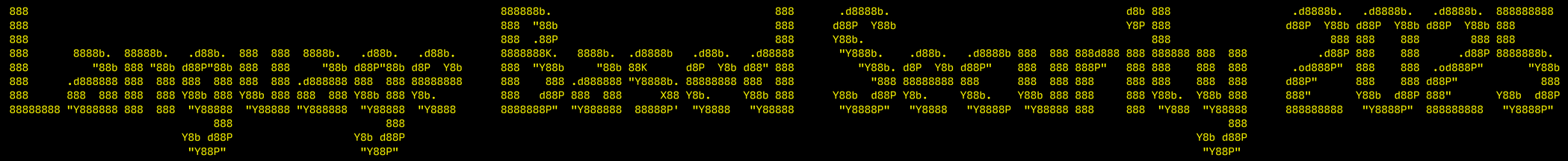
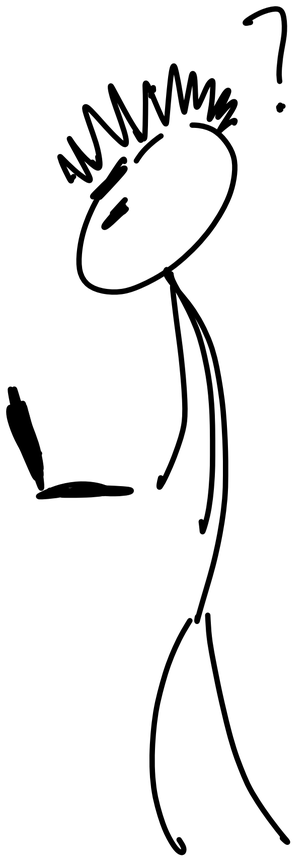


Aslan Askarov

aslan@cs.au.dk

Administrativia

- The course is a mix of lectures by instructor and student presentations
- This is a research-oriented course: be prepared to read and reflect upon research papers.
  - mix of classical papers and cutting-edge (when not everything written in a paper is “carved-in-stone”)
- OBS: I know that for many this is out their comfort zone.



Workload

- You:
  - Programming assignments
    1. warmup (covert channels)
    2. buffer overflows
    3. information flow
  - Theoretical assignments (for most weeks when we have no prog. asgmt)
  - Group presentation of a research article
  - Group project
    - report
    - oral exam
    - (time permitting) presentation to peers at the end of the course

- Me:
  - lectures
  - helping y'all w/ article presentations and advising projects
- TA
  - grading assignments
  - consultations on the material

Tentative timeline

February March April May June

Lectures *Project proposal* *Paper presentations* *Report* *Exam*

Programming assignments *Project work*

Topics:

1. Principles of security
2. Memory safety: buffer overflows, ROP, CFI
3. Capability systems (* guest lecture*)
4. Software Fault Isolation
5. Secure compilation
6. Quantitative security
7. Smart contract security

Presentation of a research article: three-phase process

1. Once we cover a bunch of introductory topics, I put up (1.2–1.5)×NGroups candidate articles for student presentations
2. Each group takes a look at the list and responds by specifying the following
    1. for each article:
        - how interested are they in studying it in detail and presenting it in class on scale from **-3** to **3**
        - how familiar are they with the topic on scale from **A** to **C**
    2. their availability (i.e., when they certainly CANNOT present)
3. All that into information is fed to a constraint solver that gives us the final schedule

- The goal is to avoid having to present something you don’t like
- I meet with every group 1-3 days before the presentation

Project

- Done in groups of up to 3.
- Projects are typically unique across the years
- Project specified in the consultation with the instructor
- We hope to have short project presentations at the end
- Exam: individual, 30 min, oral, during exam period
  - Assessment criteria: passing all assignments/presentation/ and the project
- Start thinking about your project early!
- Factors affecting the grade of the project
  - Relative novelty of the project (must be new to you)
  - Technical difficulty of execution
  - Quality of the report and the presentation
- High grade projects often involve a programming artifact

Other admin stuff

- Deadlines, etc, will be posted on Brightspace
- Form your groups
  - 2-3
  - 1 is okay, but not great
- Do read the papers
  - ideally, before the class; after class is ok
- Questions? (now or in the breaks)

Principles of security

Aslan Askarov
aslan@cs.au.dk

“I’m sure there are better ways to disguise sensitive information, but we don’t have a big budget”

computer security is hard

why is computer security hard ?

