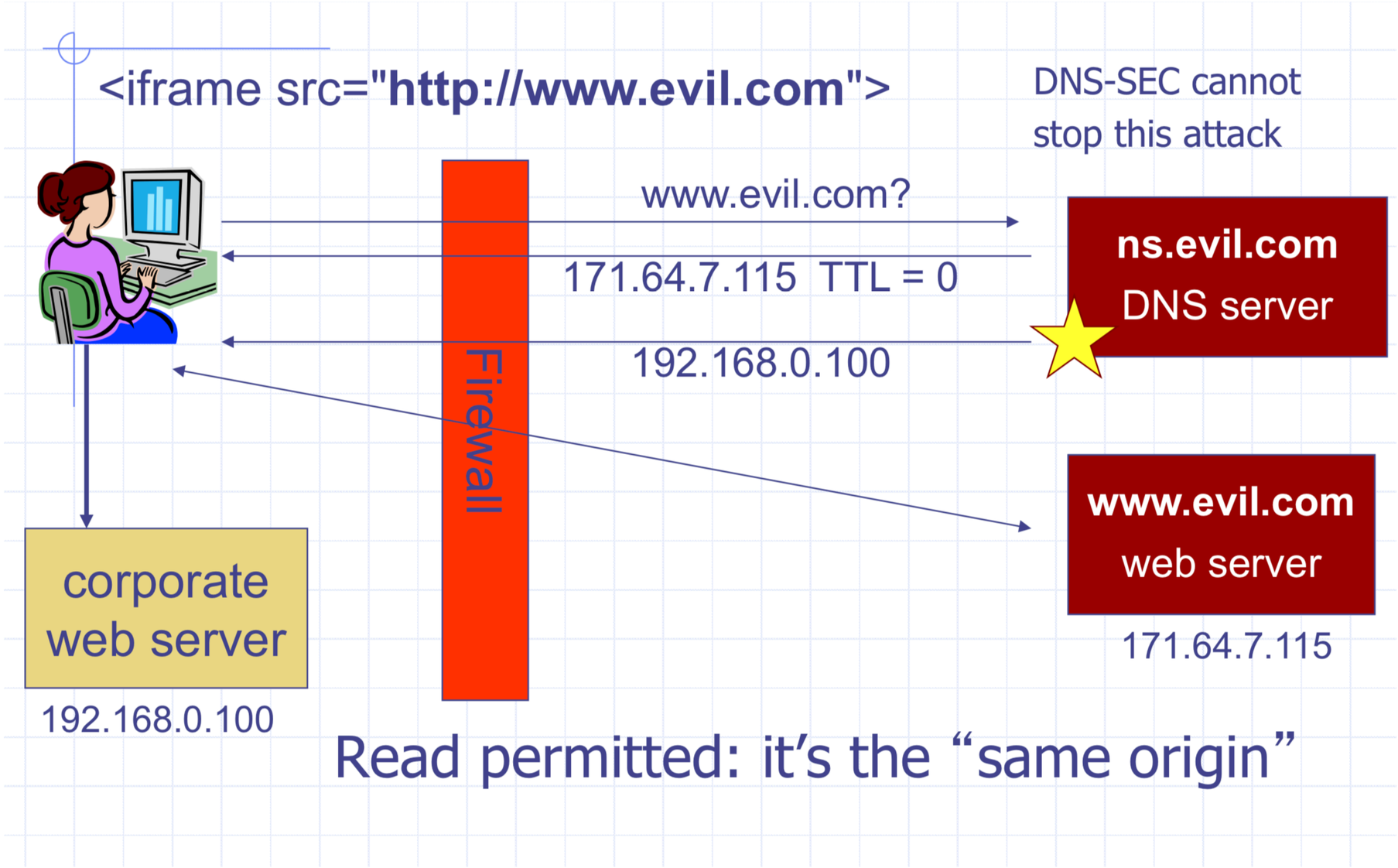
How does client authenticate response?

DNS Cache Poisoning

DNS query results include Additional Records section
– Provide records for anticipated next resolution step

Early servers accepted and cached all additional records provided in query response

DNS Rebinding



Denial of Service Attacks

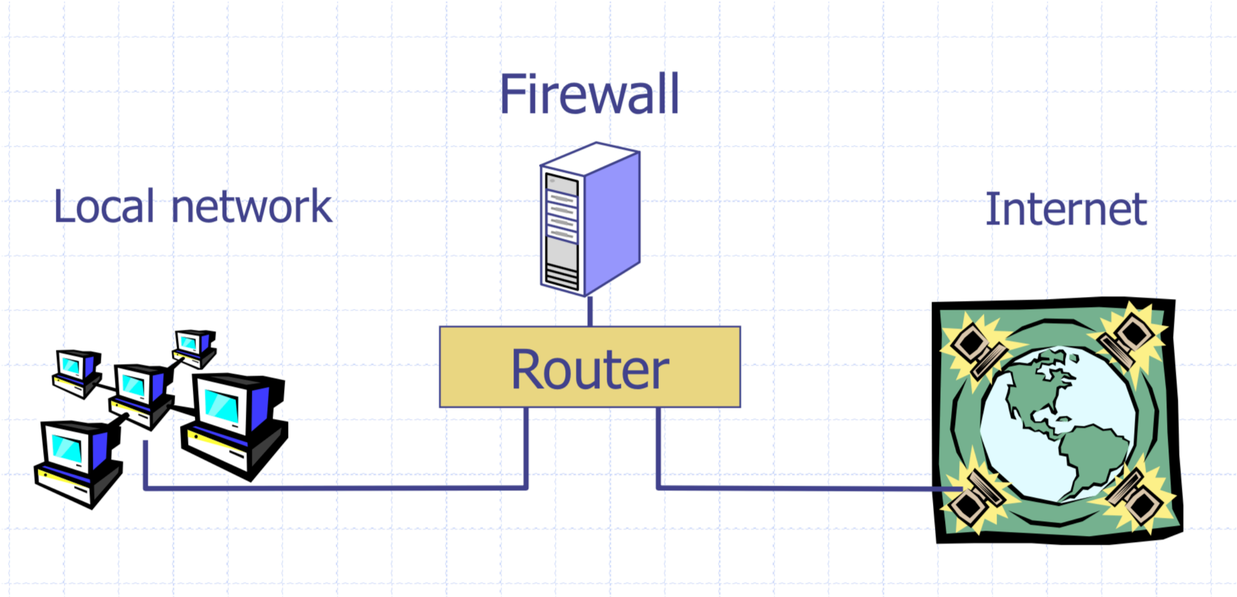
Goal: take large site offline by overwhelming it with network traffic such that they can't process real requests

How: find mechanism where attacker doesn't have to spend a lot of effort, but requests are difficult/expensive for victim to process

Firewalls

Separate local area network (LAN) from the Internet. Only allow some traffic to transit.

Sometimes rules on a router. Sometimes a standalone device.



Intrusion Detection Systems (IDS)

Software/device to monitor network traffic for attacks or policy violations

Violations are reported to a central security information and event management (SIEM) system where analysts can later investigate

Signature Detection: maintains long list of traffic patterns (rules) associated with attacks

Anomaly Detection: attempts to learn normal behavior and report deviations

Virtual Private Networks (VPNs)

Problem: How do you provide secure communication for non-TLS protocols across the public Internet?

VPNs create a fake shared network on which traffic is encrypted

Two Broad Types:

- Remote client (e.g., traveler with laptop) to corporate network
- Connect two remote networks across Internet

Software/System/Hardware Security

x86 Assembly and Call Stack

- C memory layout
  - **Code** section: Machine code (raw bits) to be executed
  - **Static** section: Static variables
  - **Heap** section: Dynamically allocated memory (e.g. from **`malloc`**)
  - **Stack** section: Local variables and stack frames
- x86 registers
  - **EBP** register points to the top of the current stack frame
  - **ESP** register points to the bottom of the stack
  - **EIP** register points to the next instruction to be executed
- x86 calling convention
  - When calling a function, the old EIP (RIP) is saved on the stack
  - When calling a function, the old EBP (SFP) is saved on the stack
  - When the function returns, the old EBP and EIP are restored from the stack

Memory Safety Vulnerabilities

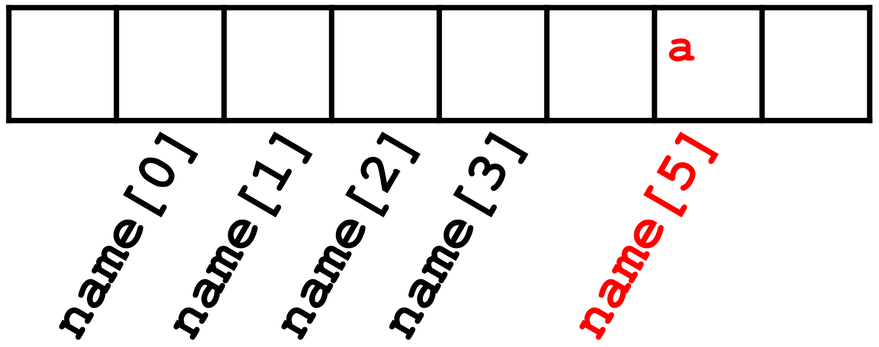
- Buffer overflows
  - Stack smashing
  - Memory-safe code
- Integer memory safety vulnerabilities
- Format string vulnerabilities
- Heap vulnerabilities
- Writing robust exploits

Buffer Overflow Vulnerabilities

- Recall: C has no concept of array length; it just sees a sequence of bytes
- If you allow an attacker to start writing at a location and don't define when they must stop, they can overwrite other parts of memory!

```
char name[4];
name[5] = 'a';
```

This is technically valid C code, because C doesn't check bounds!



