Week 12: Lecture B
Side Channels & Hardware Security

Thursday, November 13, 2025

Announcements

- **Project 3** grades are now available on **Canvas**

- Think we made an error? Request a regrade!
  - Valid regrade requests:
    - You have verified your solution is correct (i.e., we made an error in grading)

Project 3 Regrade Requests (see **Piazza** pinned link):
Submit by **11:59 PM** on **Monday 11/17** via **Google Form**

Announcements

- **Project 4: NetSec** released
  - **Deadline:** Thursday, December 4th by 11:59PM

Project 4: Network Security

Deadline: **Thursday, December 4 by 11:59PM.**

Before you start, review the course syllabus for the Lateness, Collaboration, and Ethical Use policies.

You may optionally work alone, or in teams of **at most two** and submit **one project per team**. If you have difficulties forming a team, post on **Piazza's Search for Teammates** forum. Note that the final exam will cover project material, so you and your partner should collaborate on each part.

The code and other answers your group submits must be entirely your own work, and you are bound by the University's Student Code. You may consult with other students about the conceptualization of the project and the meaning of the questions, but you may not look at any part of someone else's solution or collaborate with anyone outside your group. You may consult published references, provided that you appropriately cite them (e.g., in your code comments). **Don't risk your grade and degree by cheating!**

Complete your work in the **CS 4440 VM**—we will use this same environment for grading. You may not use any **external dependencies**. Use only default Python 3 libraries and/or modules we provide you.

Helpful Resources

- The CS 4440 Course Wiki
- VM Setup and Troubleshooting
- Terminal Cheat Sheet

Table of Contents:

- Helpful Resources
- Introduction
- Objectives
- Start by reading this!
  - Packet Traces
  - Attack Template
  - Wireshark
- Part 1: Defending Networks
  - Password Cracking
  - Port Scanning
  - Anomalous Activity
  - What to Submit
- Part 2: Attacking Networks
  - Plaintext Credentials
  - Encoded Credentials
  - Accessed URLs
  - Extra Credit: Transferred Files
  - What to Submit
- Submission Instructions

Project 4 Progress

Working on Part 1	0%
Finished Part 1, working on Part 2	0%
Finished both Part 1 and Part 2	0%
None of the above	0%

Final Exam

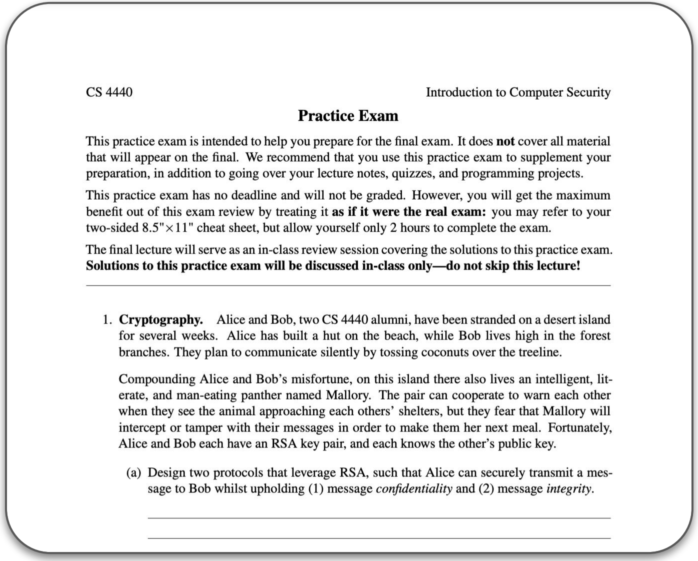
- **Save the date: 1–3PM** on **Wednesday, December 10**
  - **CDA accommodations:** schedule exam via CDA Portal

- **High-level details** (more to come):
  - One exam covering all course material
  - Similar to project/quiz/lecture exercises

- **Cheat Sheet**
  - **One 8.5”x11” paper** with handwritten/typed notes on **both** sides
  - **Suggestion:** Don’t just use someone else’s—you’ll learn better making **your own**!
  - **Suggestion:** Don’t just paste lecture slides—you’ll learn better by **writing/typing** it!

Practice Exam

- **Practice Exam** released
  - See **Assignments** page on the CS 4440 website
- **Final lecture** will serve as a **review session**
  - Solutions discussed **in-class only**—don't skip!



CS 4440 Introduction to Computer Security

Practice Exam

This practice exam is intended to help you prepare for the final exam. It does **not** cover all material that will appear on the final. We recommend that you use this practice exam to supplement your preparation, in addition to going over your lecture notes, quizzes, and programming projects.

This practice exam has no deadline and will not be graded. However, you will get the maximum benefit out of this exam review by treating it **as if it were the real exam:** you may refer to your two-sided 8.5"×11" cheat sheet, but allow yourself only 2 hours to complete the exam.

The final lecture will serve as an in-class review session covering the solutions to this practice exam. **Solutions to this practice exam will be discussed in-class only—do not skip this lecture!**

1. **Cryptography.** Alice and Bob, two CS 4440 alumni, have been stranded on a desert island for several weeks. Alice has built a hut on the beach, while Bob lives high in the forest branches. They plan to communicate silently by tossing coconuts over the treeline.

   Compounding Alice and Bob's misfortune, on this island there also lives an intelligent, literate, and man-eating panther named Mallory. The pair can cooperate to warn each other when they see the animal approaching each others' shelters, but they fear that Mallory will intercept or tamper with their messages in order to make them her next meal. Fortunately, Alice and Bob each have an RSA key pair, and each knows the other's public key.

   (a) Design two protocols that leverage RSA, such that Alice can securely transmit a message to Bob whilst upholding (1) message *confidentiality* and (2) message *integrity*.

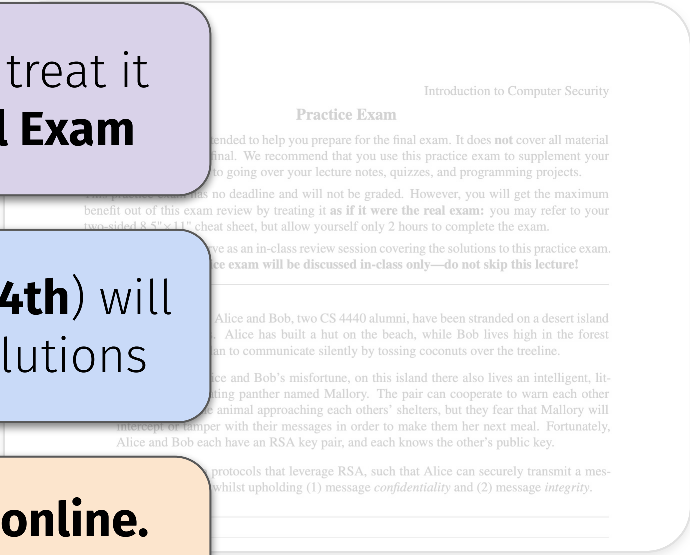
Practice Exam

- **Practice Exam** re
  - See **Assignmen**
- **Final lecture** will serve as a **review session**
  - Solutions discu

To get the most out of this, treat it just **as you would the Final Exam**

Last lecture (**Thursday, Dec. 4th**) will go over the exam review solutions

Solutions won't be posted online.
(Reminder: attendance/participation makes up **5%** of your course grade)


Introduction to Computer Security

Practice Exam

…ended to help you prepare for the final exam. It does **not** cover all material …final. We recommend that you use this practice exam to supplement your …to going over your lecture notes, quizzes, and programming projects.

…has no deadline and will not be graded. However, you will get the maximum benefit out of this exam review by treating it **as if it were the real exam:** you may refer to your two-sided 8.5″×11″ cheat sheet, but allow yourself only 2 hours to complete the exam.

…ve as an in-class review session covering the solutions to this practice exam. …**ce exam will be discussed in-class only—do not skip this lecture!**

…Alice and Bob, two CS 4440 alumni, have been stranded on a desert island … Alice has built a hut on the beach, while Bob lives high in the forest …an to communicate silently by tossing coconuts over the treeline.

…ice and Bob's misfortune, on this island there also lives an intelligent, lit-…ting panther named Mallory. The pair can cooperate to warn each other …e animal approaching each others' shelters, but they fear that Mallory will intercept or tamper with their messages in order to make them her next meal. Fortunately, Alice and Bob each have an RSA key pair, and each knows the other's public key.

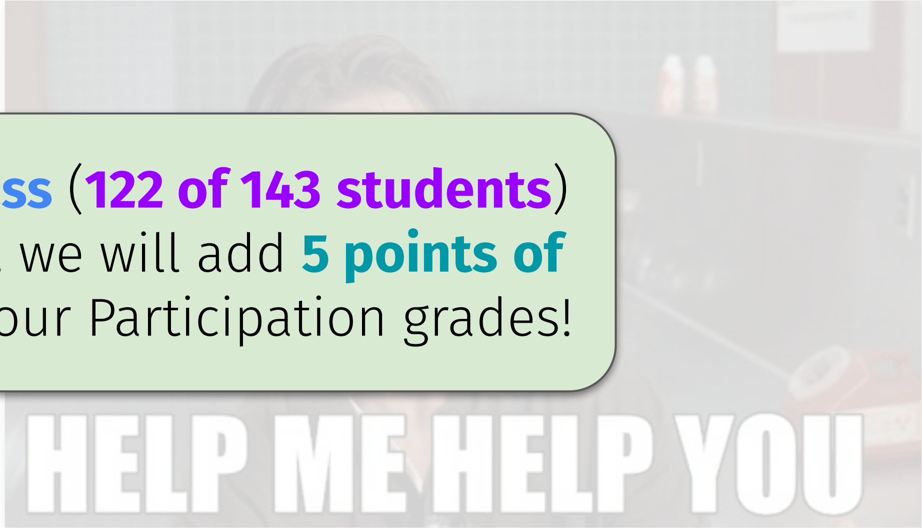
…protocols that leverage RSA, such that Alice can securely transmit a mes-…whilst upholding (1) message *confidentiality* and (2) message *integrity*.

End-of-semester Course Evals

- **I want your feedback!**
  - 3rd time teaching this course 😃
  - **Help me improve the class!**
- Due by **December 15th**
  - https://scf.utah.edu
  - **Please please please!**

End-of-semester Course Evals

- **I want your feedback!**
  - 3rd time teaching this course 😃
  - **Help me improve the class!**
- Due by **Dece**
  - https://sc
  - **Please pl**



If 85% of the class (**122 of 143 students**) submits an eval, we will add **5 points of extra credit** to your Participation grades!

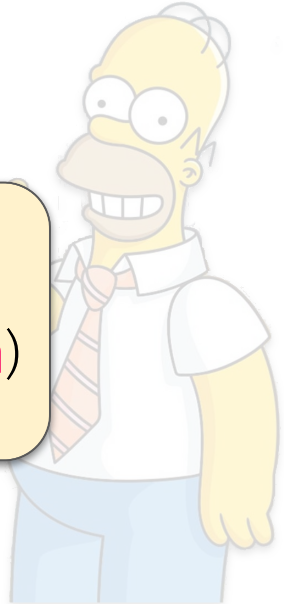
Reminders: Participation Extra Credit

- **Piazza:** 5 points per top-10 student contributors
  - Answering peers' questions
  - Providing helpful resources
- **Wiki Contributions:** 1 point per approved contribution
  - Must be cleared in advance
- **Course Evals:** 5 points if 85% of class submits evals
  - Will be released soon on **scf.utah.edu**

Reminders: Participation Extra Credit

- **Piazza:** 5 points per top-10 student contributors
  - Answering peers' questions
  - Providing helpful resources
- **Wiki Contri**
  - Must be
- **Course Eva**
  - Will be released soon on sci.utah.edu

Final deadline for extra credit will be the last day of class (**Thursday, December 4th**)



Kahlert School of Computing

Graduate Program Open House

Why PhD?

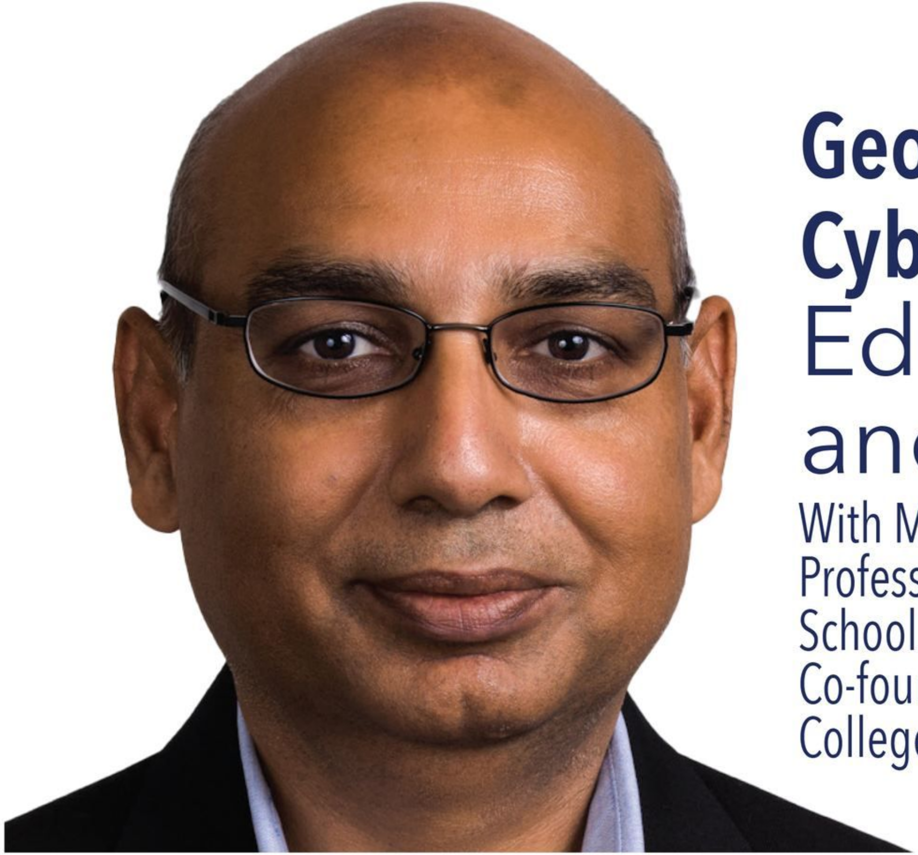
Information session for prospective graduate students

- What to expect from graduate school
- Reasons to pursue graduate career
- Perspective of alumni and current students
- How to prepare your application (and a statement of purpose)

November 14, 3:00pm – 5:00pm

MEB 3147 (LCR) and Zoom (free pizza—please RSVP)

RSVP / Zoom links:



Georgia Tech's Cybersecurity Journey: Education, Research, and Entrepreneurship

With Mustaque Ahamad
Professor, Regents' Entrepreneur, and Interim Chair
School of Cybersecurity and Privacy
Co-founder, Pindrop Security and Codoxo
College of Computing, Georgia Institute of Technology

Kahlert Distinguished Lecture

Announcements

- Instructor on work travel next week
  - Presenting our **GUI fuzzing work** at ASE'25 😀
- Guest lectures planned for both days
  - **Week 13A:** Cyber-physical Systems Security
    - **Guest speaker:** Dr. Luis Garcia (Asst. Prof @ UofU)
  - **Week 13B:** Binary Reverse Engineering
    - **Guest speaker:** Zao Yang (researcher in my group)
- Attendance **not graded** for these lectures…
  - But you should definitely show up
  - **These are major hot topics in security!**

Questions?