computer system is as secure as its weakest link

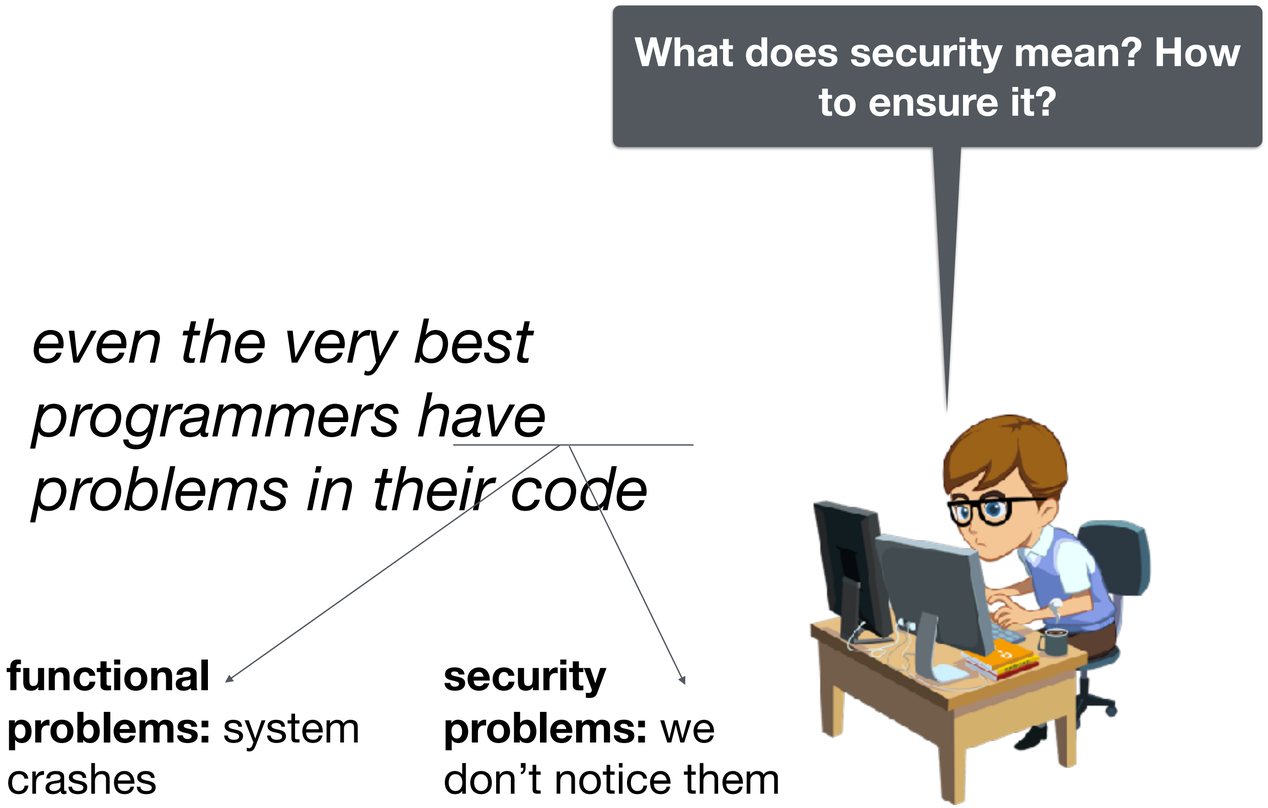
computer system is as secure as its weakest link

computer system is as secure as its weakest link

computer system is as secure as its weakest link

computer system is as secure as its weakest link

computer system is as secure as its weakest link



A successful security attack is the one you don’t know about

Attackers go to great lengths to be unnoticed

- Common for malware and backdoors:
  - Malware attacking Danish bank accounts will idle in the US
  - Stuxnet worm scanned for computers running Siemens software
  - Obfuscated backdoors in random generators for sketchy cryptocurrencies

- NSA caught off-guard when their hacking tools were by leaked “Shadow brokers”
  - “These leaks have been incredibly damaging to our intelligence and cyber capabilities. The fundamental purpose of intelligence is to be able to effectively penetrate our adversaries in order to gather vital intelligence. By its very nature, that only works if secrecy is maintained and our codes are protected.”