Vulnerable C Library Functions

- **`gets`** - Read a string from stdin
  - Use **`fgets`** instead
- **`strcpy`** - Copy a string
  - Use **`strncpy`** (more compatible, less safe) or **`strlcpy`** (less compatible, more safe) instead
- **`strlen`** - Get the length of a string
  - Use **`strnlen`** instead (or **`memchr`** if you really need compatible code)
- … and more (look up C functions before you use them!)

Signed/Unsigned Vulnerabilities

Is this safe?

```
void func(int len, char *data) {
    char buf[64];
    if (len > 64)
        return;
    memcpy(buf, data, len);
}
```

`int` is a **signed** type, but `size_t` is an **unsigned** type. What happens if `len == -1`?

This is a **signed** comparison, so `len > 64` will be false, but casting **-1** to an unsigned type yields `0xffffffff`: another buffer overflow!

```
void *memcpy(void *dest, const void *src, size_t n);
```

Integer Overflow Vulnerabilities

Is this safe?

What happens if `len == 0xffffffff`?

```
void func(size_t len, char *data) {
    char *buf = malloc(len + 2);
    if (!buf)
        return;
    memcpy(buf, data, len);
    buf[len] = '\n';
    buf[len + 1] = '\0';
}
```

`len + 2 == 1`, enabling a heap overflow!

Memory Safety Mitigations

- Memory-safe languages
- Writing memory-safe code
- Building secure software
- Exploit mitigations (Increase the difficulty of exploitations)
  - Non-executable pages
  - Stack canaries
  - Pointer authentication
  - Address space layout randomization (ASLR)
- Combining mitigations

Writing Memory-Safe Code

- Defensive programming: Always add checks in your code just in case
  - Example: Always check a pointer is not null before dereferencing it, even if you're sure the pointer is going to be valid
  - Relies on programmer discipline
- Use safe libraries
  - Use functions that check bounds
  - Example: Use **`fgets`** instead of **`gets`**
  - Example: Use **`strncpy`** or **`strlcpy`** instead of **`strcpy`**
  - Example: Use **`snprintf`** instead of **`sprintf`**
  - Relies on programmer discipline or tools that check your program

Malware

- **Malware** (**mal**icious soft**ware**): Attacker code running on victim computers
  - Sometimes called **malcode** (**mal**icious **code**)
- Catch-all term for many different types of attacker code, including code that:
  - Deletes files
  - Sends spam email
  - Launches a DoS attack
  - Steals private information
  - Records user inputs (keylogging, screen capture, webcam capture)
  - Encrypts files and demands money to decrypt them (ransomware)
  - Physically damages machines

Malware

- Detection methods: Signature-based detection, antivirus, flag unfamiliar code
- Propagation methods
  - Polymorphic code: Encrypt the malware with a different key each time
  - Metamorphic code: Change the semantics of the code each time
  - Helps evade signature-based detection
- Recovery method: Reset everything and start from scratch
- Rootkits: Malware in the operating system that hides its presence

Malware

- Self-replicating code
- Viruses
  - Propagation strategies
  - Detection strategies
  - Polymorphic code
  - Metamorphic code
- Worms
  - Propagation strategies
  - Modeling worm propagation
  - History of worms
- Infection cleanup and rootkits

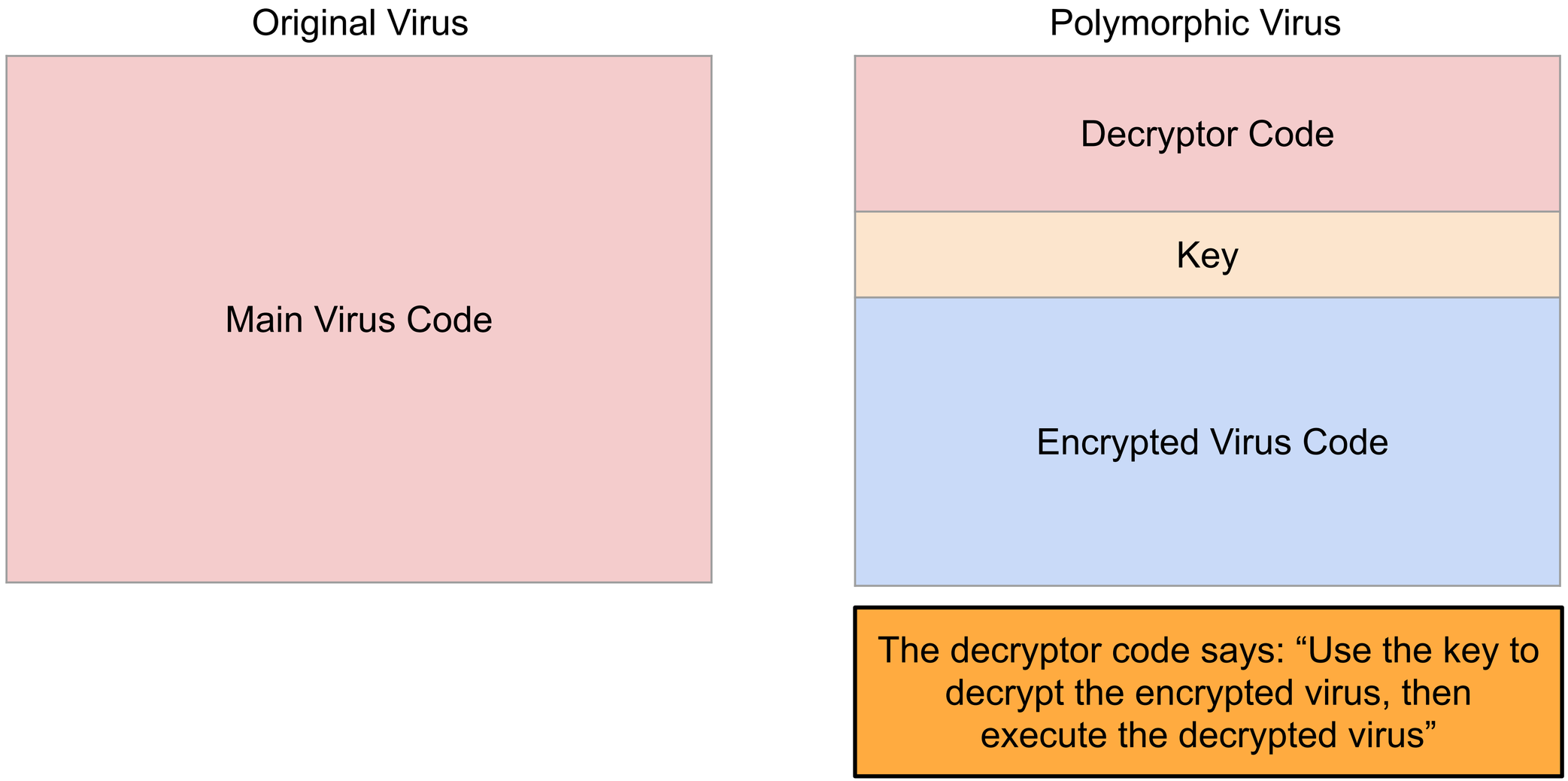
Viruses and Worms

- Viruses and worms are both malware that automatically self-propagate
  - The malicious code sends copies of itself to other users
- **Virus**: Code that requires user action to propagate
  - Usually infects a computer by altering some stored code
  - When the user runs the code, the code spreads the virus to other users
- **Worm**: Code that does not require user action to propagate
  - Usually infects a computer by altering some already-running code
  - No user interaction required for the worm to spread to other users
- The difference between a virus and a worm is not always clear
  - Some malware uses both approaches together
  - Example: Trojan malware does not self-propagate, but instead requires user action