Last time on CS 4440…

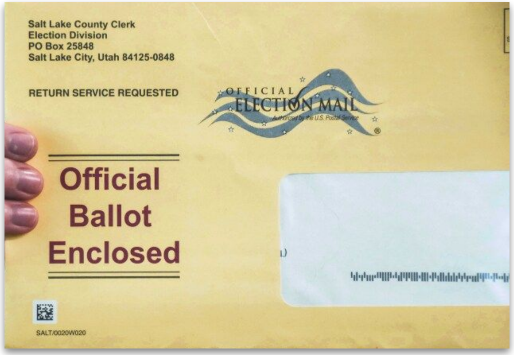
Election Cybersecurity

Requirement #1: Integrity

- **Goals:** outcome matches voter's **intent**
  - Votes are cast as intended
  - Votes are counted as cast

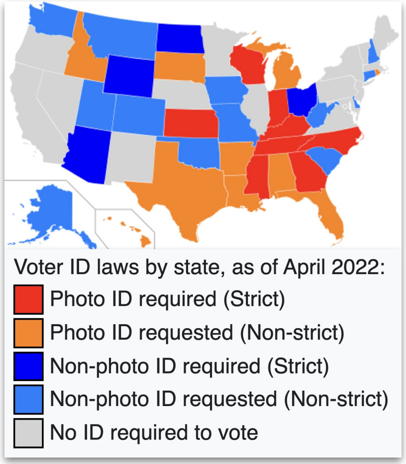
Requirement #2: Confidentiality

- **Goals:** nobody can figure out **how** you voted
  - … even if you try to prove it to them



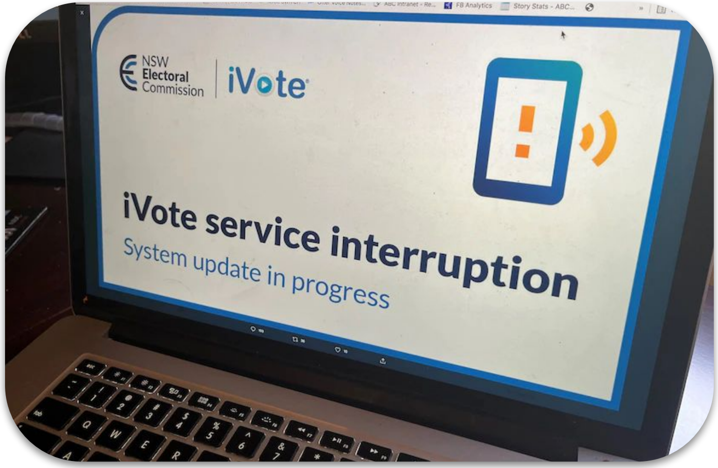
Requirement #3: Authentication

- **Goals:**
  - Only **authorized voters** can cast votes
  - Each voter can cast **at most one** vote

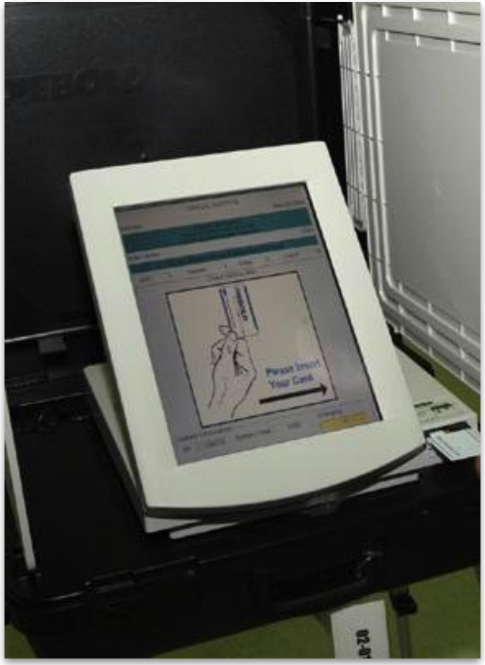


Requirement #4: Availability

- **Goals:**
  - All authorized voters have **opportunity** to vote
  - System is able to **accept all votes** on schedule
  - System can produce results in a **timely manner**



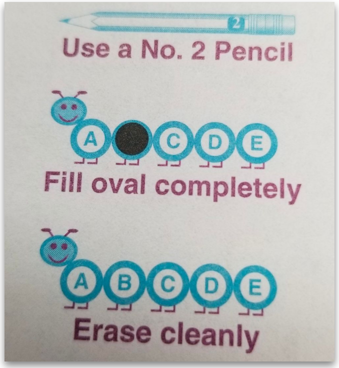
Computer-based Voting Devices

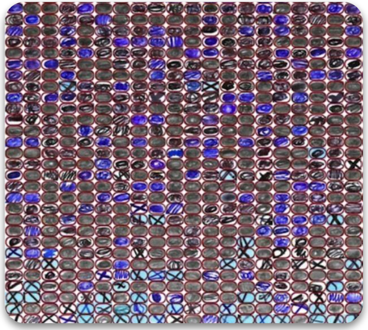


DRE Machine

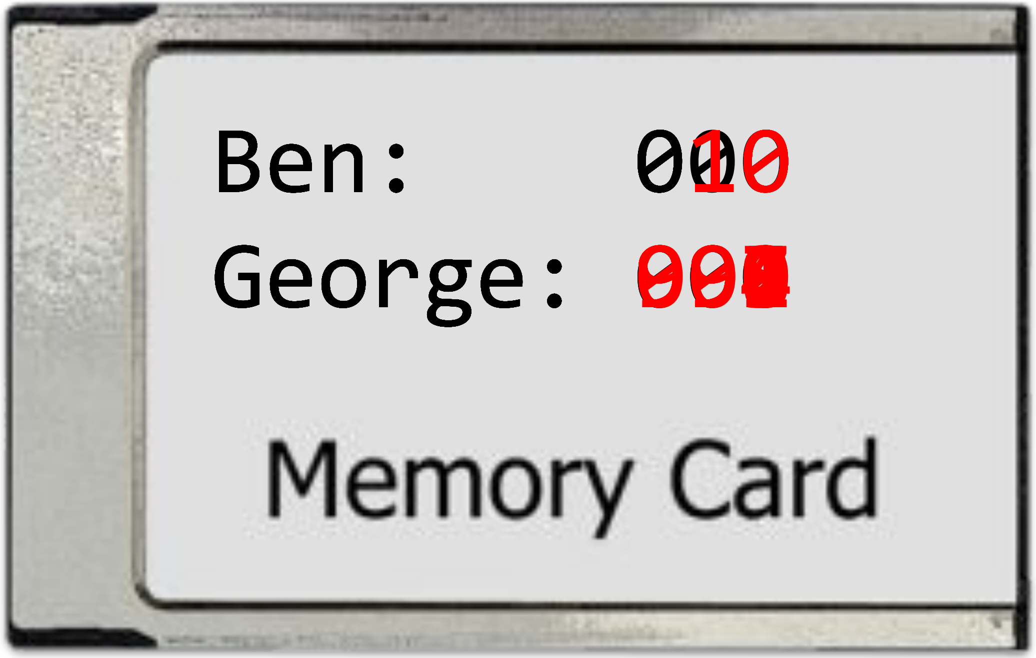
Optical Scanner

Ballot Tampering Attacks





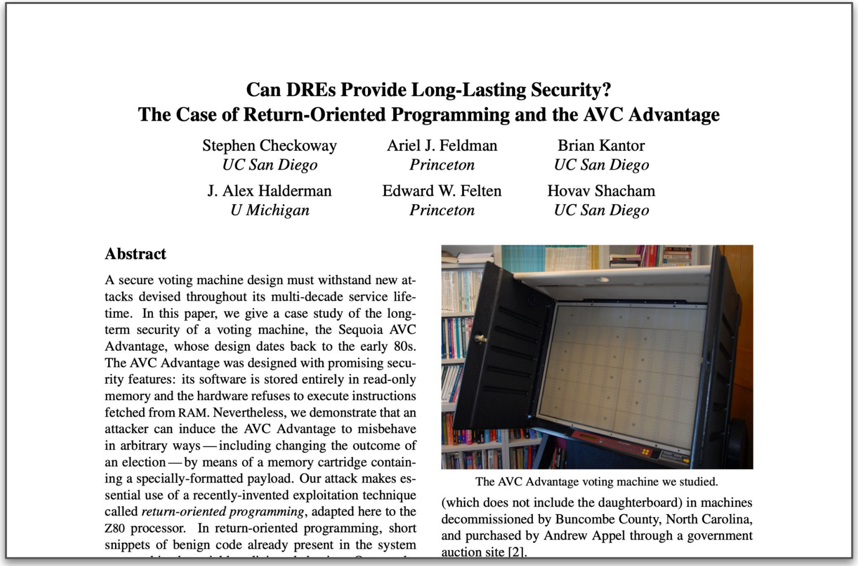
Memory Corruption Attacks



Memory Corruption Attacks

Return-oriented Programming (ROP)
Use code gadgets to achieve functionality

previous frame ptr	string "/bin/sh"
AAAAAAAAA...\0	system()'s first arg
foo()'s first arg	system()'s return addr
foo()'s return addr	Address of system()
main()'s frame ptr	AAAAAAAAAAAAAAAAAAAAAA
Buffer (non-executable)	AAAAAAAAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAAAAAAAA



Can DREs Provide Long-Lasting Security?
The Case of Return-Oriented Programming and the AVC Advantage

Stephen Checkoway, *UC San Diego*; Ariel J. Feldman, *Princeton*; Brian Kantor, *UC San Diego*
J. Alex Halderman, *U Michigan*; Edward W. Felten, *Princeton*; Hovav Shacham, *UC San Diego*

Abstract

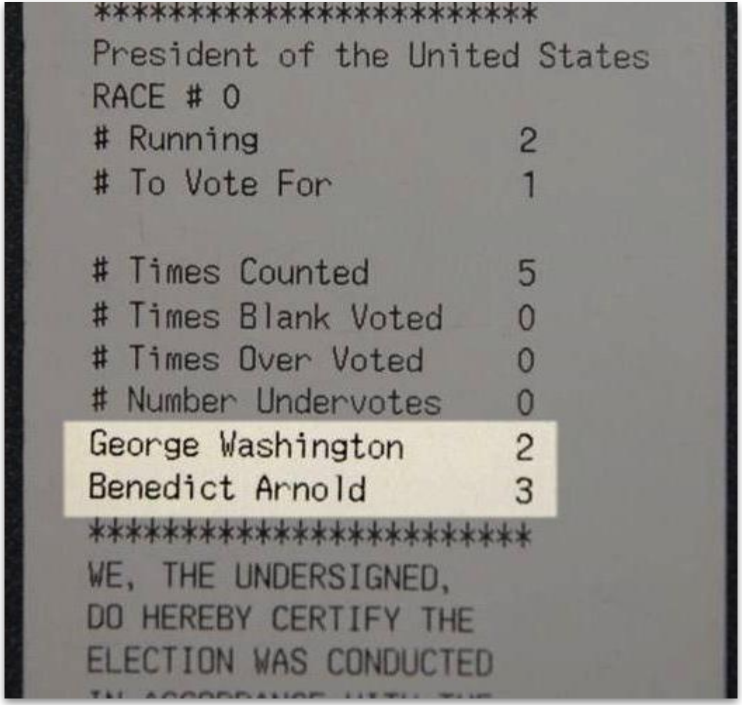
A secure voting machine design must withstand new attacks devised throughout its multi-decade service lifetime. In this paper, we give a case study of the long-term security of a voting machine, the Sequoia AVC Advantage, whose design dates back to the early 80s. The AVC Advantage was designed with promising security features: its software is stored entirely in read-only memory and the hardware refuses to execute instructions fetched from RAM. Nevertheless, we demonstrate that an attacker can induce the AVC Advantage to misbehave in arbitrary ways — including changing the outcome of an election — by means of a memory cartridge containing a specially-formatted payload. Our attack makes essential use of a recently-invented exploitation technique called *return-oriented programming*, adapted here to the Z80 processor. In return-oriented programming, short snippets of benign code already present in the system

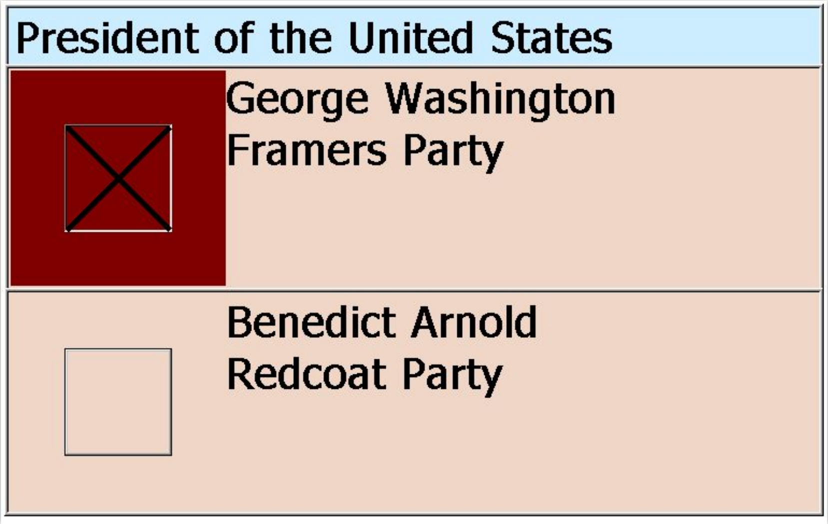
The AVC Advantage voting machine we studied.