    — Leon E. Panetta, the former defense secretary and director of CIA

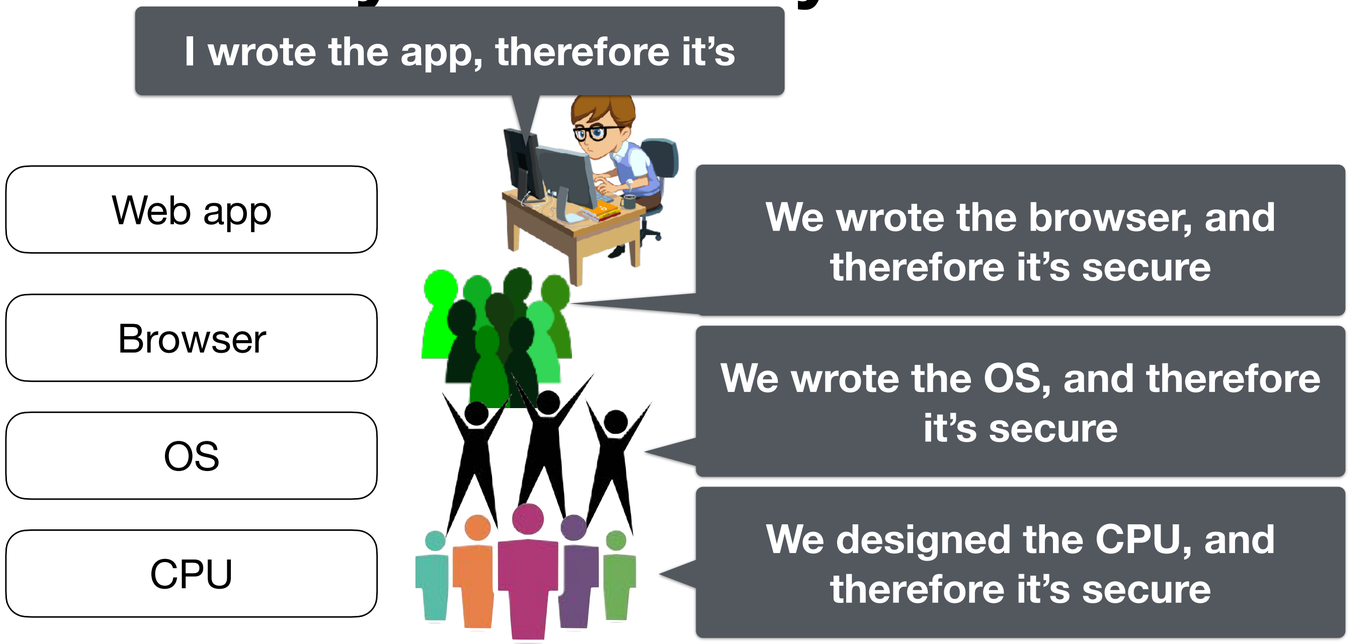
- Historic examples: Coventry Blitz controversy during WW2

Why is security hard?

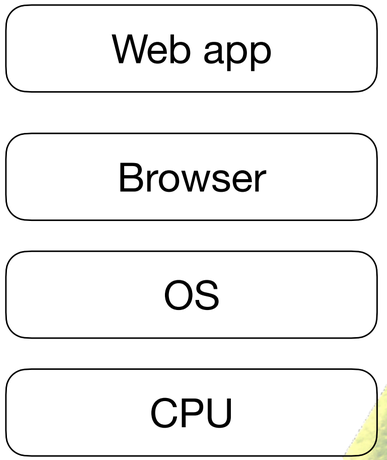
Web app

I wrote the app, therefore it's secure…

Why is security hard?



How likely the whole stack is secure?



Informal math

Probability that the Web app has a security bug = p_1

Probability that the Web app is secure = $1 - p_1$

...

Probability that the CPU has a security bug = p_4

Probability that the CPU is secure = $1 - p_4$

Probability that the "whole stack" is secure

$(1 - p_1) \cdot (1 - p_2) \cdot (1 - p_3) \cdot (1 - p_4)$

Why is security hard?

Web app

Browser

OS

CPU

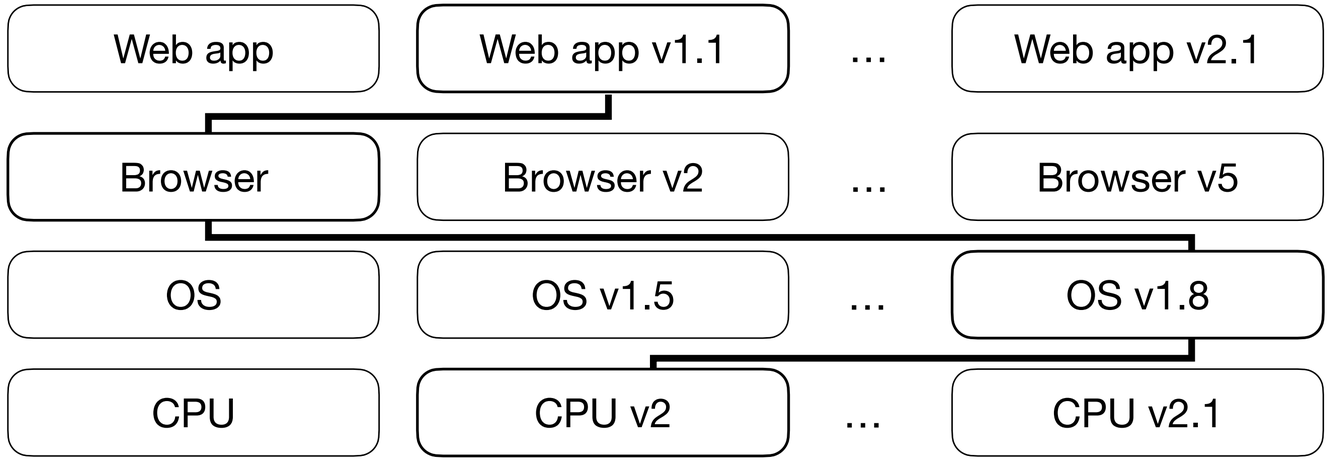
I wrote the app, and carefully studied the stack, and I trust all these other smart people, therefore, running my app on this browser on this OS on this CPU

Why is security hard?