Polymorphic Code

- **Polymorphic code**: Each time the virus propagates, it inserts an encrypted copy of the code
  - The code also includes the key and decryptor
  - When the code runs, it uses the key and decryptor to obtain the original malcode
- Recall: Encryption schemes can produce different-looking output on repeated encryptions
  - Example: Using a different IV for each encryption
  - Example: Using a different key for each encryption
- Encryption is being used for **obfuscation**, not confidentiality
  - The goal is to evade detection by making the virus look different
  - **The goal is not to prevent anyone from reading the virus contents, Why?**
  - Weaker encryption algorithms can be used, and the key can be stored in plaintext

Polymorphic Code

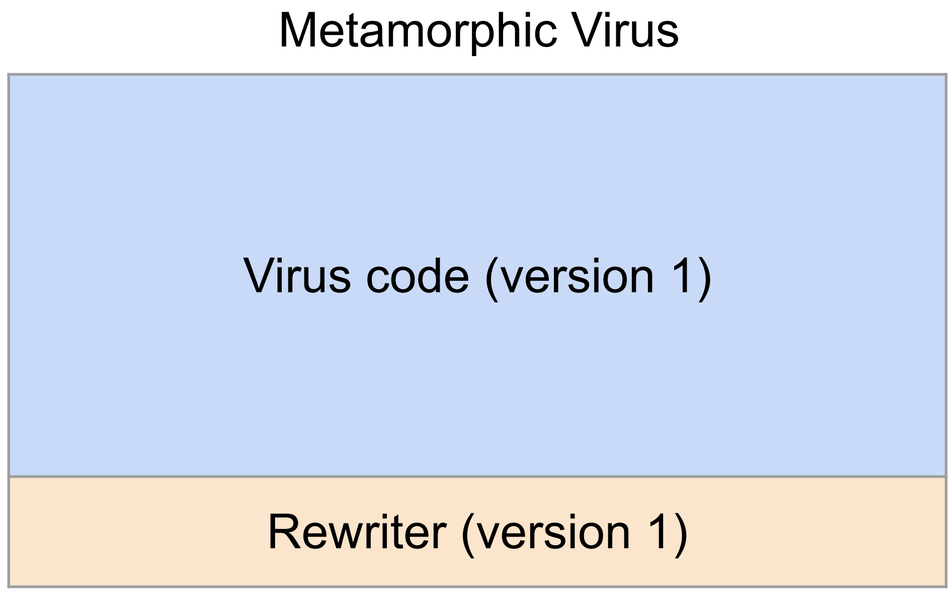


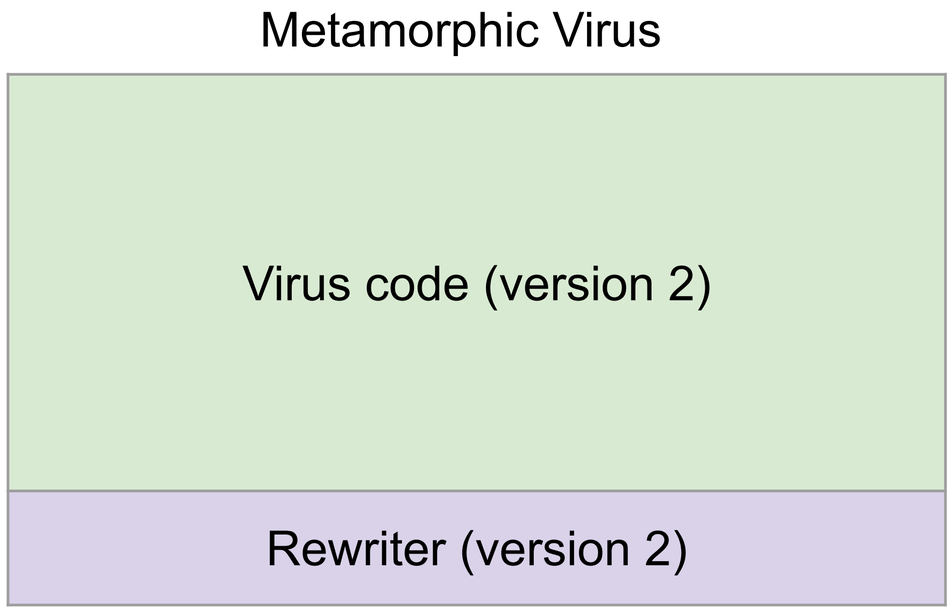
Metamorphic Code

- **Metamorphic code**: Each time the virus propagates, it generates a semantically different version of the code
  - The code performs the same high-level action, but with minor differences in execution
- Include a code rewriter with the virus to change the code randomly each time
  - Renumber registers
  - Change order of conditional (if/else) statements
  - Reorder independent operations
  - Replace a low-level algorithm with another (e.g. mergesort and quicksort)
  - Add some code that does nothing useful (or is never executed)

(blue, not tested on exams)

Metamorphic Code





The rewriter code says: “Construct a semantically different version of this virus, and spread the new version”

Note: The rewriter code itself will also be rewritten!

Running untrusted code

We often need to run buggy/unstrusted code:

- programs from untrusted Internet sites:
  - mobile apps, Javascript, browser extensions
- exposed applications: browser, pdf viewer, outlook
- legacy daemons: sendmail, bind
- honeypots

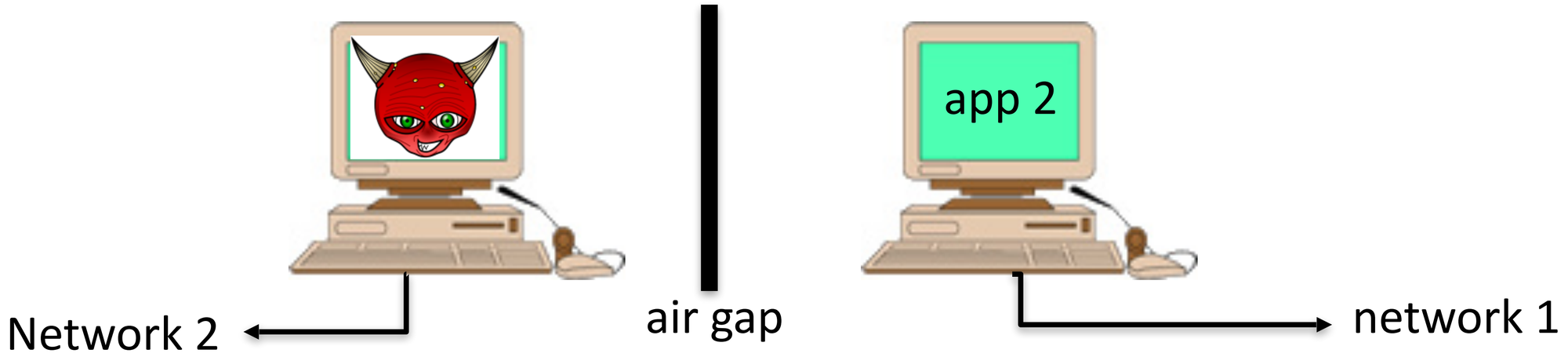
Goal: if an application “misbehaves” ⇒ Fire alarms and Kill it

Approach: confinement

Confinement: ensure misbehaving app cannot harm rest of system

Can be implemented at many levels:

- **Hardware**: run application on isolated hw (air gap)