(which does not include the daughterboard) in machines decommissioned by Buncombe County, North Carolina, and purchased by Andrew Appel through a government auction site [2].

Code Insertion Attacks

Code Insertion Attacks





Voting Machine Security

LILY HAY NEWMAN SECURITY 08.28.18 11:04 AM

VOTING MACHINES ARE STILL ABSURDLY VULNERABLE TO ATTACKS

BILL CLARK/GETTY IMAGES

WHILE RUSSIAN INTERFERENCE operations in the 2016 US presidential elections focused on misinformation and targeted hacking, officials have scrambled ever since to shore up the nation's vulnerable election infrastructure. New research, though, shows they haven't done nearly enough, particularly when it comes to voting machines.

Voting Machine Manual Instructed Election Officials to Use Weak Passwords

A vendor manual for voting machines used in about ten states shows the vendor instructed customers to use trivial, easy to crack passwords and to re-use the passwords when changing log-in credentials.

SHARE TWEET

Image: Shutterstodck

States and counties have had two years since the 2016 presidential election to educate themselves about security best practices and to fix security vulnerabilities in their election systems and processes. But despite widespread concerns about election interference from state-sponsored hackers in Russia and elsewhere, apparently not everyone received the memo about security, or read it.

An election security expert who has done risk-assessments in several states since

Latest

The Socialist Memelords Radicalizing Instagram

16 minutes ago

This Guy Wants to Open a DIY Tesla Repair Shop

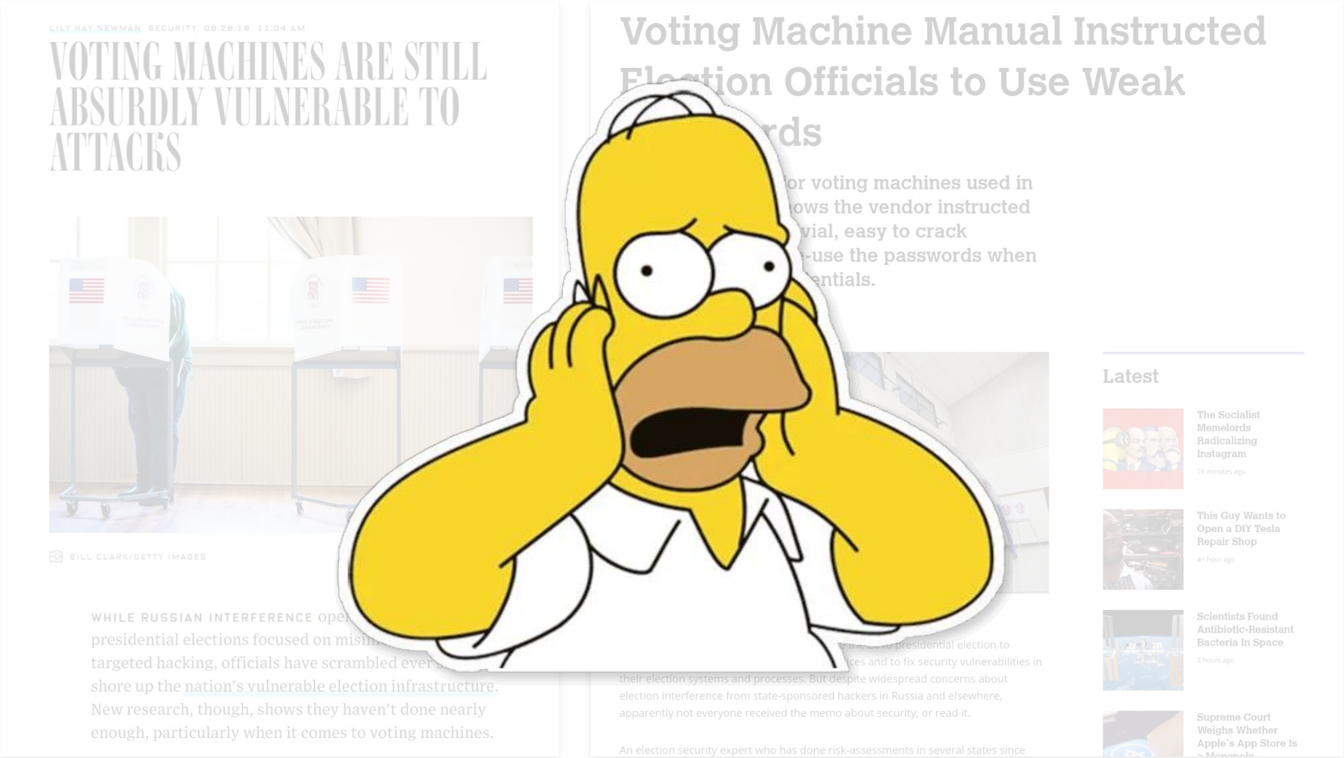
an hour ago

Scientists Found Antibiotic-Resistant Bacteria In Space

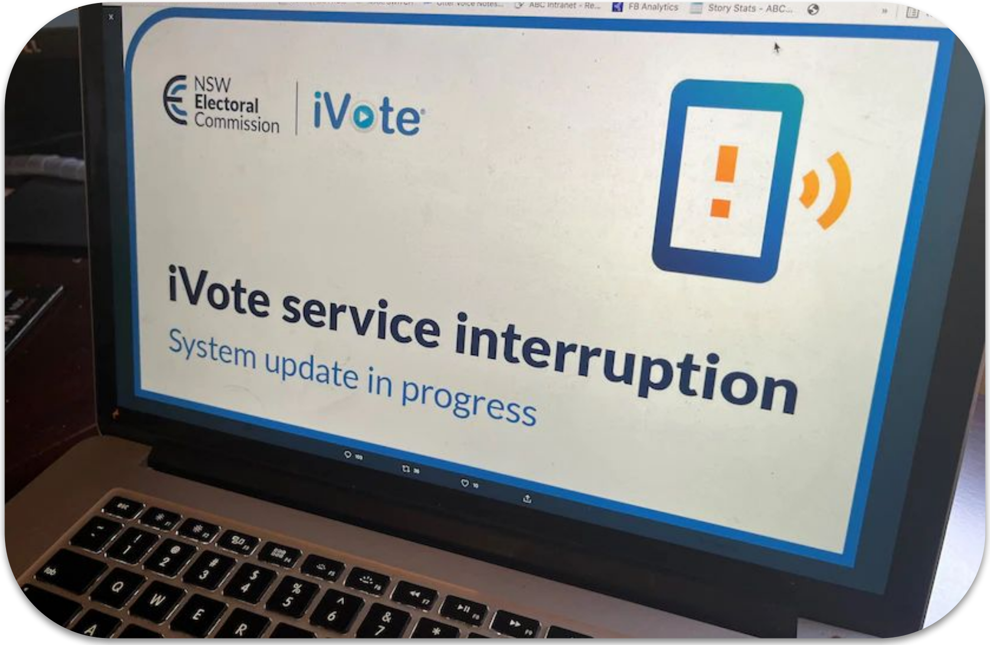
2 hours ago

Supreme Court Weighs Whether Apple's App Store Is a Monopoly

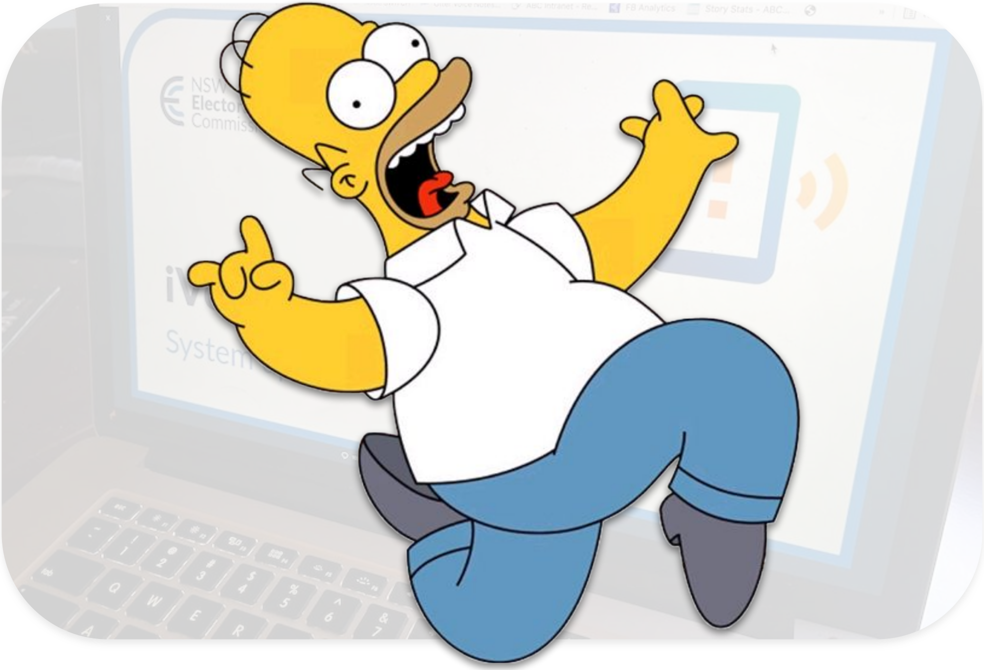
Voting Machine Security



Internet-based Voting Security



Internet-based Voting Security



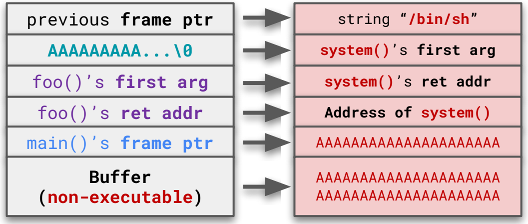
Questions?

This time on CS 4440…

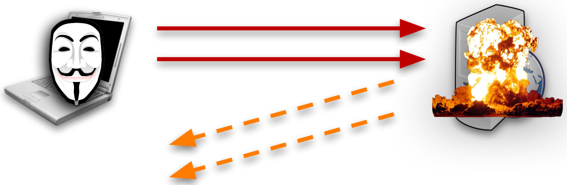
Side Channels
Hardware Security
Hardware Supply Chain Attacks

Exploitable Security Flaws

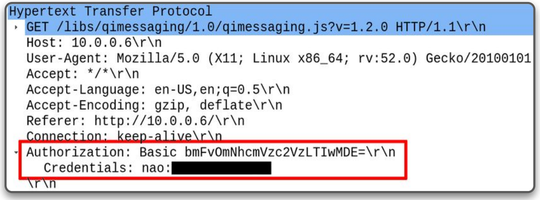
- **So far, we have studied attacks that exploit design flaws**



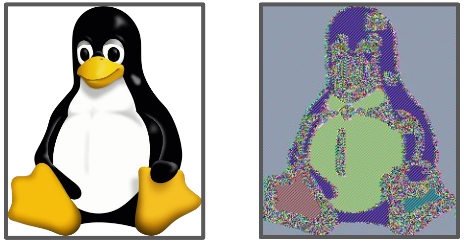
Buffer Overflows



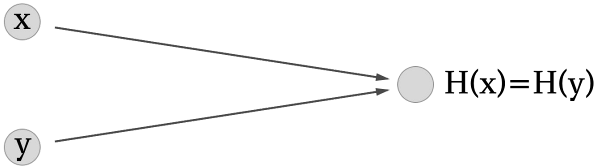
SYN Flooding


```
Hypertext Transfer Protocol
  GET /libs/qimessaging/1.0/qimessaging.js?v=1.2.0 HTTP/1.1\r\n
  Host: 10.0.0.6\r\n
  User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101
  Accept: */*\r\n
  Accept-Language: en-US,en;q=0.5\r\n
  Accept-Encoding: gzip, deflate\r\n
  Referer: http://10.0.0.6/\r\n
  Connection: keep-alive\r\n
  Authorization: Basic bmFvOmNhcmVzc2VzLTIwMDE=\r\n
    Credentials: nao:[redacted]
  \r\n
```

Sniffing Unencrypted Data



ECB Diffusion Analysis



Hash Collisions

```
http://cs4440.eng.utah
.edu/project3/search
?q=%3Cscript%3E...
```

Cross-site Scripting

Exploitable Security Flaws

- So far, we have studied attacks that exploit design flaws

What if I told you that **implementation flaws** can be just as severe?