Web app	Web app v1.1	…	Web app v2.1
Browser	Browser v2	…	Browser v5
OS	OS v1.5	…	OS v1.8
CPU	CPU v2	…	CPU v2.1

Analyzing, or even anticipating, all possible combinations is much harder

Why is security hard?



Analyzing, or even anticipating, all possible combinations is much harder
Eventually, there is a crack somewhere…

“Mrs. Thatcher will now realise that Britain cannot occupy our country and torture our prisoners and shoot our people in their own streets and get away with it.

Today we were unlucky, but remember we only have to be lucky once. You will have to be lucky always.

Give Ireland peace and there will be no more war”.

IRA statement claiming responsibility for Brighton hotel bombing in 1984

Today we were unlucky, but remember we only have to be lucky once. **You will have to be lucky always.**

With everything at stake today, leaving computer security to luck is irresponsible

Is there good news? Yes… Sort of

- Computer security is a mature area of computer science
  - Many classical papers are from 1970s and 1980s; solid books written on security engineering
  - First IEEE Symposium on Security & Privacy took place in 1980.
  - Even though examples in the literature may look outdated, many principles remain valid

- Computer security offers a fascinating combination of pragmatism and elegant theory
  - While nothing can be made 100% secure, can we define how secure we can be? Can we find out when something goes wrong? Can we build-in accountability to know whom to blame when something is wrong?…

Principled approach to security

- A newcomer's approach to computer security
  - Point out a problem; suggest an ad-hoc repair; move on
  - What's wrong w/ this approach?
    - Solves a particular problem – OK
    - But often difficult to generalize – BAD
      - "what do we learn from this?"
      - if we build another system where will the details differ?
        - hard to say when we have so many ad-hoc problems/repairs

- Principled approach: state one's problem in a conceptual framework – ***a security model***
  - This allows others to draw similarities and learn from it
  - May point out to a solution already discovered

Aspects of Security Research [Lampson'04]

Do the mechanisms work?

Policy

What does it mean to be secure?

- Secrecy
- Integrity
- Availability
- Accountability
- Privacy
- Anonymity

Assurance

Mechanism

How is security implemented?

- Access control
- Cryptography
- Firewalls
- Isolation
- Program analysis
- Anti-virus

Security Model

- Security requirements – what do we want from the system?
  - System model – how does the system behave?
  - Threat model – what exactly can attacker do?
- Security mechanisms – how do we implement security?
- Assurance – does it really work?
  - Trust – whom do we trust?