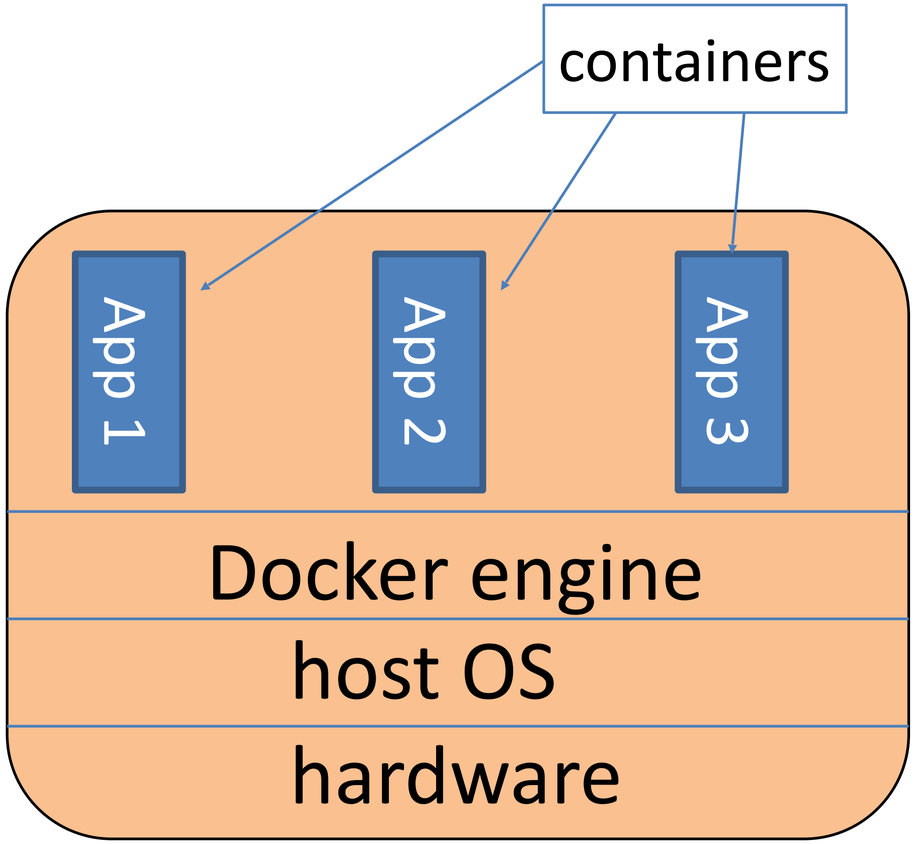
⇒ difficult to manage

Docker: isolating containers using seccomp-bpf

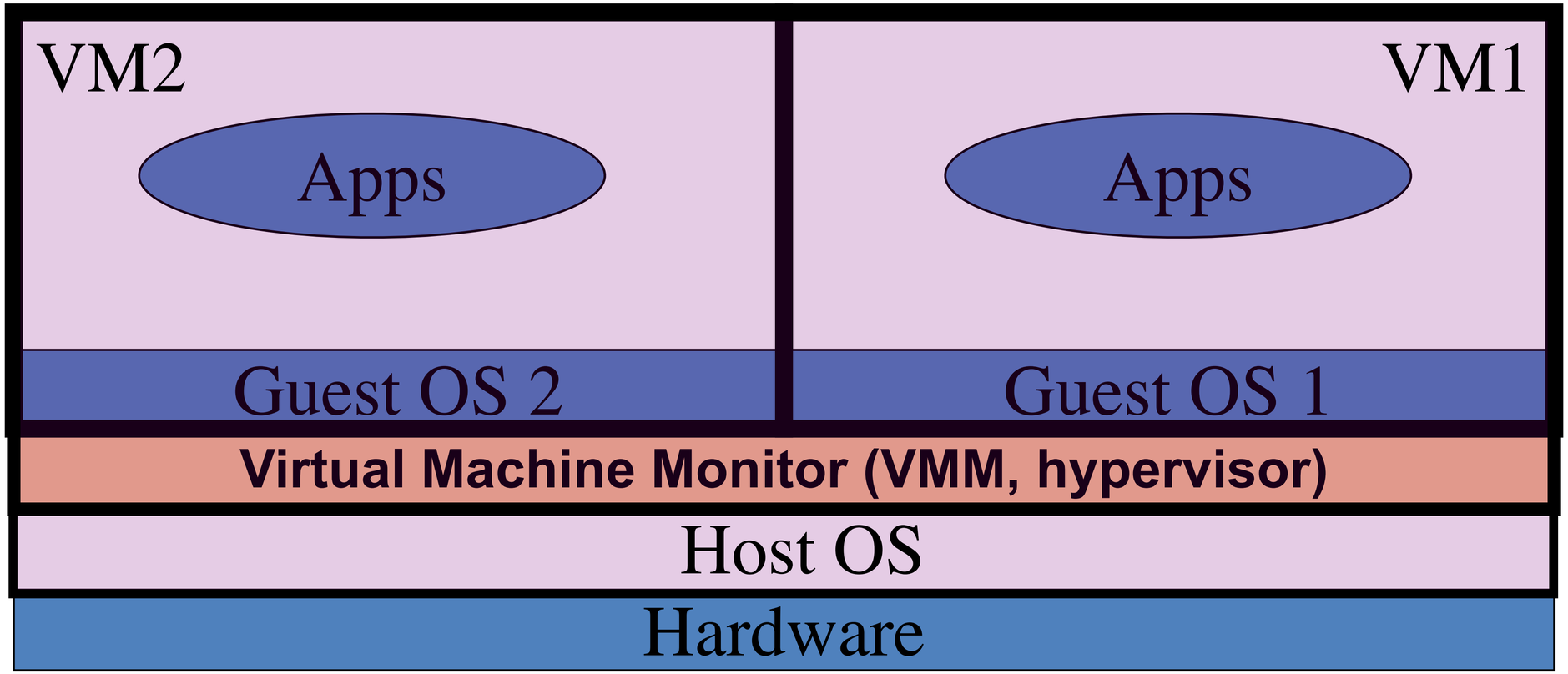
Container: process level isolation

- Container prevented from making sys calls filtered by secomp-BPF

- Whoever starts container can specify BPF policy
  - default policy blocks many syscalls, including ptrace



Virtual Machines

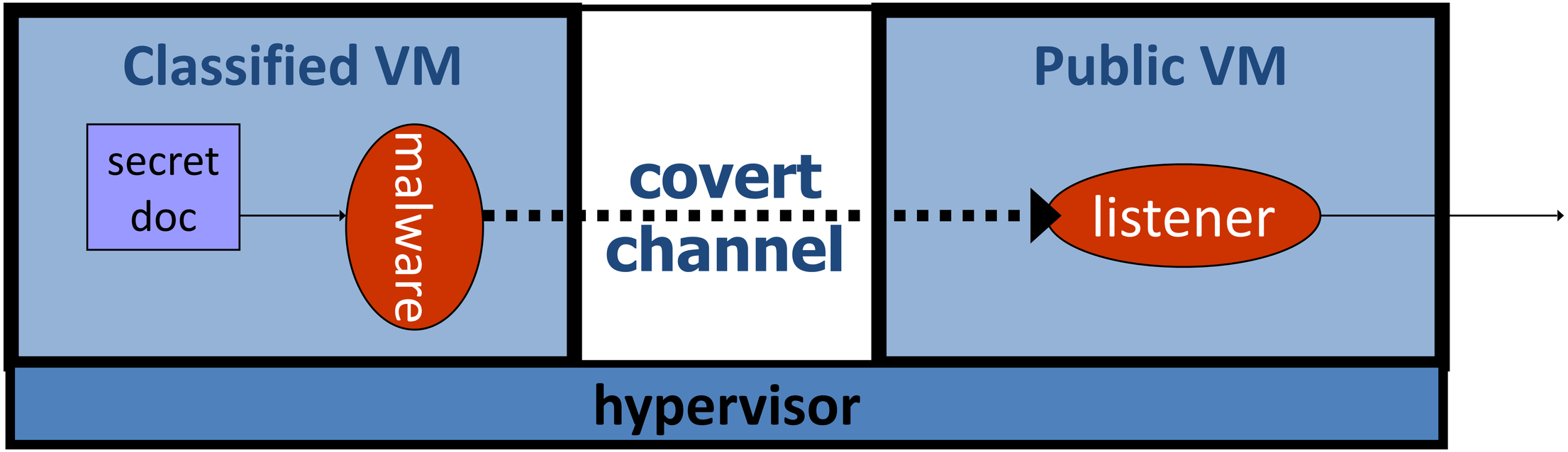


single HW platform with isolated components

Problem: covert channels

Covert channel: unintended communication channel between isolated components

- Can leak classified data from secure component to public component



Hypervisor detection

- VM platforms often emulate simple hardware
  - VMWare emulates an ancient i440bx chipset

    … but report 8GB RAM, dual CPUs, etc.

- Hypervisor introduces time latency variances
  - Memory cache behavior differs in presence of hypervisor
  - Results in relative time variations for any two operations

- Hypervisor shares the TLB (translation lookaside buffer) with GuestOS
  - GuestOS can detect reduced TLB size

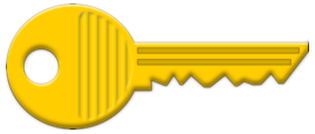
- … and many more methods **[GAWF'07]**

What if the computing environment is not trusted?

SGX: Goals

- Extension to Intel processors that support:
- **Enclaves**: running code and memory isolated from the rest of system
- **Attestation**: prove to local/remote system what code is running in enclave
- **Minimum TCB**: only processor is trusted
- nothing else: DRAM and peripherals are untrusted
- ⇒ all writes to memory are encrypted

Applications



- **<u>Server side</u>**:
- Storing a Web server HTTPS secret key:
- secret key only opened inside an enclave
- ⇒ malware cannot get the key
- Running a private job in the cloud: job runs in enclave
- Cloud admin cannot get code or data of job
- **<u>Client side</u>**:
- Hide anti-virus (AV) signatures:
- AV signatures are only opened inside an enclave
- not exposed to adversary in the clear

Privacy

What is Privacy?