ECB Diffusion Analysis

Hash Collisions

Cross-site Scripting

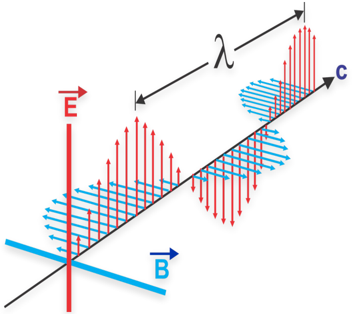
Side Channel Attacks

Side Channel Attacks

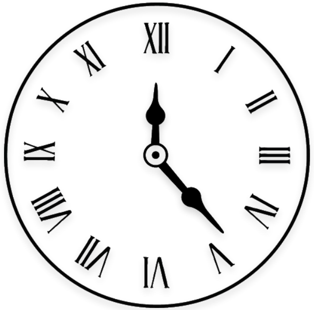
"Any attack based on **extra information** that can be **gathered** because of the fundamental way a computer protocol or algorithm is **implemented**, or minor, but potentially devastating, mistakes or oversights in the implementation."

Side Channels

- What are some potential sources of **indirect info** emitted by your computer?
  - **Additional channels** of information beyond what is directly visible/accessible to you



Emitted Radiation

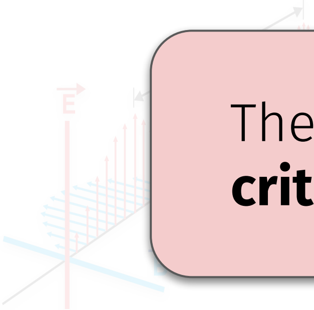
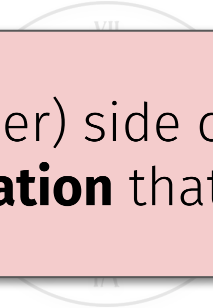


Execution Time

Power Consumption

Side Channels

- What are some potential sources of **indirect info** emitted by your computer?
  - **Additional channels** of information beyond what is directly visible/accessible to you

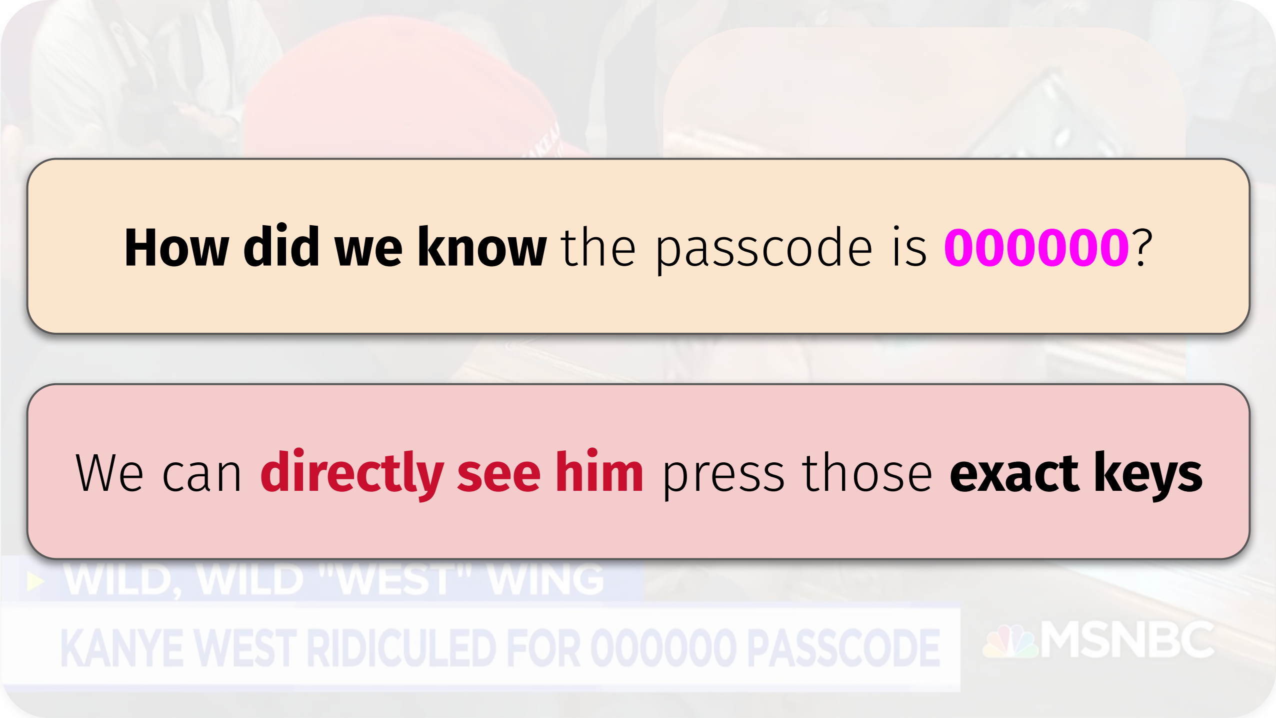

Emitted Radiation | Execution Time | Power Consumption

These (and other) side channels reveal **critical information** that is **exploitable**

Optical and Acoustic Side Channels

Stealing Passwords

Stealing Passwords



How did we know the passcode is **000000**?

We can **directly see him** press those **exact keys**

Stealing Passwords

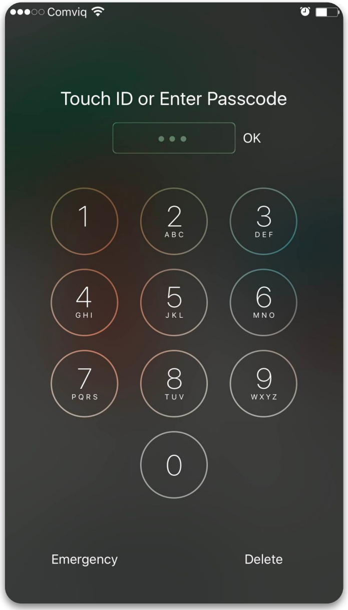
- What if we **can't** **directly see** keys that someone is pressing?

Stealing Passwords

- What if we **can't directly see** keys that someone is pressing?
- **Optical side channel:**
  - Capture visible **hand movements**

Stealing Passwords

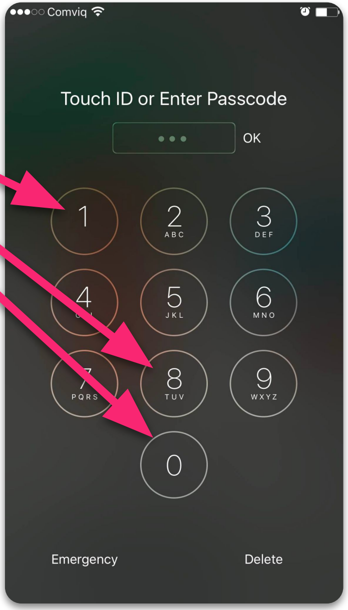
- What if we **can't directly see** keys that someone is pressing?
- **Optical side channel:**
  - Capture visible **hand movements**
  - Assume attacker **knows (or can easily guess)** the key interface



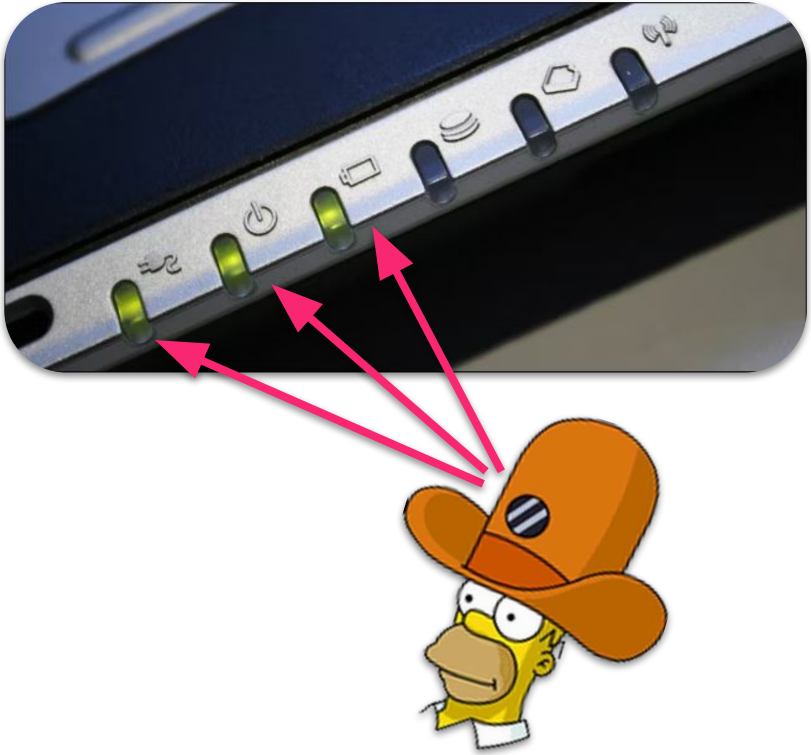
Stealing Passwords

- What if we **can't directly see** keys that someone is pressing?

- **Optical side channel:**
  - Capture visible **hand movements**
  - Assume attacker **knows (or can easily guess)** the key interface
  - Attacker **maps movements** to pressed keys on the interface



Stealing Information



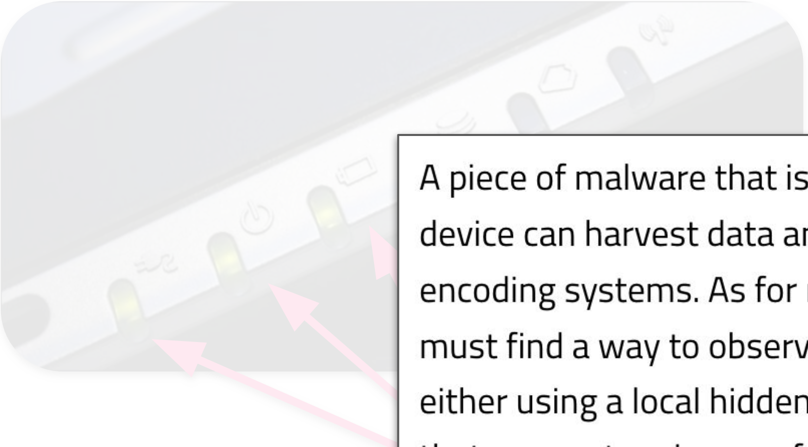
Hard Drive LED Allows Data Theft From Air-Gapped PCs

Researchers at Ben-Gurion University of the Negev in Israel have disclosed yet another method that can be used to exfiltrate data from air-gapped computers, and this time it involves the activity LED of hard disk drives (HDDs).

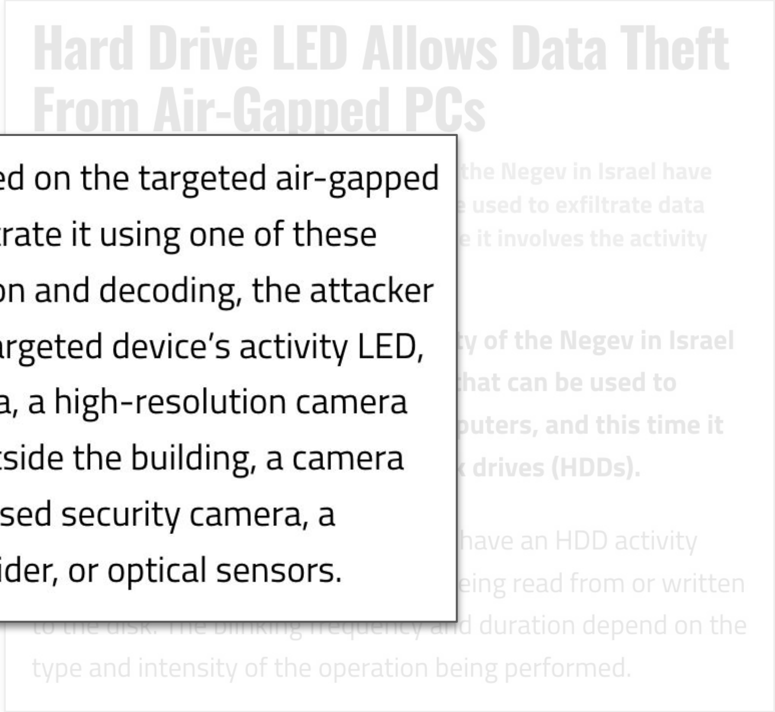
Researchers at Ben-Gurion University of the Negev in Israel have disclosed yet another method that can be used to exfiltrate data from air-gapped computers, and this time it involves the activity LED of hard disk drives (HDDs).

Many desktop and laptop computers have an HDD activity indicator, which blinks when data is being read from or written to the disk. The blinking frequency and duration depend on the type and intensity of the operation being performed.

Stealing Information



A piece of malware that is installed on the targeted air-gapped device can harvest data and exfiltrate it using one of these encoding systems. As for reception and decoding, the attacker must find a way to observe the targeted device's activity LED, either using a local hidden camera, a high-resolution camera that can capture images from outside the building, a camera mounted on a drone, a compromised security camera, a camera carried by a malicious insider, or optical sensors.



Hard Drive LED Allows Data Theft From Air-Gapped PCs

the Negev in Israel have
used to exfiltrate data
e it involves the activity

ty of the Negev in Israel
hat can be used to
puters, and this time it
drives (HDDs).

have an HDD activity
eing read from or written
d duration depend on the
type and intensity of the operation being performed.

Acoustic Side Channels

- **Sound** can leak information, too!
  - Keyboard enthusiasts beware

Acoustic Side Channels

- **Sound** can leak information, too!
  - Keyboard enthusiasts beware

- **Build model of key press noises**
  - Model refinement:
    - **???**

Acoustic Side Channels

- **Sound** can leak information, too!
  - Keyboard enthusiasts beware

- **Build model of key press noises**
  - Model refinement:
    - Consider microphone
    - Remove ambient noise
  - Use model to infer entered data
    - Passwords
    - Usernames
    - Phone numbers

Questions?

Timing Side Channels: Password Checking

Password Checking

- **Password verification**—how would you implement this?

```
bool checkPW(char *testPW, char *realPW, int len) {
    for (int i = 0; i < len; i++) {
        if (testPW[i] != realPW[i]) {
            return false;
        }
    }

    return true;
}
```

Analogous to memcmp( )

Password Checking

- **Password verification**—how would you implement this?

```
bool checkPW(char *testPW, char *realPW, int len) {

    for (int i = 0; i < len; i++) {

        if (testPW[i] != realPW[i]) {
            return false;
        }
    }

    return true;
}
```

Analogous to memcmp( )

Does this password checking code reveal a **security flaw**?