Security mechanisms $\xrightarrow{\text{Assurance}}$ Security requirements

Adversarial thinking: exploiting gaps between abstraction layers

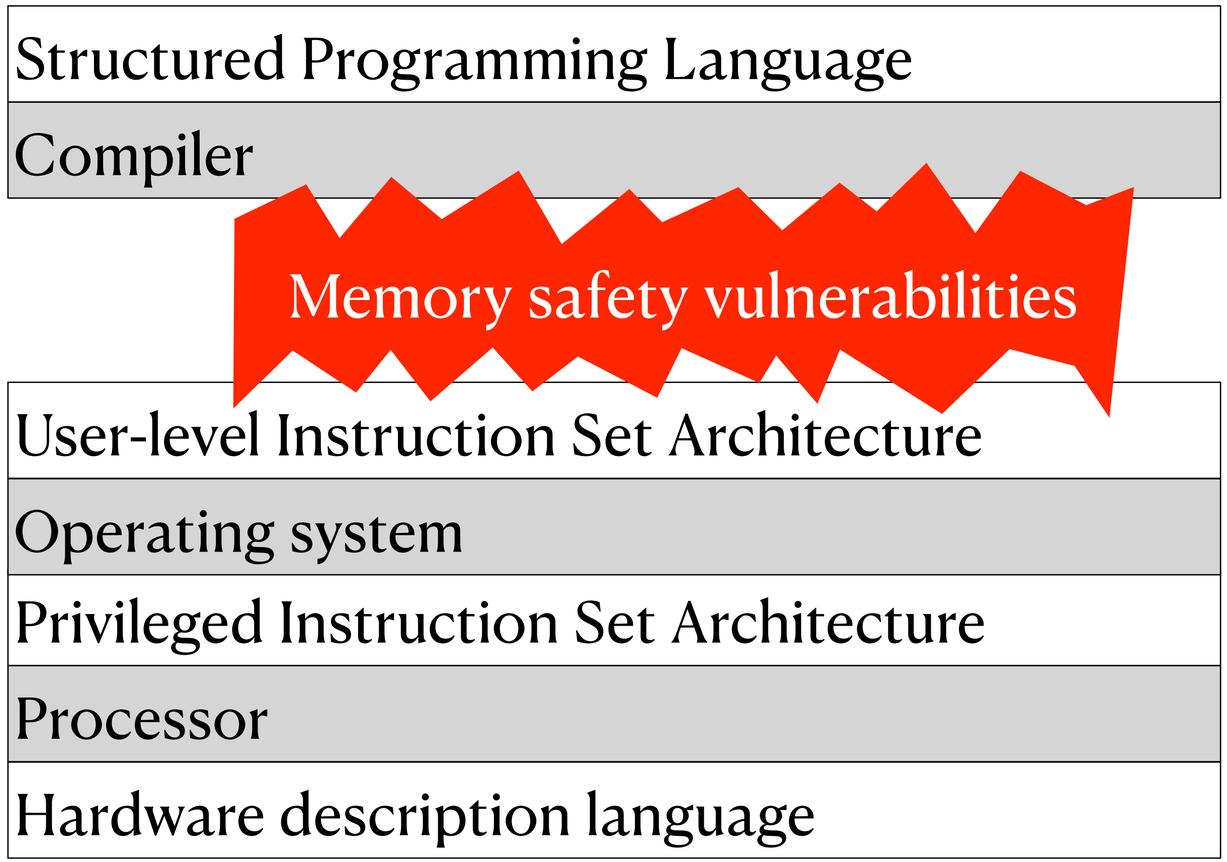
Structured Programming Language
Compiler
User-level Instruction Set Architecture
Operating system
Privileged Instruction Set Architecture
Processor
Hardware description language

Security is about what is *not* supposed to happen

Difficult to test, even more so across abstraction layers

[Security across abstraction layers: old and new examples, Frank Piessens, 2020]

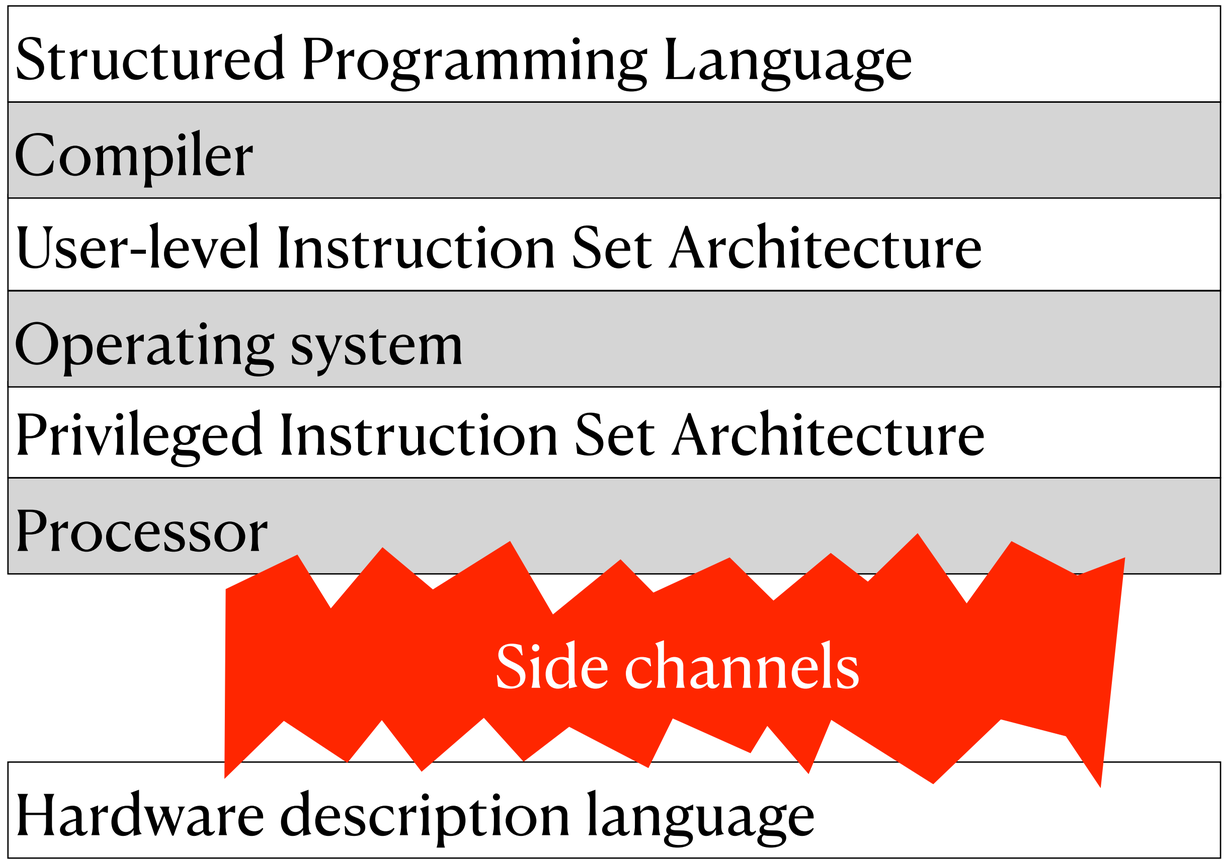
Adversarial thinking: exploiting gaps between abstraction layers