Privacy is control over your own information. Freedom from intrusion into personal matters

Privacy is **a person's right or expectation** to control the disclosure of his/her personal information, including activity metadata

Privacy is the "right to be let alone" — Louis Brandeis

Privacy means something like what the Founders of US meant by "liberty"
- Free speech, free association, autonomy, …
- freedom from censorship and constant surveillance

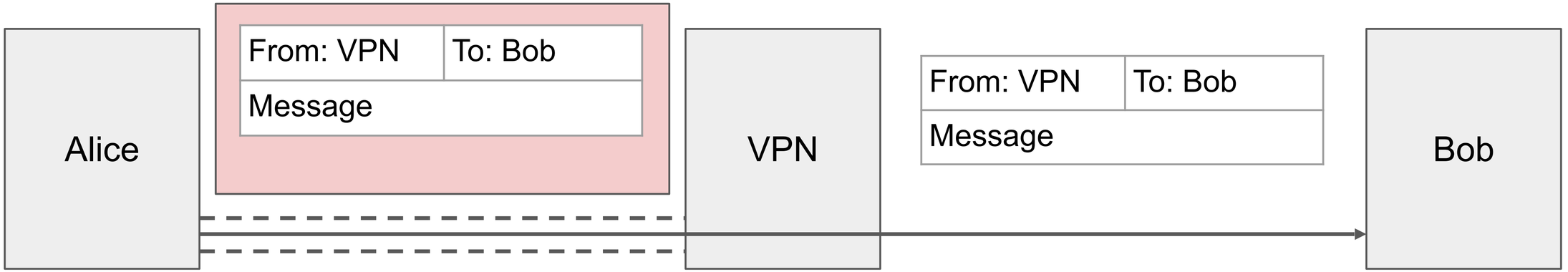
Privacy-motivating examples
- Martin Luther King Jr. "blackmailed" by FBI
- McCarthyism witch-hunt for communists

Anonymity

- **Anonymity**: Concealing your identity
  - Anonymous communication on the Internet: The identity of the source and/or destination are concealed
- Anonymity is not confidentiality
  - Confidentiality hides the contents of the communication
  - Anonymity hides the identities of who is communicating with whom

Can Virtual Private Networks (VPNs) Fulfill Anonymity?

- Recall VPNs: A virtual connection to an internal network
  - Allows access to an internal network through an encrypted tunnel
  - Creates an alternative use case: Appear as though you are coming from the virtually connected network instead of your real network!
    - Similar concept to proxies, but Alice directly sends packets as though coming from the VPN, wrapped in the VPN's layer of encryption
    - Proxies operate at the application layer, while VPNs operate at the network layer



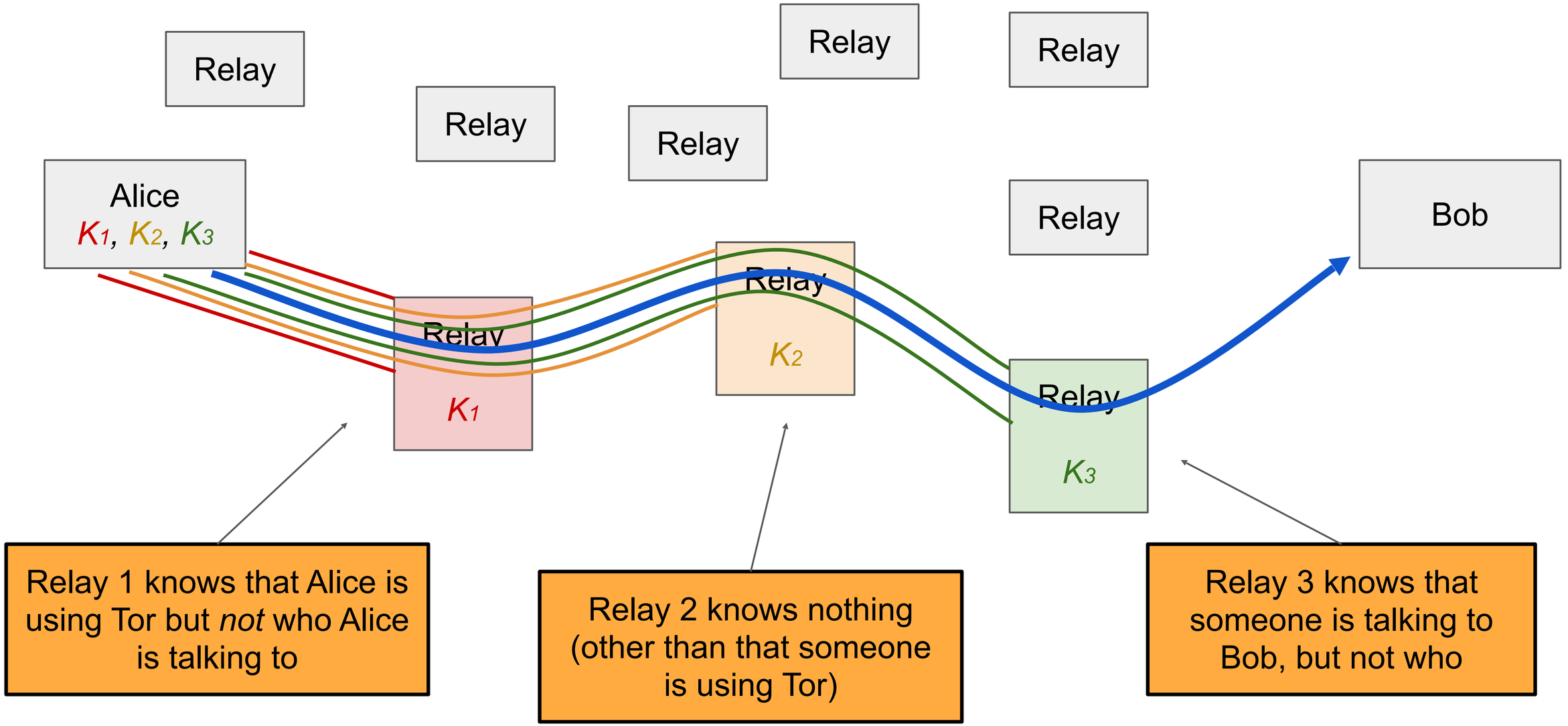
Real Use for VPNs

- Evading censorship
  - A local adversary can’t see your traffic…
    - But the censor could just block VPN traffic instead
- Access control
  - Systems that only allow “internal” access or access from known networks…
  - The campus VPN is about solving this problem

Tor

- **Idea**: Send the packet through multiple proxies instead of just one proxy
- **Tor**: A network that uses multiple proxies (relays) to enable anonymous communications
  - Stands for **T**he **O**nion **R**outer
- Components of Tor
  - Tor network: A network of many **Tor relays** (proxies) for forwarding packets
  - Directory server: Lists all Tor relays and their public keys
  - Tor Browser: A web browser configured to connect to the Tor network (based on Firefox)
  - Tor onion services: Servers that can only be reached through the Tor network
  - Tor bridges: Tor relays that try to hide the fact that a user is connecting to the Tor network

Tor Circuits



Tor Weaknesses: Timing Attacks

- A network attacker who has a full (**global**) view of the network can learn that Alice and Bob are talking
  - Exploit a timing attack: Observe when Alice sends a message, when Bob receives a message, and link the two together
- Global adversaries are *outside* of Tor's threat model and are not defended against
  - Tor only defends against local adversaries with partial views of the network
  - Timing attacks could be defended against by delaying the timing of packets, but this violates Tor's performance goal