Does this password-checking code reveal a security flaw?

No—an attacker could only brute-force guess!

0%

Yes—the design is vulnerable (e.g., buffer overflow).

0%

None of the above

0%

```
bool checkPW(char *testPW, char *realPW, int len) {

    for (int i = 0; i < len; i++) {

        if (testPW[i] != realPW[i]) {
            return false;
        }
    }

    return true;
}
```

Password Checking

- **Password verification**—how would you implement this?

```
bool checkPW(char *testPW, char *realPW, int len) {

    for (int i = 0; i < len; i++) {

        if (testPW[i] != realPW[i]) {
            return false;
        }
    }

    return true;
}
```

Password Login Attempts:

ABCDEFGH == PASSWORD

- ???

Password Checking

- **Password verification**—how would you implement this?

```
bool checkPW(char *testPW, char *realPW, int len) {

    for (int i = 0; i < len; i++) {

        if (testPW[i] != realPW[i]) {
            return false;
        }
    }

    return true;
}
```

Password Login Attempts:

ABCDEFGH == PASSWORD

- **False** on first iteration

PASSEFGH == PASSWORD

- **???**

Password Checking

- **Password verification**—how would you implement this?

```
bool checkPW(char *testPW, char *realPW, int len) {

    for (int i = 0; i < len; i++) {

        if (testPW[i] != realPW[i]) {
            return false;
        }
    }

    return true;
}
```

Password Login Attempts:

ABCDEFGH == PASSWORD

- **False** on first iteration

PASSEFGH == PASSWORD

- **True** on iterations **1–4**
- **False** on **fifth** iteration

More code executed for a **correct** symbol!

Password Checking

How can this **side channel** be **exploited**?

Password Checking

How can this **side channel** be **exploited**?

Attacker: ABCDEF

Password Checking

How can this **side channel** be **exploited**?

Attacker: ABCDEF

Server: `False`
Server took **1ms** to respond

Password Checking

How can this **side channel** be **exploited**?

"C" took longer!

Attacker: ABCDEF

Server: `False`
Server took **1ms** to respond

Attacker: CBCDEF

Server: `False`
Server took **2ms** to respond

Password Checking

How can this **side channel** be **exploited**?

Attacker: `CRCDEF`

Server: `False`
Server took **2ms** to respond

Password Checking

How can this **side channel** be **exploited**?

"CHI"...
Getting warmer!

Attacker: CRCDEF

Server: False
Server took 2ms to respond

Attacker: CHIDEF

Server: False
Server took 4ms to respond

Password Checking

How can this **side channel** be **exploited**?

Attacker: `CHIEFS`

Server: `True`
Server took **7ms** to respond

Password Checking

How can this **side channel** be **exploited**?

Attacker: `CHIEFS`

Server: `True`
Server took **7ms** to respond

Through **timing analysis**, attacker can infer the **correctness** of individual **password symbols**!

Password Checking

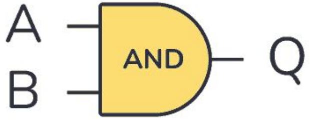
- **Solution:**
  - **???**

Password Checking

- **Solution:**
  - **Constant-time** implementation (e.g., using bitwise **AND**-ing)

```
bool checkPW(char *testPW, char *realPW, int len) {
    bool result = 1; // integer equiv of "true"
    for (int i = 0; i < len; i++) {
        result &= ca[i] == cb[i];
        return result;
    }
}
```

Guess: PASSEFGH
Bit: 11110000
Result: **False**



A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

Password Checking

- **Solution:**
  - **Constant-time** implementation (e.g., using bitwise **AND**-ing)

```
bool checkPW(char *testPW, char *realPW, int len) {
    bool result = 1; // integer equiv of "true"
    for (int i = 0; i < len; i++) {
        result &= ca[i] == cb[i];
        return result;
    }
}
```

Guess: PASSEFGH
Bit: 11110000
Result: **False**

Password Login Attempts:

ABCDEFGH == PASSWORD
- **False** on **last** iteration

PASSEFGH == PASSWORD
- **False** on **last** iteration

PASSWORD == PASSWORD
- **True** on **last** iteration

True and **False** run for **identical time**!

Password Checking

- **Implications:**
  - **???**

Password Checking

- **Implications:**
  - **Never** use **timing-unsafe string compares** when handling **sensitive data**!

FreeBSD Manual Pages

timingsafe_bcmp | man | apropos

3 - Subroutines | FreeBSD 13.1-RELEASE and Ports | All Architectures | html

home | help

```
TIMINGSAFE_BCMP(3)      FreeBSD Library Functions Manual      TIMINGSAFE_BCMP(3)

NAME
     timingsafe_bcmp, timingsafe_memcmp -- timing-safe byte sequence compar-
     isons

SYNOPSIS
     #include <string.h>

     int
     timingsafe_bcmp(const void *b1, const void *b2, size_t len);

     int
     timingsafe_memcmp(const void *b1, const void *b2, size_t len);
```

FreeBSD Manual Pages

consttime_memequal | man | apropos

All Sections | NetBSD 7.0 | All Architectures | html

home | help

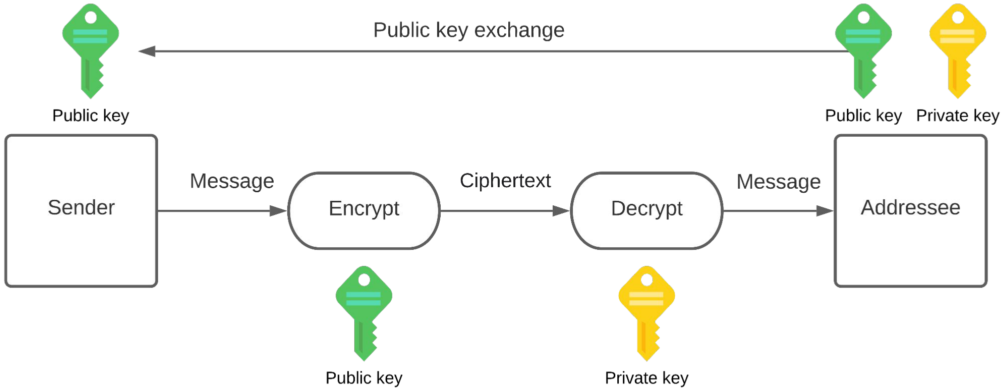
```
CONSTTIME_MEMEQUAL(3)    BSD Library Functions Manual    CONSTTIME_MEMEQUAL(3)

NAME
     consttime_memequal -- compare byte strings for equality without timing
     leaks

LIBRARY
     Standard C Library (libc, -lc)

SYNOPSIS
     #include <string.h>

     int
     consttime_memequal(void *b1, void *b2, size_t len);
```

Questions?

Timing (and Power) Side Channels: RSA Encryption

Recap: RSA for Confidentiality

- **Summary:**
  - Encrypt with **???**
  - Decrypt with **???**

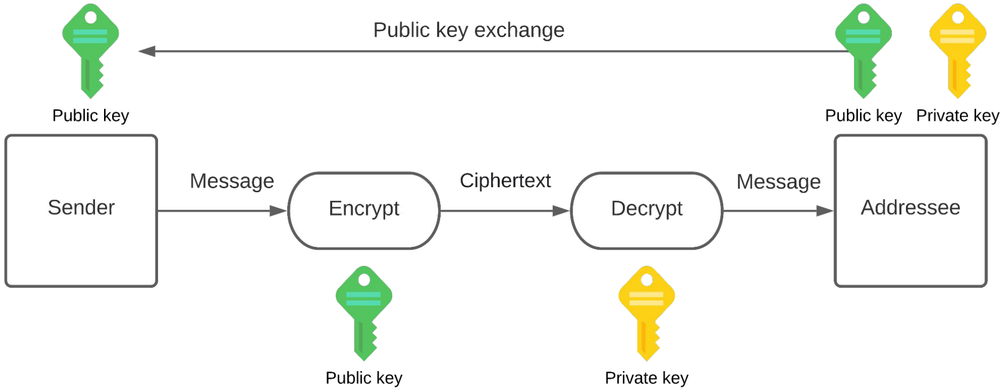


Recap: RSA for Confidentiality

- **Summary:**
  - Encrypt with **public key**
  - Decrypt with **private key**
  - **Public key** = (**e**,**N**)
  - **Private key** = (**d**,**N**)

- To **encrypt:**
  - $E(x) = x^{e} \bmod N$

- To **decrypt:**
  - $D(x) = x^{d} \bmod N$



Recap: RSA for Confidentiality

- **Summary:**
  - Encrypt with **public key**
  - Decrypt with **private key**
  - **Public key** = (e,N)
  - **Private key** = (d,N)

- To **encrypt:**
  - $E(x) = \mathbf{x}^{e} \bmod \mathbf{N}$

- To **decrypt:**
  - $D(x) = \mathbf{x}^{d} \bmod \mathbf{N}$

Modular exponentiation must be implemented **efficiently**

Modular Exponentiation

- **Decryption:** $D(x) = C^{\text{privKey}} \bmod N$

```
x = C

for (int i = 0; i < len; i++){

    x = (x·x) mod(N)

    if (privKey[i] == 1){
        x = (x·C) mod(N)
    }
}
return x
```

Does this decryption code reveal a **security flaw**?

Modular Exponentiation

- **Decryption:** $D(x) = C^{\text{privKey}} \bmod N$

```
x = C

for (int i = 0; i < len; i++){

    x = (x·x) mod(N)

    if (privKey[i] == 1){
        x = (x·C) mod(N)
    }
}
return x
```

Bit-specific Operations:

`privKey[i] == 0`
1. Find square of **x**
2. Take modulo **N**

`privKey[i] == 1`
1. Find square of **x**
2. Take modulo **N**

Modular Exponentiation

- **Decryption:** $D(x) = C^{privKey} \bmod N$

```
x = C

for (int i = 0; i < len; i++){

    x = (x·x) mod(N)

    if (privKey[i] == 1){
        x = (x·C) mod(N)
    }
}
return x
```

Bit-specific Operations:

`privKey[i] == 0`

1. Find square of **x**
2. Take modulo **N**

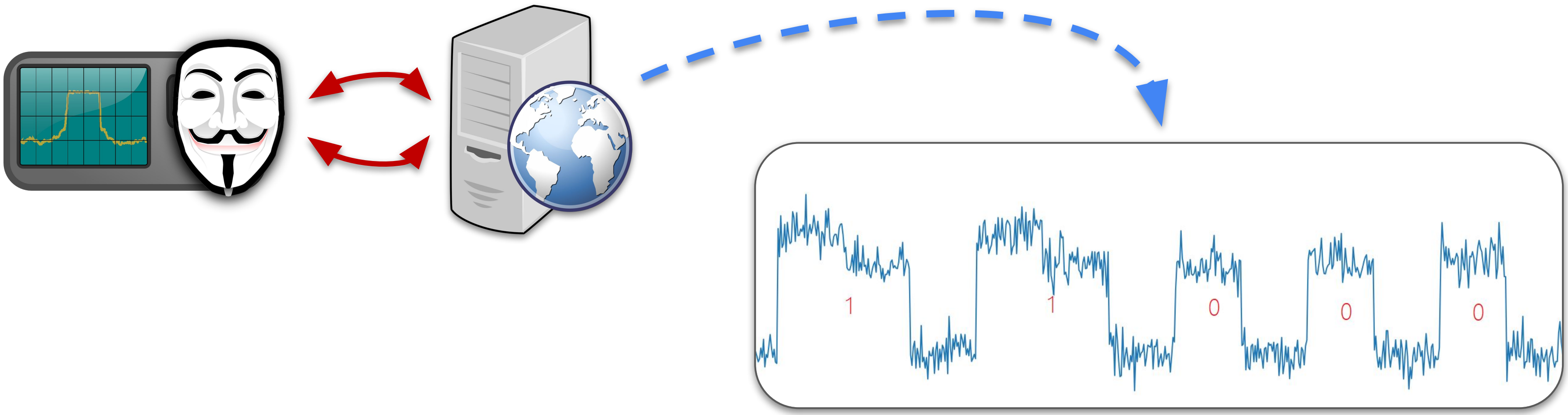
`privKey[i] == 1`

1. Find square of **x**
2. Take modulo **N**
3. Multiply by **C**
4. Take modulo **N**

Timing and **power** will **differ** between key bits **0** versus **1**!

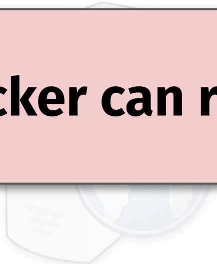
RSA Power Analysis

How can this **side channel** be **exploited**?