Attacks that take over software programmed in languages like C

In February 2024, White House issued a report calling for broad adoption of memory safe languages

Adversarial thinking: exploiting gaps between abstraction layers



Attacks that leak secrets based on the characteristics of the device

Adversarial thinking: exploiting gaps between abstraction layers

Structured Programming Language
Compiler
User-level Instruction Set Architecture
Operating system
Privileged Instruction Set Architecture
Processor
Hardware description language

Network transmission

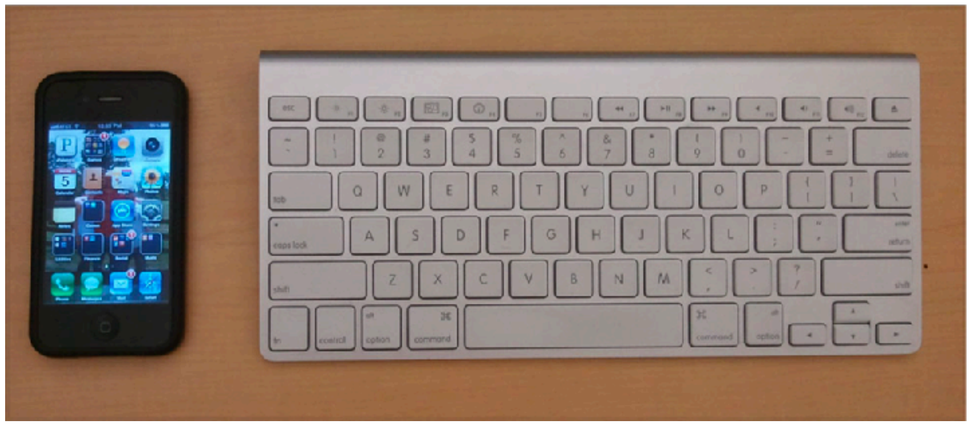
Adversarial thinking: exploiting gaps between abstraction layers

Structured Programming Language
Compiler
User-level Instruction Set Architecture
Operating system
Privileged Instruction Set Architecture
Processor
Hardware description language

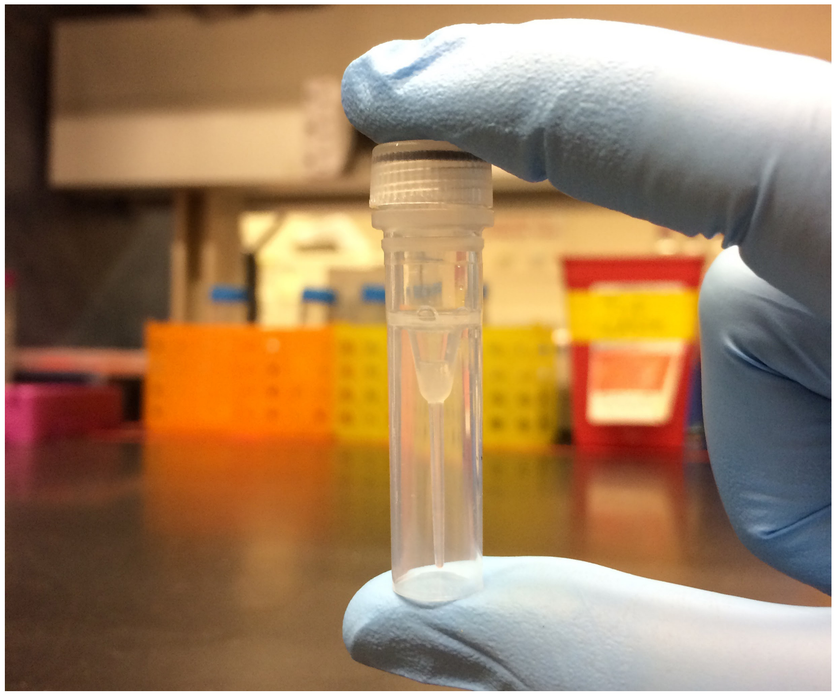
Learning secrets from transmission metadata: *endpoint, time, duration, length*