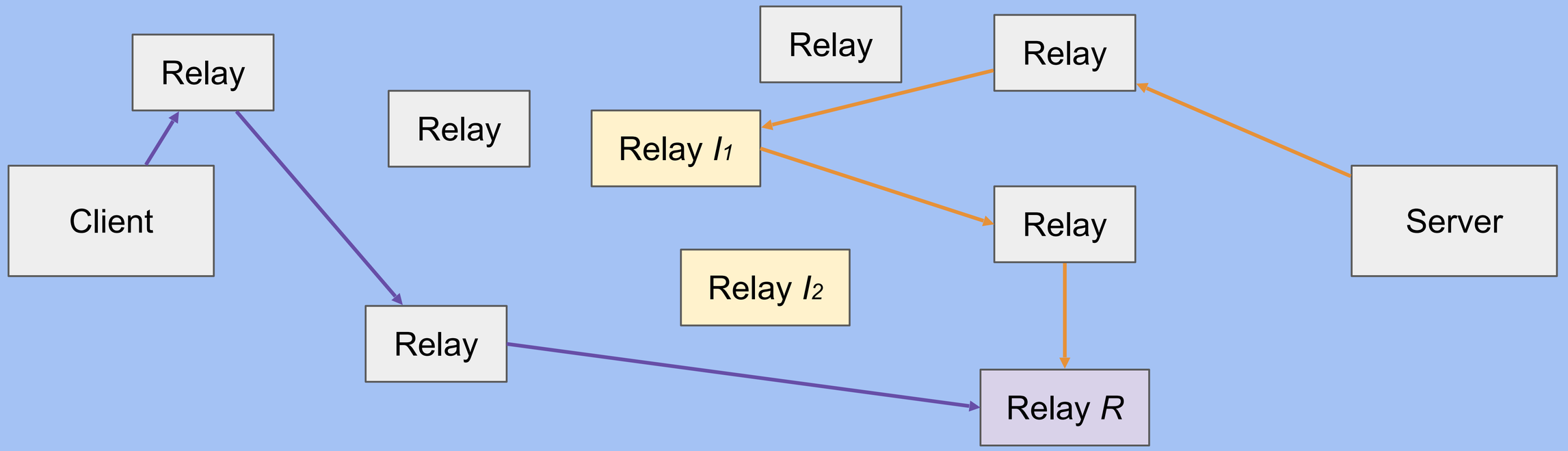
Tor Weaknesses: Collusion

- **Collusion**: Multiple nodes working together and sharing information
  - Collusion is adversarial (dishonest) behavior
  - Honest nodes should never share information with other proxies
  - If *all* nodes in the circuit collude, anonymity is broken
  - If *at least one* nodes in the circuit is honest, anonymity is preserved
- It is easy to form some amount of colluding nodes
  - An attacker can create hundreds nodes in the Tor network to increase the chance that your circuit consists entirely of the attacker's nodes!
- The more nodes we use, the more confident we are that they are not all colluding
  - It's much harder for 10 nodes to collude than for 2 nodes to collude
  - 3 nodes is generally considered good enough for industrial-grade security and is the default

Tor Weaknesses: Distinguishable Traffic

- Tor does *not* hide the fact that you are using Tor
  - Example: A local adversary can see that you are sending packets to a Tor relay
  - Anonymity only works in a crowd
    - Example: A Harvard student sends an anonymous threatening message using Tor. The administrators notice that only one student on the Harvard network is using Tor!
    - Every Tor browser should be configured similarly, so network adversaries cannot distinguish any patterns in the packets
    - Tor browsers should resist tracking (e.g. no tracking cookies)

Tor Onion Services



Notice: The client and server never directly communicate, and the introduction and rendezvous points don't know the client or the server

Trustworthy AI

Adversarial ML

- Attacks
  - Studies how can Machine Learning Fail
  - Different attack models
    - Attack objective and knowledge about the ML system
- Defenses
  - How to defend Machine Learning against different failures and improve their robustness
  - What are the tradeoffs between accuracy and robustness

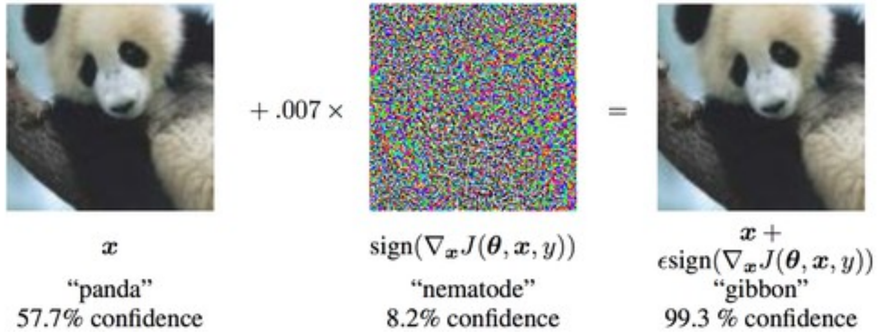
Adversarial Machine Learning: Taxonomy

Attacker's Objective

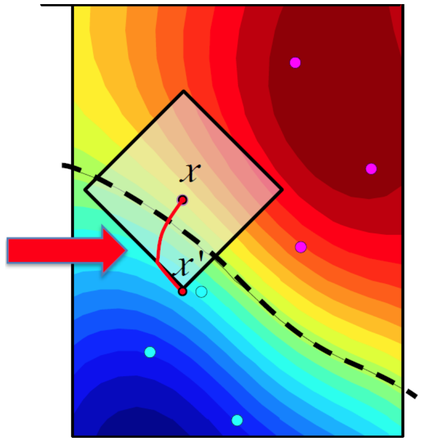
Learning stage

	Targeted Target small set of points	**Availability** Target majority of points	**Privacy** Learn sensitive information
Training	Targeted Poisoning Backdoor Trojan Attacks	Poisoning Availability Model Poisoning	-
Testing	Evasion Attacks Adversarial Examples	-	Reconstruction Membership Inference Model Extraction

Evasion Attacks



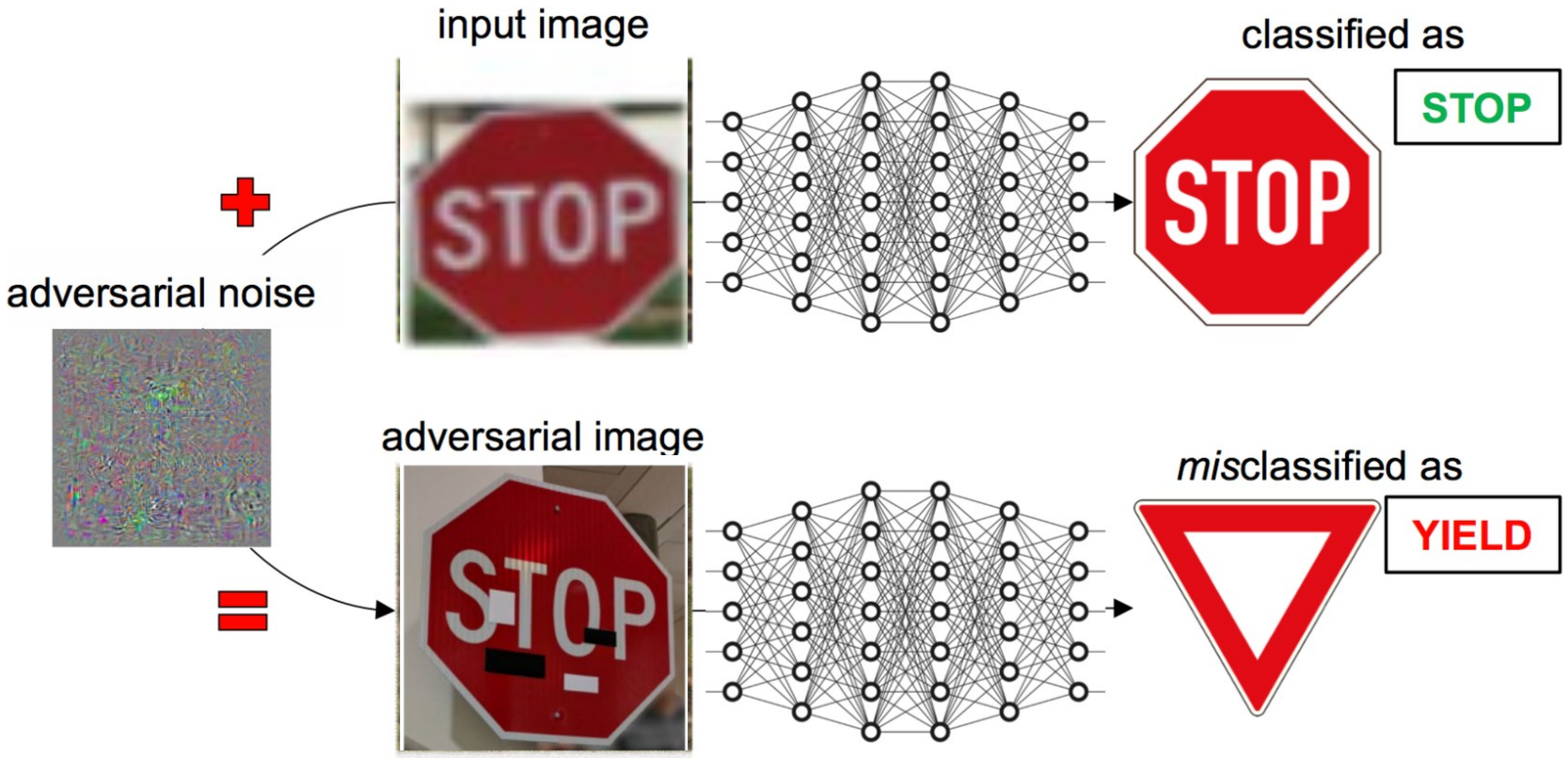
Adversarial example



- Evasion attack: attack against ML at testing time
- Implications
  - Small (imperceptible) modification at testing time can change the classification of any data point to any targeted class