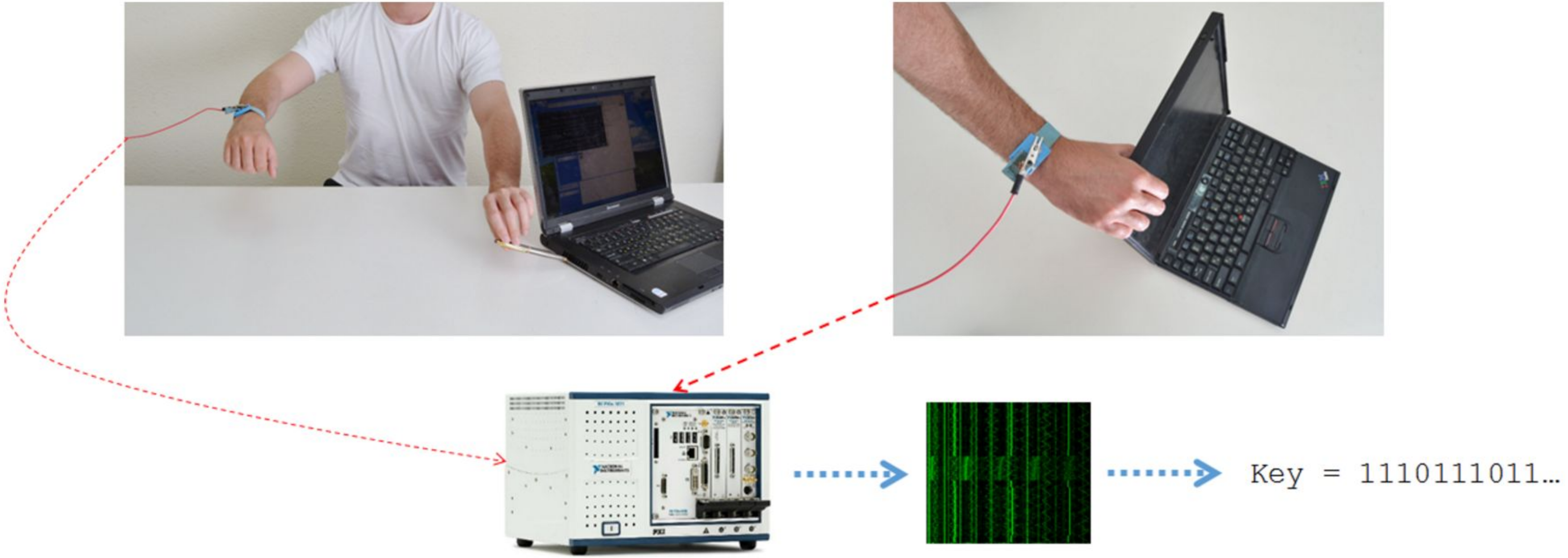
RSA Power Analysis

How can this **side channel** be **exploited**?



Attacker can retrieve a user's private key!

Realistic Power Analysis

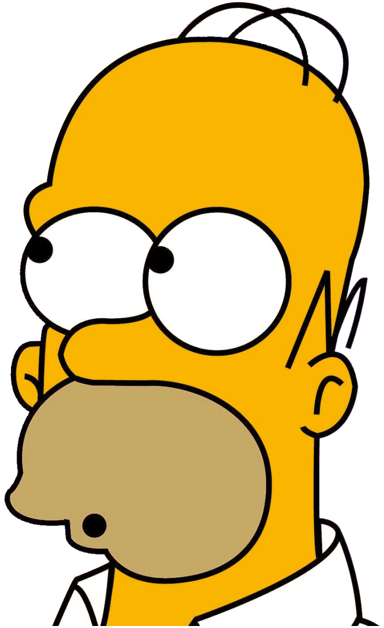


Mitigations

- **Solution: ???**

Mitigations

- **Solution:**
  - Make critical code **constant-time**
- **… but what could go wrong?**



Mitigations?

- **Solution:**
  - Make critical code **constant-time**
- **… but what could go wrong?**
  - Code may end up being compiled **without constant-time protection**
  - The code that you see may **NOT** be the code **that you will execute**

Breaking Bad: How Compilers Break Constant-Time Implementations

Moritz Schneider
ETH Zurich
Zurich, Switzerland
moritz.schneider@inf.ethz.ch

Daniele Lain
ETH Zurich
Zurich, Switzerland
daniele.lain@inf.ethz.ch

Ivan Puddu
ETH Zurich
Zurich, Switzerland
ivan.puddu@inf.ethz.ch

Nicolas Dutly
ETH Zurich
Zurich, Switzerland
nicolas.dutly@inf.ethz.ch

Srdjan Capkun
ETH Zurich
Zurich, Switzerland
srdjan.capkun@inf.ethz.ch

Abstract

The implementations of most hardened cryptographic libraries use defensive programming techniques for side-channel resistance. These techniques are usually specified as guidelines to developers on specific code patterns to use or avoid. Examples include performing arithmetic operations to choose between two variables instead of executing a secret-dependent branch. However, such techniques are only meaningful if they persist across compilation. In this paper, we investigate how optimizations used by modern compilers break the protections introduced by defensive programming techniques. Specifically, how compilers break high-level constant-time implementations used to mitigate timing side-channel attacks. We run a large-scale experiment to see if such compiler-induced issues manifest in state-of-the-art cryptographic libraries. We develop a tool that can profile virtually any architecture, and we use it to run trace-based dynamic analysis on 44,604 different targets. Particularly, we focus on the most widely deployed cryptographic libraries, which aim to provide side-channel resistance. We are able to evaluate whether their claims hold across various CPU architectures, including x86-64, x86-i386, armv7, aarch64, RISC-V, and MIPS-32.

Our large-scale study reveals that several compiler-induced secret-dependent operations occur within some of the most highly re-

Keywords

Constant time code, cryptographic implementations, compilers

ACM Reference Format:

Moritz Schneider, Daniele Lain, Ivan Puddu, Nicolas Dutly, and Srdjan Capkun. 2025. Breaking Bad: How Compilers Break Constant-Time Implementations. In *ACM Asia Conference on Computer and Communications Security (ASIA CCS '25), August 25–29, 2025, Hanoi, Vietnam.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3708821.3733909

1 Introduction

Since the discovery of timing attacks [24], side-channel vulnerabilities have been one of the major concerns for developers of security-critical code and libraries [21]. Particular attention and expert knowledge are devoted to avoiding side-channel issues in security-critical libraries. Three main hardening techniques approaches are generally followed: i) manual assembly hardening, ii) using special compilers that provide constant time guarantees, and iii) hardening the source code. However, all of these approaches suffer from practical limitations.

The first option is vetting hand-written assembly either by a developer [43] or with automated tools [10]. However, this limits code

Questions?

Cache-based Timing Side Channels: Spectre & Meltdown

CPU Caches

- **RAM** is expensive to load from
  - **Disk** is even more expensive!
- Fastest retrieval: **???**

Storage	Read Time	Capacity	Managed By
Hard Disk	10ms	1 TB	Software/OS
Flash Drive	10–100us	100 GB	Software/OS
RAM	200 cycles	10 GB	Software/OS

https://computationstructures.org/lectures/caches/caches.html

CPU Caches

- **RAM** is expensive to load from
  - **Disk** is even more expensive!
- Fastest retrieval: the **CPU cache**
  - Small storage built-in to CPU
  - Common hierarchy: L1, L2, L3, L4
- Key purpose: accelerate retrieval of **commonly-accessed data**

Storage	Read Time	Capacity	Managed By
Hard Disk	10ms	1 TB	Software/OS
Flash Drive	10–100us	100 GB	Software/OS
RAM	200 cycles	10 GB	Software/OS
L3 Cache	40 cycles	10 MB	Hardware
L2 Cache	10 cycles	256 KB	Hardware
L1 Cache	2–4 cycles	32 KB	Hardware

https://computationstructures.org/lectures/caches/caches.html

Program Execution

- What do you expect to happen here?
  - `index < arraySize`
    - ???

```
int read(int index){
    int result = -1;
    result = array[index];
    return result;
}
```

Program Execution

- What do you expect to happen here?
  - `index < len(array)`
    - Within-bounds read… **success**
  - `index > len(array)`
    - **???**

```
int read(int index){
    int result = -1;
    result = array[index];
    return result;
}
```

Program Execution

- What do you expect to happen here?
  - `index < len(array)`
    - Within-bounds read... **success**
  - `index > len(array)`
    - Out-of-bounds read... **prevent**

- Optimization: **Speculative Execution**
  - Perform the **OOB read** anyways

```
int read(int index){
    int result = -1;
    result = array[index];
    return result;
}
```

Program Execution

- What do you expect to happen here?
  - index < len(array)
    - Within-bounds read... **success**
  - index > len(array)
    - Out-of-bounds read... **prevent**
- Optimization: **Speculative Execution**
  - Perform the **OOB read** anyways
  - **Cache** whatever data is accessed
  - Check if it's allowed... **after the fact**
  - **Roll-back** the cache to correct state

```
int read(int index){
    int result = -1;
    result = array[index];
    return result;
}
```

Save time by having data **pre-cached** and ready to go!

Program Execution

- What do you expect to happen here?

Implication: data we **shouldn't** have access to (e.g., from another program) is **cached**

Cache lookup is faster... can we exploit a **timing side channel** to recover this data?

- Roll-back the cache to correct state

Attacking Speculative Execution

- Suppose speculative execution caches a secret `result` of **`4440`**

```
// index > len(array)
int read(int index){
     int result = -1;
     result = array[index];
     return result;
}
```

Attacking Speculative Execution

- Suppose speculative execution caches a secret `result` of **`4440`**

```
// index > len(array)
int read(int index){
    int result = -1;
    result = array[index];
    return result;
}
```

1. Cache `array[index]`
2. Bounds check `index`
3. **Clear `array[index]`**

Due to roll-back, we can't retrieve result!

Attacking Speculative Execution

- Suppose speculative execution caches a secret `result` of `4440`

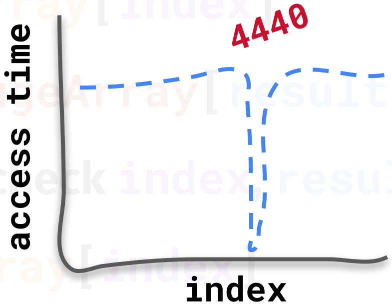
```
// index > len(array)
int read(int index){
    int result = -1;
    result = array[index];
    int dummy = hugeArray[result];
    return result;
}
```

1. **Cache `array[index]`**
2. **Cache `hugeArray[result]`**
3. **Bounds check `index`, `result`**
4. **Clear `array[index]`**
5. **`hugeArray[result]` stays...**

Attacking Speculative Execution

How can attacker figure out `result` is `4440`?

```
for (int i=0; i<...; i++){
    int x = hugeArray[i];
}
```



Since 4440 was cached, `hugeArray[4440]` has the fastest access time of all array indices!

Attacking Speculative Execution

- **Widespread:** affects nearly every device (laptop, phone, etc.)
  - Both ARM and Intel variants
  - **Fully breaks process isolation**

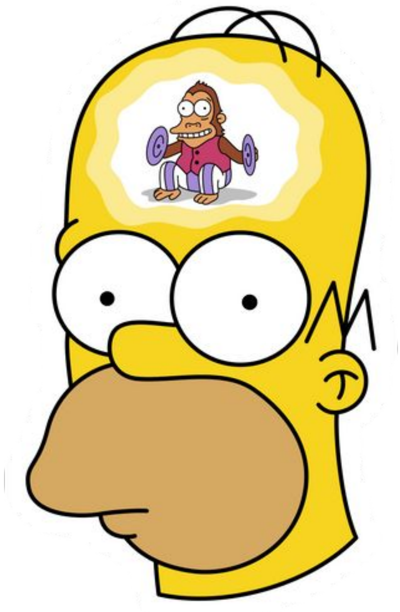
> - On 27 March 2017, researchers at Graz University of Technology developed a proof-of-concept that could grab RSA keys from Intel SGX enclaves running on the same system within five minutes by using certain CPU instructions in lieu of a fine-grained timer to exploit cache DRAM side-channels.[41]

SPECTRE

MELTDOWN

Mitigations?

- **Can CPU-based side channels be practically fixed?**



Mitigations?

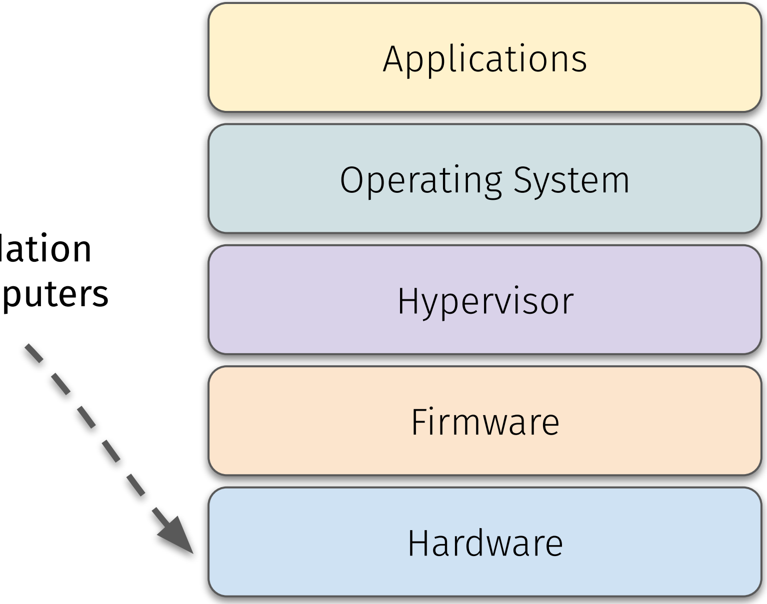
- **Can CPU-based side channels be practically fixed?**
  - Not really… must **disable speculative execution**
  - **Goodbye performance!**

Questions?