Exploiting gaps between abstraction layers

Also, in the physical world



Using smartphone accelerometer to infer keystrokes from desk vibrations [Marquardt et al., 2011]



Synthesized biomaterial that triggers buffer overflows in a DNA sequencing machine [Ney et al, 2017]

Trust

- Trusted Computing Base (TCB): the set of all system components that must be trusted in order to maintain system security
  - Security meta-goal: minimize TCB
- “Reflections on Trusting Trust” [K. Thompson, 1984]

- Particularly important when society is based on trust
  - It is dangerous to transfer trust between people to trust between between people and technology.
    - Just because I trust you does not mean I trust your laptop or your server — they can betray my (and yours) trust despite your best intentions.

Threat model – what exactly can attacker do?

- What can attacker *observe*?
  - Final output, I/O events, parallel memory access, timing, power consumptions?
  - Special case: forensic attacker (access to full device)
- How can attacker *influence* the system?
  - Attacker can provide part of the input
    - often times, the attacker is just another user/principal
  - Co-resident attackers that not only observe but also influence the runtime
    - CPU caches, GC, browser event queue, network congestion
  - Attacker provides code that we run

System model – how does the system behave?

- If we can’t even describe how the system works, it’s pointless to reason about its security

- Examples of formal models:
  - Trace models:
    - sequence of actions in a network protocol
    - a sequence of sys calls in an OS execution
  - Models based on PL semantics – these allow us to formally characterize behavior of programs
    - we can reuse rigorous techniques from PL theory

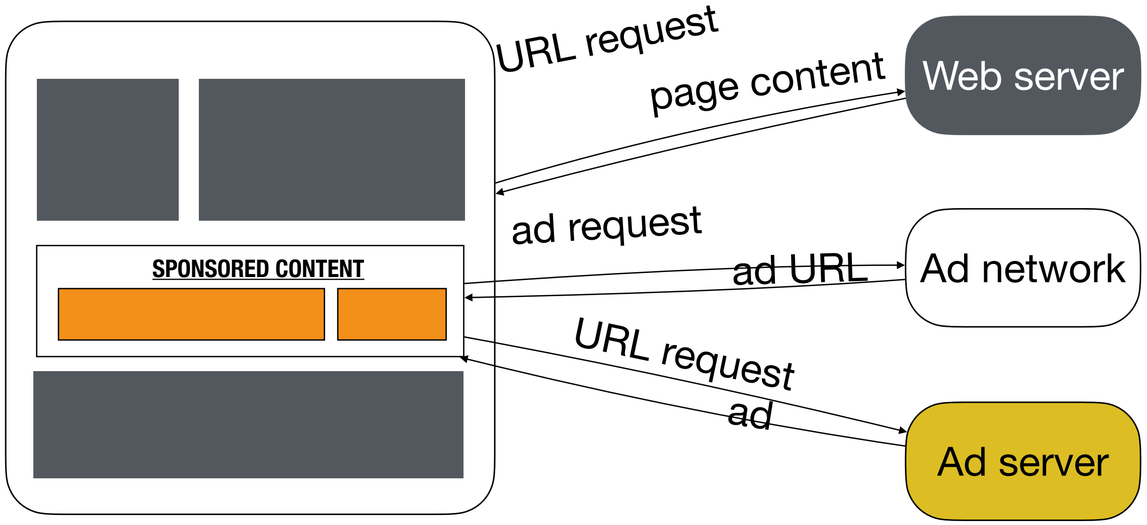
Common security policies

- Confidentiality
  - preventing unauthorized information release
- Integrity
  - preventing unauthorized information modification
- Availability
  - preventing unauthorized denial of use

Security requirements – what do we want from the system

- Common requirements:
  - Confidentiality/Integrity/Availability
- Caveat:
  - We don’t really have a foundational understanding of why these requirements are considered security requirements and the others aren’t.
  - Important to clarify different classes of security requirements and their relationships
- What other requirements can you think of?
  - Privacy?
    - How do you define privacy?

Example: a typical web scenario



We have four principals: *browser, webserver, ad network, ad server*

What are the security requirements here?

https://padlet.com/aslan23/gdqqon2smhfp6p5s

Mechanisms – how do we implement security

- Authentication
- Authorization
- Access control lists or capability (ticked-based) systems
- Isolation/Sandboxing
- Fine-grained monitoring during execution
- Static analysis and verification

Assurance – do our mechanisms really work?