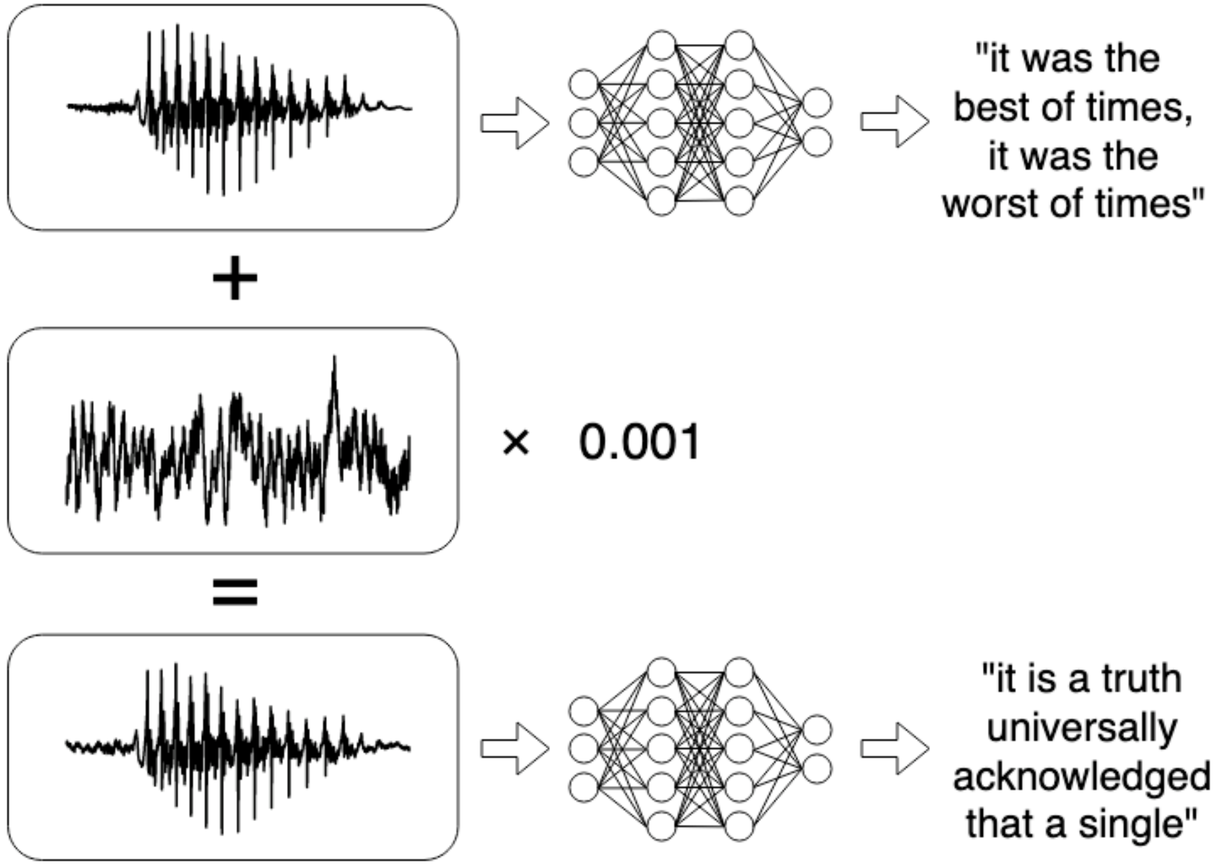
- Szegedy et al. *Intriguing properties of neural networks*. 2014 https://arxiv.org/abs/1312.6199
- Goodfellow et al. *Explaining and Harnessing Adversarial Examples*. 2014. https://arxiv.org/abs/1412.6572

Adversarial Attacks on Road Signs



Eykholt et al. *Robust Physical-World Attacks on Deep Learning Visual Classification*. In CVPR 2018

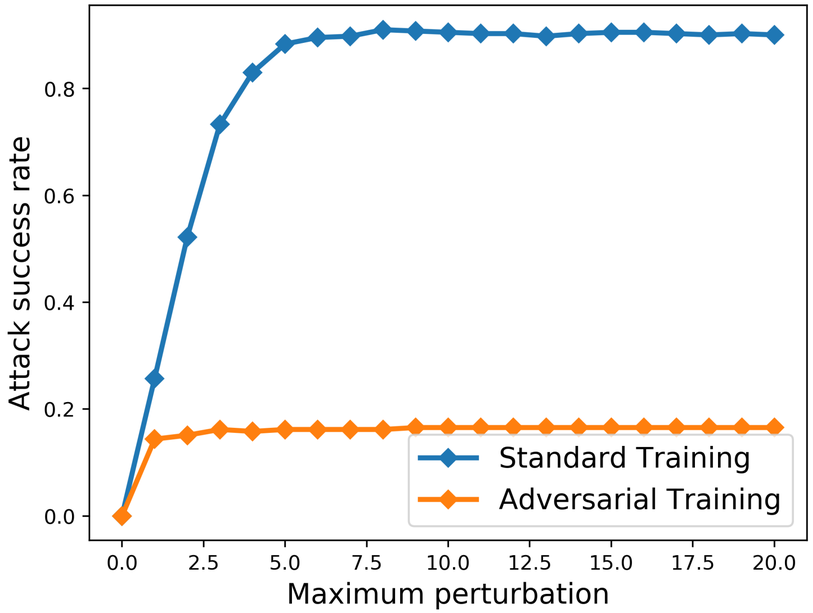
Speech Recognition



https://nicholas.carlini.com/code/audio_adversarial_examples/

Defense: Adversarial Training

- Adversarial Training
  - Train model iteratively
  - In each iteration, generate adversarial examples and add to training with correct label
- Implications
  - Adversarial training can improve ML robustness
- Challenges
  - Computationally expensive
    - Specific to certain attacks
    - Does it generalize to other attacks?