Hardware Security

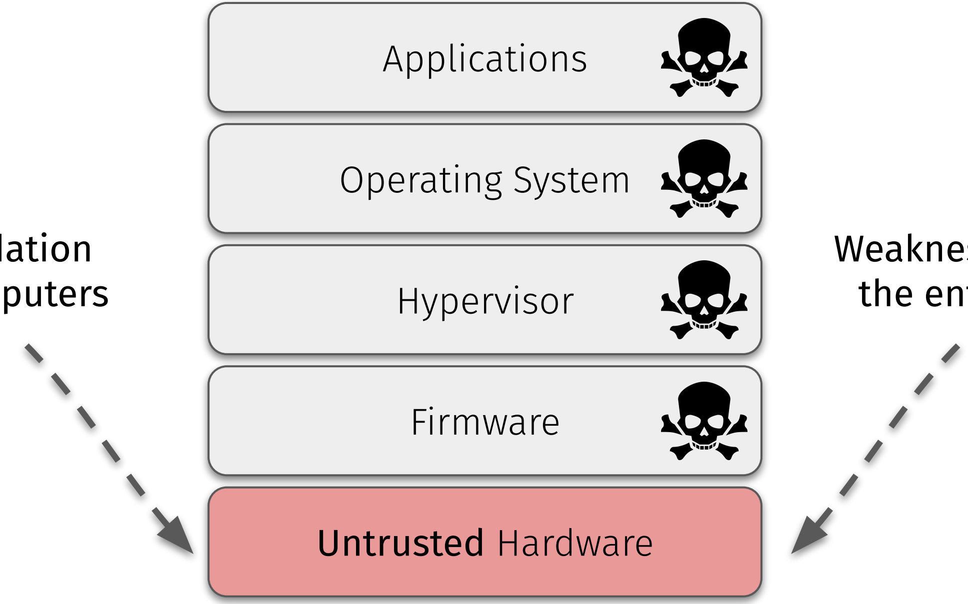
Hardware

The foundation
of our computers



Hardware

The foundation
of our computers



Weaknesses weaken
the entire system

Hardware

FORESHADOW

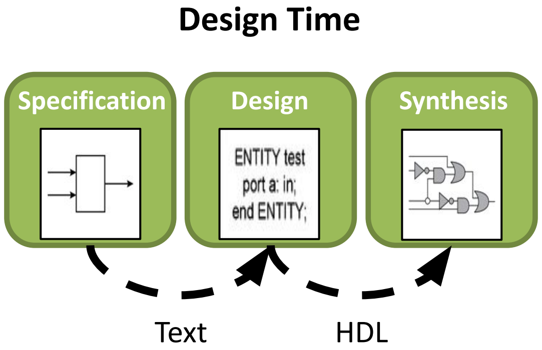
The foundation of our computers

Weaknesses weaken the entire system

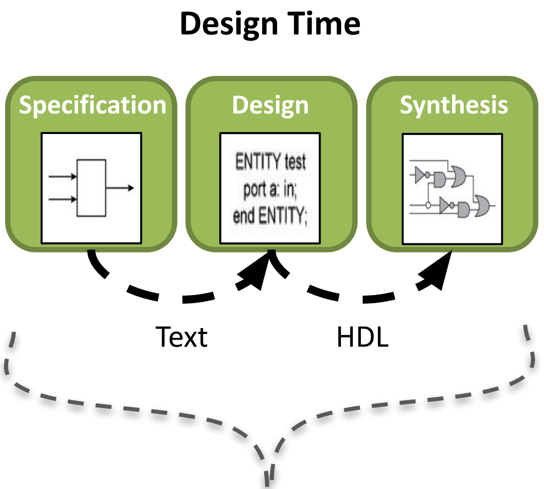
MELTDOWN

SPECTRE

Creating Hardware

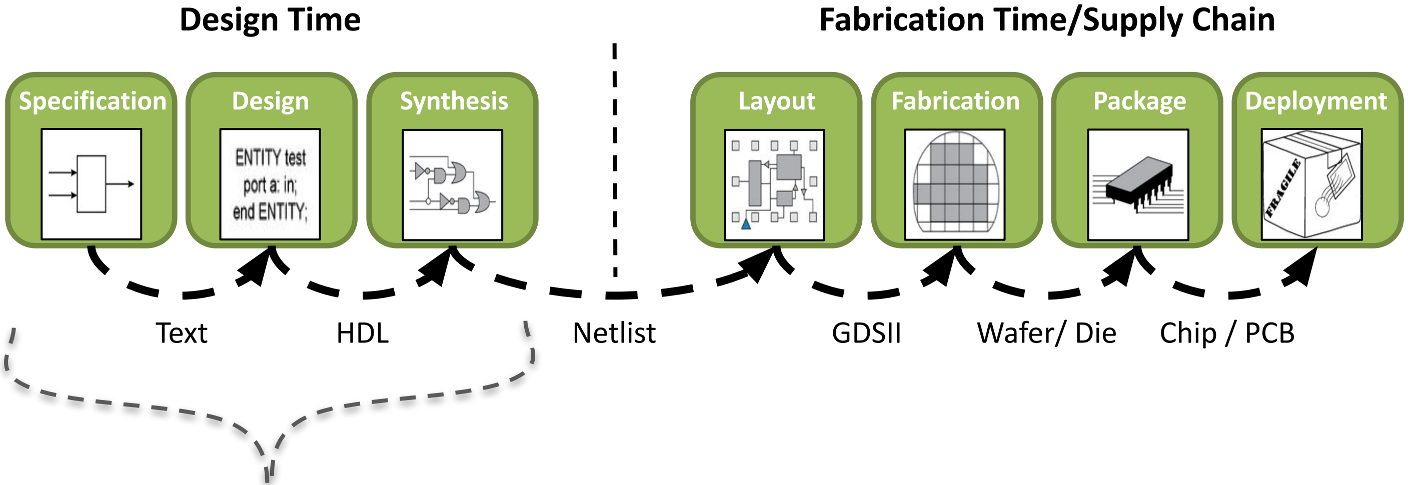


Creating Hardware



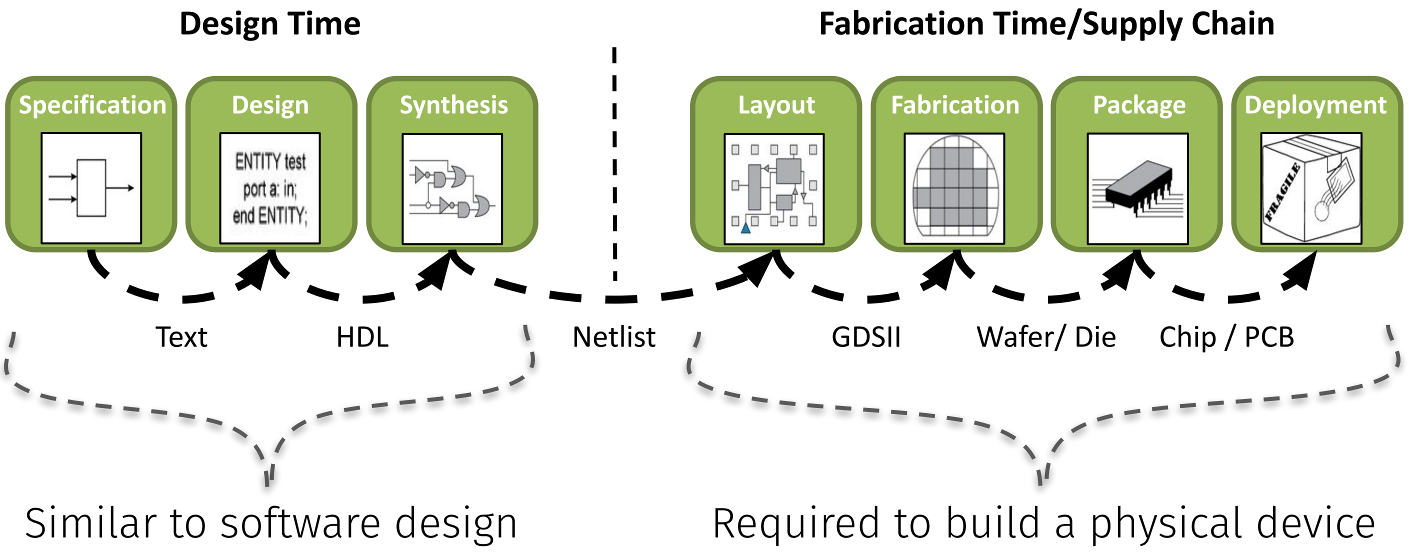
Similar to software design

Creating Hardware

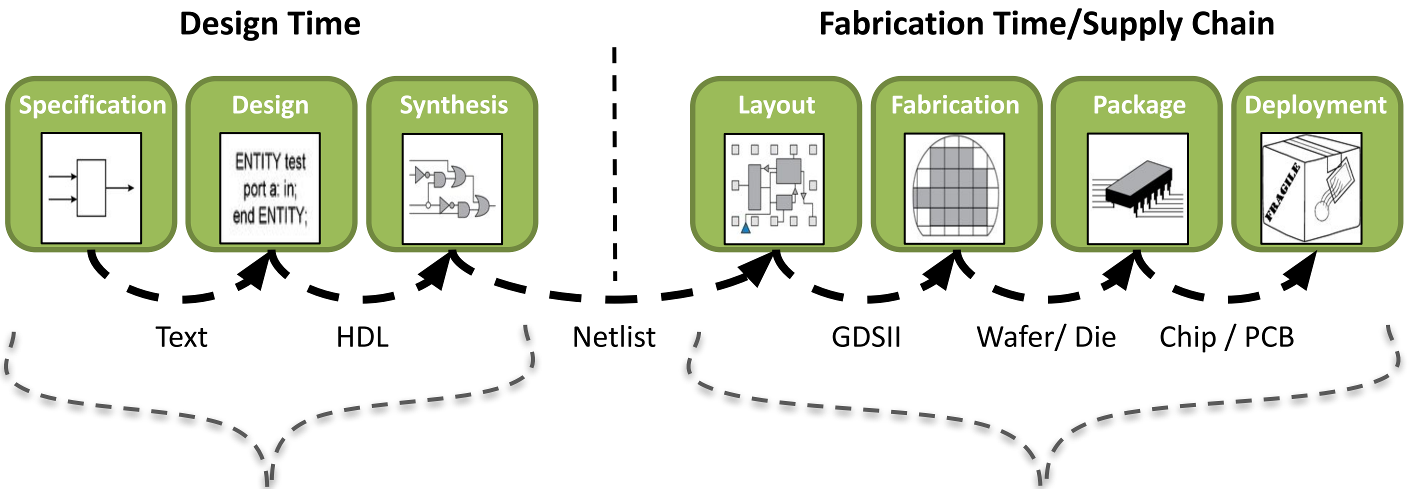


Similar to software design

Creating Hardware



Creating Hardware

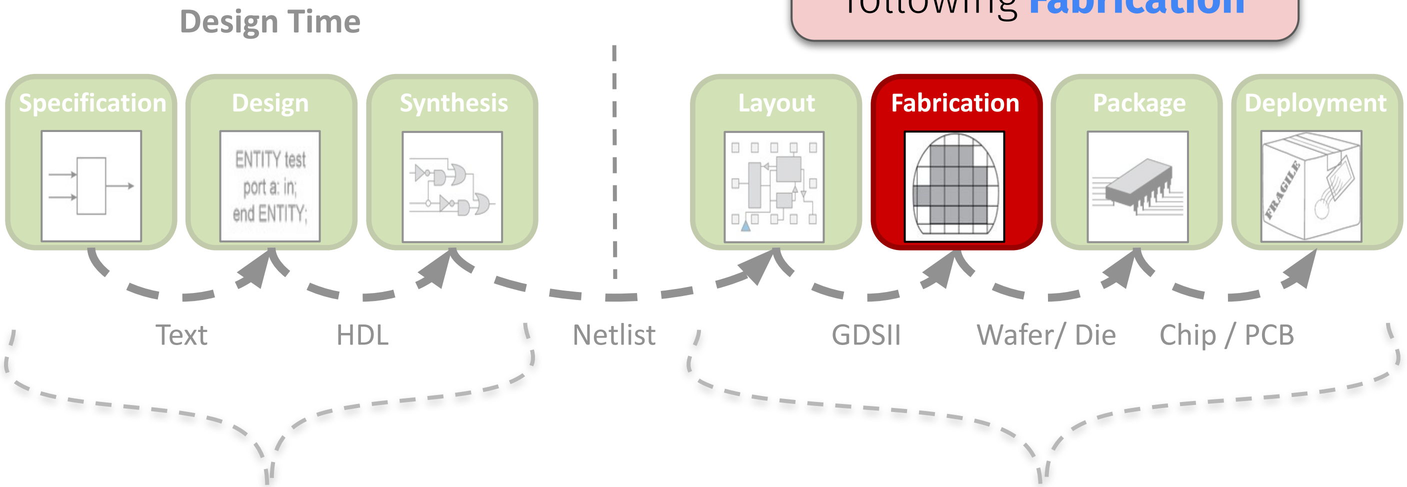


Similar to software design
Verification

Required to build a physical device
Testing

Hardware Bugs

Cannot be **patched** following **Fabrication**



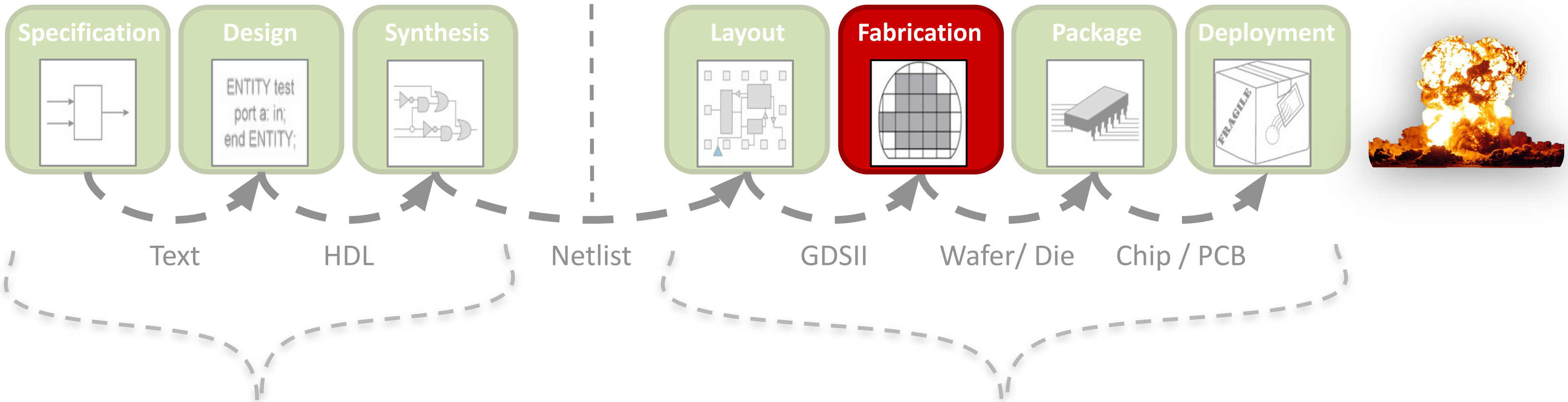
Similar to software design
Verification

Required to build a physical device
Testing

Hardware Bugs

Cannot be **patched**
following **Fabrication**

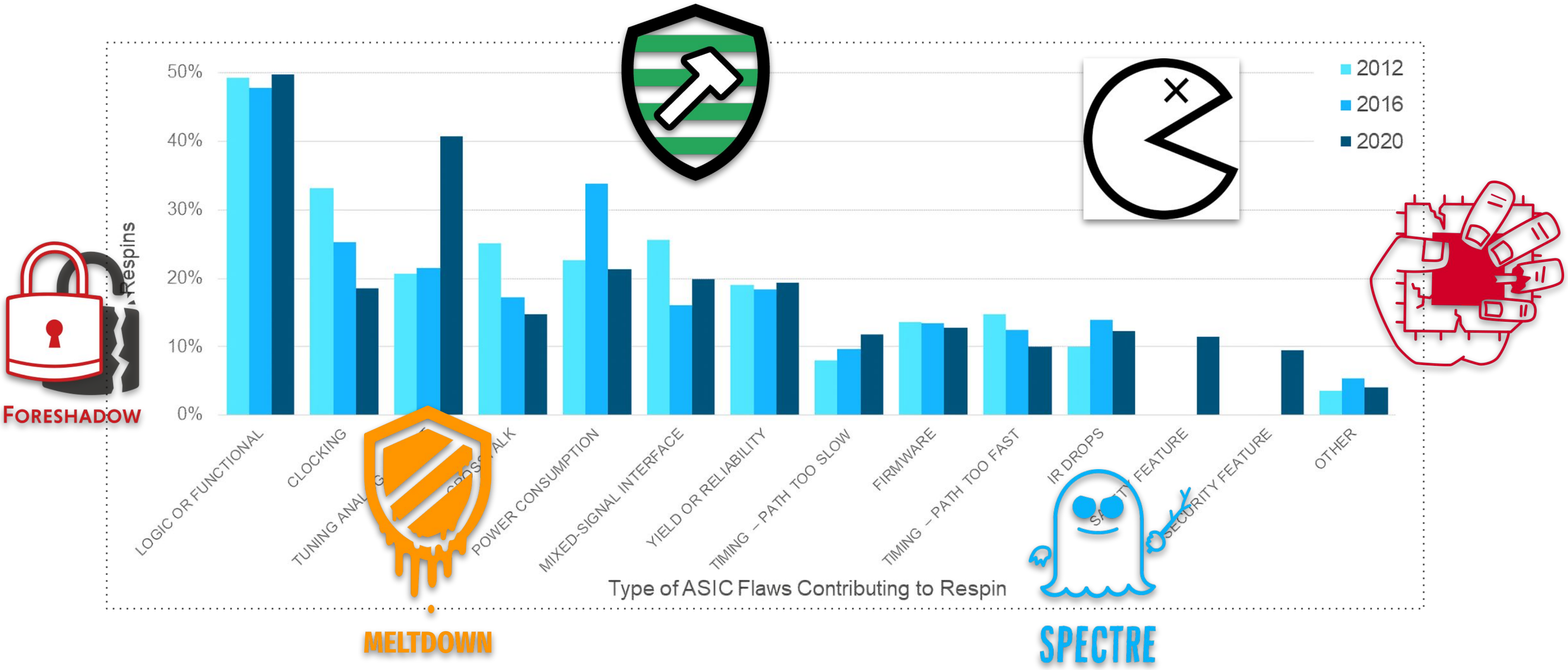
Design Time



Similar to software design
Verification

Required to build a physical device
Testing

Hardware Bugs



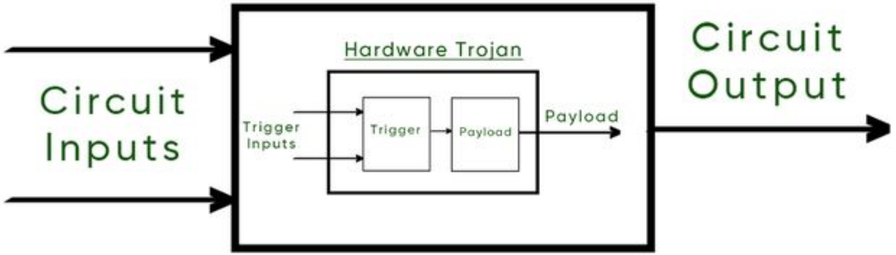
Hardware Threats

Hardware Trojans

- **Trojan Horse:**
  - **???**

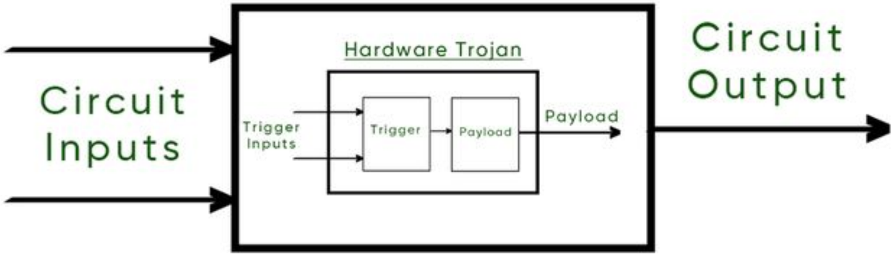
Hardware Trojans

- **Trojan Horse:**
  - Attack pre-inserted into chip
  - Will be **exploited** at **run time**
  - **Remotely triggered** by attacker



Hardware Trojans

- **Trojan Horse:**
  - Attack pre-inserted into chip
  - Will be **exploited** at **run time**
  - **Remotely triggered** by attacker



- **Ideal characteristics:**
  - Small
  - Stealthy
  - Controllable

Hardware Trojans

- **Trojan Horse:**
  - Attack pre-inserted into chip
  - Will be **exploited** at **run time**
  - **Remotely triggered** by attacker
- **Ideal characteristics:**
  - Small
  - Stealthy
  - Controllable
- **Engineering a trigger**

```
void attack_signed_c() {
    volatile int a, b, c = 0;

    while(1) {