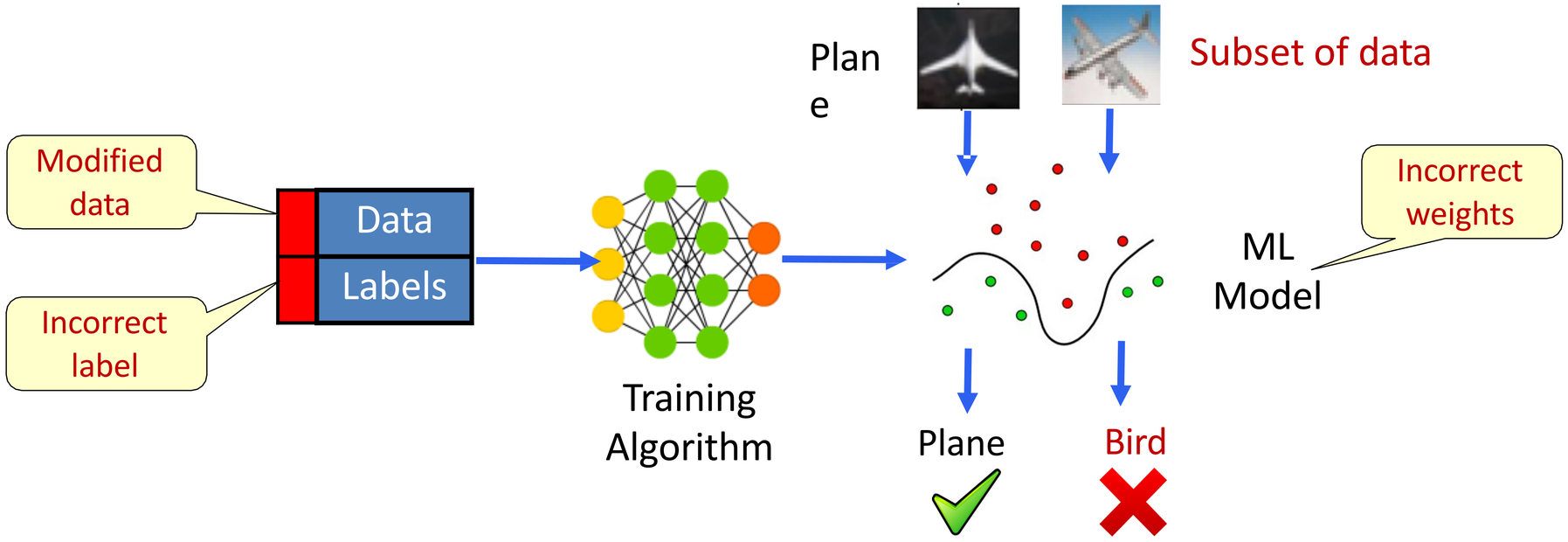


Malicious domain classifier

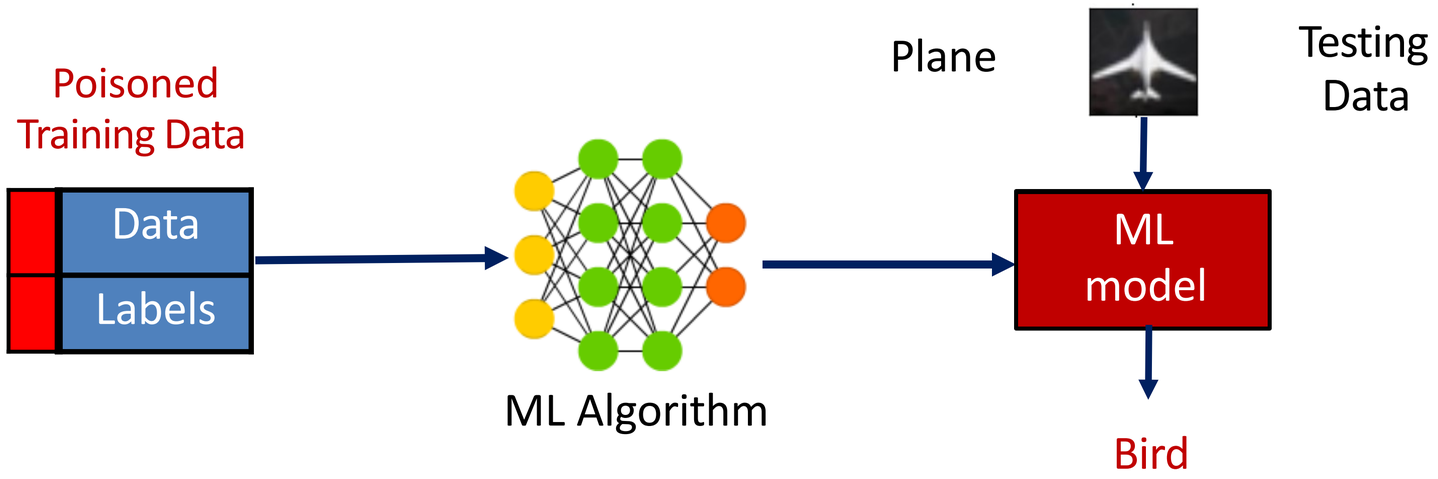
A. Chernikova and A. Oprea.
FENCE: Feasible Evasion Attacks on Neural Networks in Constrained Environments
http://arxiv.org/abs/1909.10480, 2019.

Poisoning Attacks

- Microsoft Industry Survey: Poisoning is top concern
  - Kumar et al. *Adversarial Machine Learning – Industry Perspective*. 2020
- Supply Chain vulnerabilities started to gain attention (SolarWinds attack)



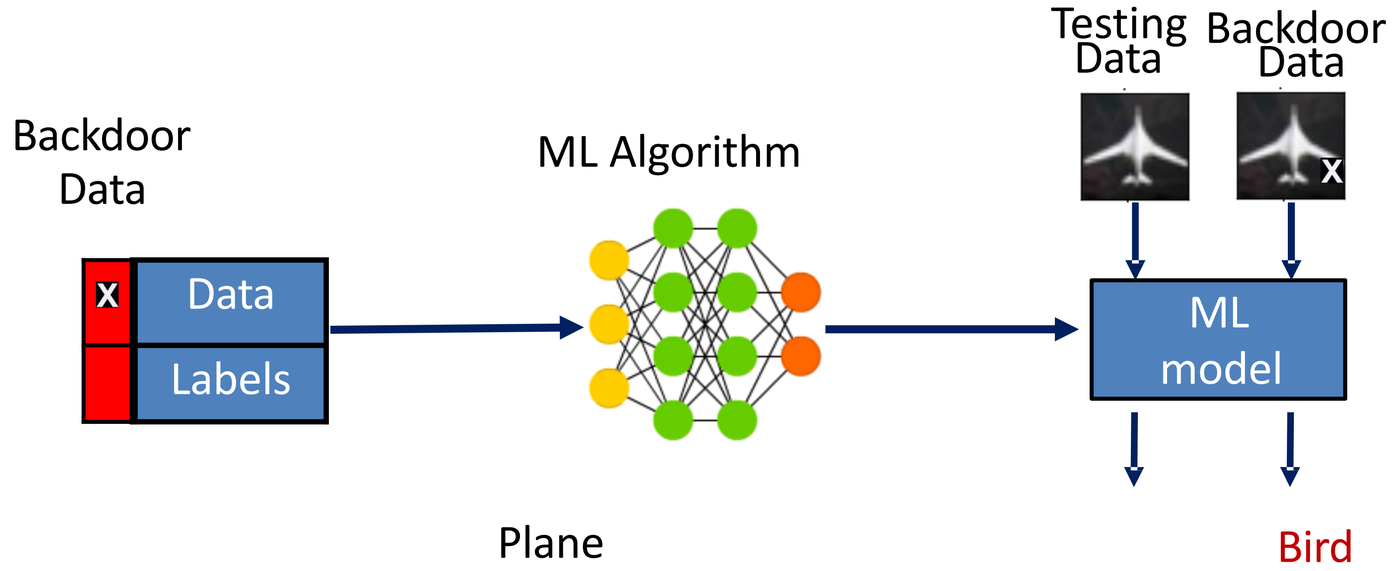
Poisoning Availability Attacks



- Attacker Objective:
  - Corrupt the predictions by the ML model **significantly**
- Attacker Capability:
  - Insert fraction of poisoning points in training
  - Find the points that cause the maximum impact

M. Jagielski, A. Oprea, B. Biggio, C. Liu, C. Nita-Rotaru, and B. Li. *Manipulating Machine Learning: Poisoning Attacks and Countermeasures for Regression Learning*. In IEEE S&P 2018

Backdoor Poisoning Attacks

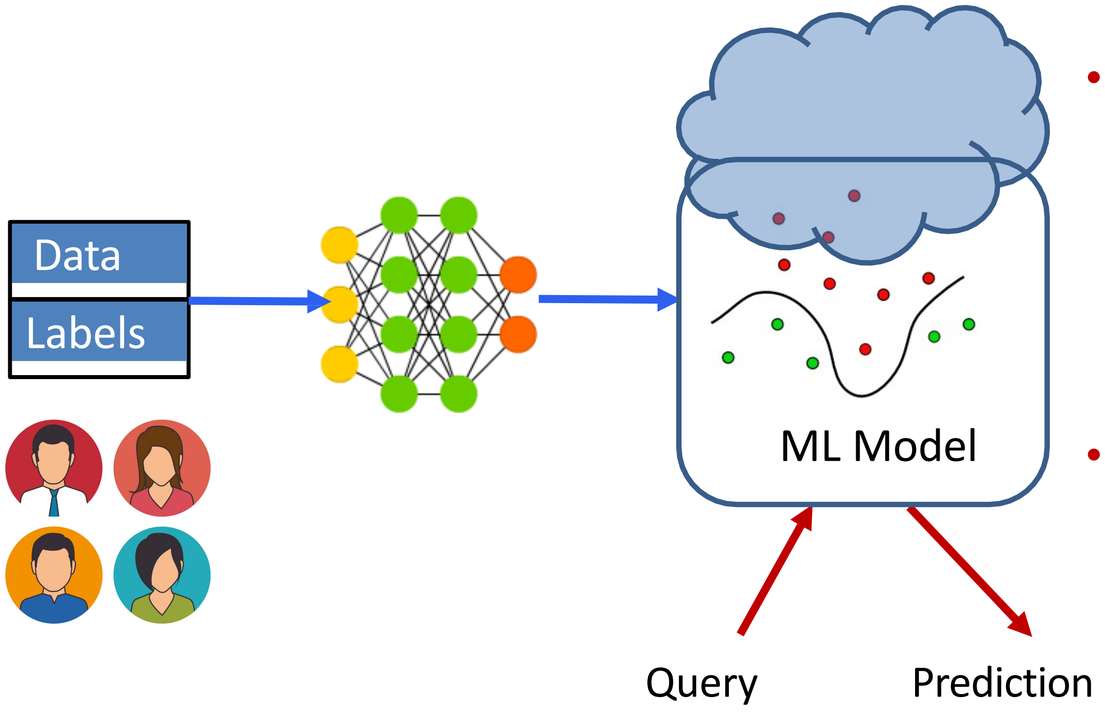


- Attacker Objective:
  - Prediction on clean data is unchanged
  - Change prediction of *backdoor data* in testing
- Attacker Capability:
  - Add backdoored poisoning points in training
  - Add backdoor pattern in testing
- [Gu et al. 17], [Chen et al. 17], [Turner et al. 18], [Shafahi et al. 18]

BadNets

Gu et al. *BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain*. 2017. https://arxiv.org/abs/1708.06733

Privacy Attacks on ML



- Reconstruction attacks: Extract sensitive attributes
  - [Dinur and Nissim 2003]
- Membership Inference: Determine if sample was in training
  - [Shokri et al. 2017], [Yeom et al. 2018], [Hayes et al. 2019], [Jayaraman et al. 2020]
- Model Extraction: Determine model architecture and parameters
  - [Tramer et al. 2016], [Jagielski et al. 2020], [Chandrasekaran et al. 2020]
- Memorization: Determine if model memorizes training data
  - [Carlini et al. 2021]

Q&A