        int c1 = c;
        int b1 = b;

        int i1 = ((b1 / c1) + 1);
        int i2 = ((i1 / c1) + 1);
        int i3 = ((i2 / c1) + 1);
        int i4 = ((i3 / c1) + 1);
        int i5 = ((i4 / c1) + 1);
        int i6 = ((i5 / c1) + 1);
        int i7 = ((i6 / c1) + 1);
        int i8 = ((i7 / c1) + 1);
        int i9 = ((i8 / c1) + 1);

        a = ((i9 / c1) + 1);
    }
}
```

Division sets div-by-zero flag

Addition resets div-by-zero flag

Software state will affect **analog state**!

Hardware Trojans

Israeli sky-hack switched off Syrian radars countrywide

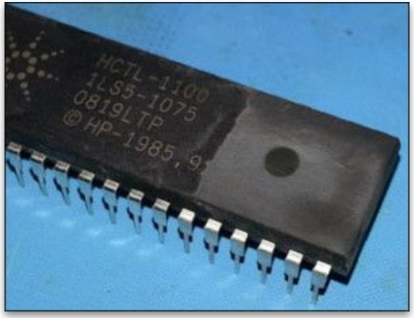
Backdoors penetrated without violence

Lewis Page   Thu 22 Nov 2007 // 13:57 UTC

More rumours are starting to leak out regarding the mysterious Israeli air raid against Syria in September. It is now suggested that "computer to computer" techniques and "air-to-ground network penetration" took place.

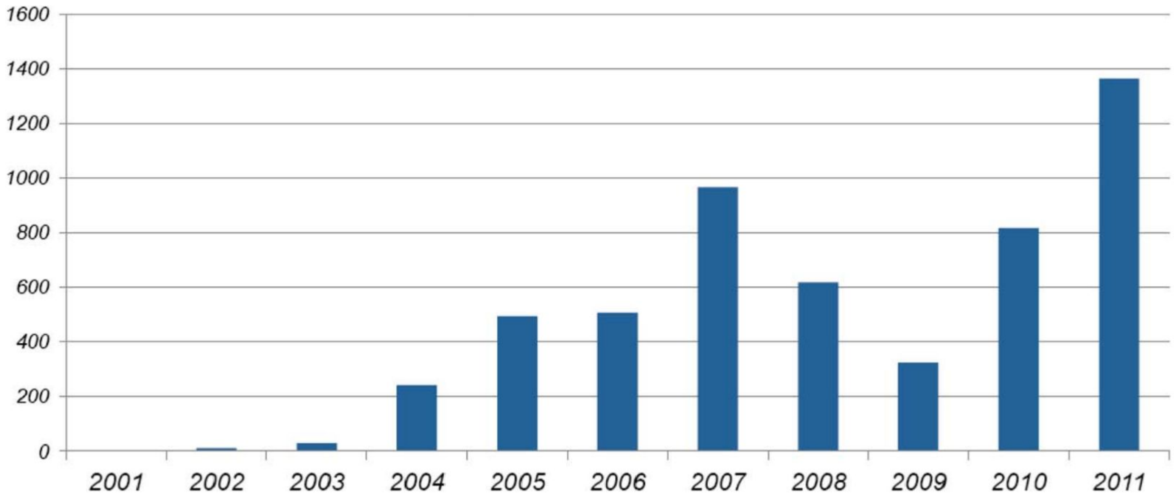
The latest revelations are made by well-connected *Aviation Week* journalists. Electronic-warfare correspondent David Fulghum says that US intelligence and military personnel "provided advice" to the Israelis regarding methods of breaking into the Syrian air-defence network.

Recycled and Counterfeit Hardware



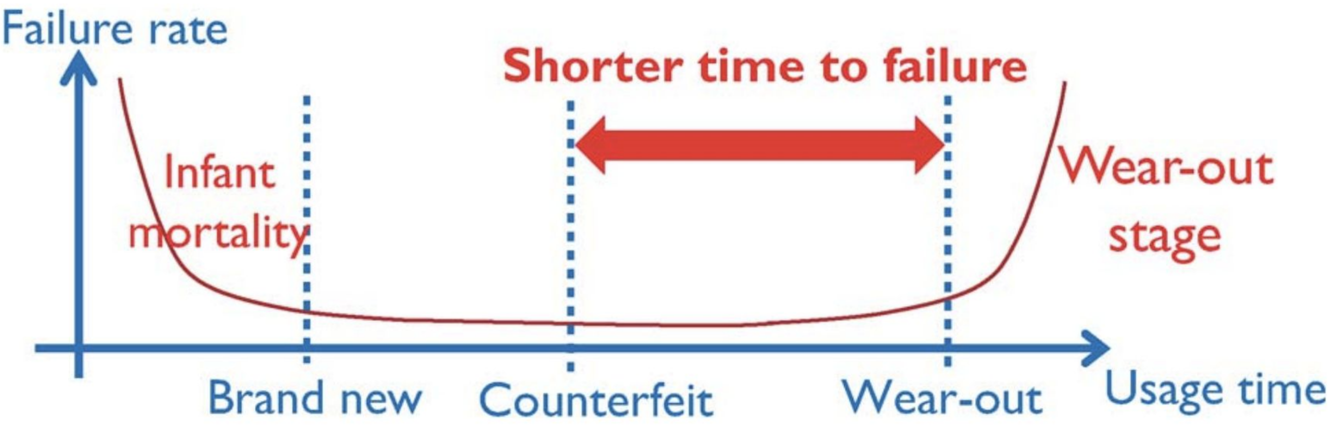
Russia is resorting to putting computer chips from dishwashers and refrigerators in tanks due to US sanctions, official says

Guin *et al.*: Counterfeit Integrated Circuits: A Rising Threat in the Global Semiconductor Supply Chain



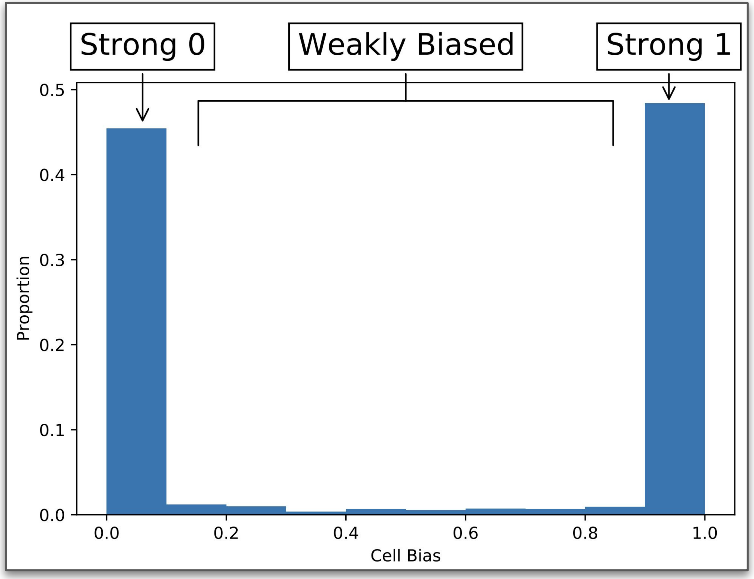
Recycled and Counterfeit Hardware

- **Counterfeit** and **recycled chips** have a **shorter lifespan**
  - Absolutely dangerous for security-critical use cases



Recycled and Counterfeit Hardware

- **Counterfeit** and **recycled chips** have a **shorter lifespan**
  - Absolutely dangerous for security-critical use cases



Secure Hardware

- **Can we ever know for sure that a chip is secure?**

Next time on CS 4440...

Cyber-physical Systems & IoT Security