- Can we reduce the security of the system to the security of the trusted computing base?
  - If the attacker can violate our requirements, then our TCB must be broken
- Important to keep TCB small
- Also: defense in depth through redundancy
- Formal reasoning:
  - proving soundness of our mechanisms

A common pitfall

- Reducing all of security to access control/ authentication

- Recall that the objective is to prevent all unauthorized use of information – this is a negative requirement –need to reason about every possible threat
  - narrow concentration on access control/ authentication may lead to false confidence in the system as a whole

The Protection of Information in Computer Systems

JEROME H. SALTZER, SENIOR MEMBER, IEEE, AND
MICHAEL D. SCHROEDER, MEMBER, IEEE

Invited Paper

Abstract - This tutorial paper explores the mechanics of protecting computer-stored information from unauthorized use or modification. It concentrates on those architectural structures--whether hardware or software--that are necessary to support information protection. The paper develops in three main sections. Section I describes desired functions, design principles, and examples of elementary protection and authentication mechanisms. Any reader familiar with computers should find the first section to be reasonably accessible. Section II requires some familiarity with descriptor-based computer architecture. It examines in depth the principles of modern protection architectures and the relation between capability systems and access control list systems, and ends with a brief analysis of protected subsystems and protected objects. The reader who is dismayed by either the prerequisites or the level of detail in the second section may wish to skip to Section III, which reviews the state of the art and current research projects and provides suggestions for further reading.

Glossary

The following glossary provides, for reference, brief definitions for several terms as used in this paper in the context of protecting information in computers.

Access
: *The ability to make use of information stored in a computer system. Used frequently as a verb, to the horror of grammarians.*

Access control list
: *A list of principals that are authorized to have access to some object.*

Authenticate
: *To verify the identity of a person (or other agent external to the protection system) making a request.*

Authorize
: *To grant a principal access to certain information.*

Capability
: *In a computer system, an unforgeable ticket, which when presented can be taken as incontestable proof that the presenter is authorized to have access to the object named in the ticket.*

Certify
: *To check the accuracy, correctness, and completeness of a security or protection mechanism.*

Complete isolation
: *A protection system that separates principals into compartments between which no flow of information or control is possible.*

Confinement
: *Allowing a borrowed program to have access to data, while ensuring that the program cannot release the information.*

Descriptor
: *A protected value which is (or leads to) the physical address of some protected object.*

Discretionary
: *(In contrast with nondiscretionary.) Controls on access to an object that may be changed by the creator of the object.*

Domain
: *The set of objects that currently may be directly accessed by a principal.*

Encipherment
: *The (usually) reversible scrambling of data according to a secret transformation key, so as to make it safe for transmission or storage in a physically unprotected environment.*

Grant
: *To authorize (q. v.).*

Hierarchical control
: *Referring to ability to change authorization, a scheme in which the record of each authorization is controlled by another authorization, resulting in a hierarchical tree of authorizations.*

List-oriented
: *Used to describe a protection system in which each protected object has a list of authorized principals.*

Password
: *A secret character string used to authenticate the claimed identity of an individual.*

Permission
: *A particular form of allowed access, e.g., permission to READ as contrasted with permission to WRITE.*

Prescript
: *A rule that must be followed before access to an object is permitted, thereby introducing an opportunity for human judgment about the need for access, so that abuse of the access is discouraged.*

Principal
: *The entity in a computer system to which authorizations are granted; thus the unit of accountability in a computer system.*

Privacy
: *The ability of an individual (or organization) to decide whether, when, and to whom personal (or organizational) information is released.*

Propagation

Secure design principles

[Saltzer & Schroeder' 1975]

1. Economy of mechanism – simple design makes it feasible to evaluate all possible paths
2. Fail-safe defaults – conservative defaults make security errors observable
3. Complete mediation – check every access to every object; skeptically examine performance gains that sacrifice security
4. Open design – decouple protection by keys from design; it is not realistic to maintain secrecy for the design of a system that is widely used
5. Separation of privilege – avoid single point of compromise
6. Least privilege – limit the damage from error (cf. "need-to-know")
7. Least common mechanism – every shared mechanism presents a new path to leak information
8. Psychological acceptability – design for human use

Functional levels of information protection

[Saltzer & Schroeder' 1975]

- Unprotected systems
- All-or-nothing systems
- Controlled sharing
- User-programmed sharing controls
- Labeled information (putting strings on information)

Revisiting Security Model

- Security requirements – what do we want from the system?
- System model/design – how does a system behave?
  - Security mechanism – how do we implement security
- Threat model – what are the adversary's capabilities?
  - Trust – whom do we trust?
- Assurance – does it really work?

Important to be pragmatic: if the model is not adequate w.r.t. reality, our security assurance is moot.

Threat and trust models sometimes change w/ time

OBS: when reading papers, keep this model and its adequacy in mind

Operating Systems
C. Weissman Editor

A Note on the Confinement Problem

Butler W. Lampson
Xerox Palo Alto Research Center

This note explores the problem of confining a program during its execution so that it cannot transmit information to any other program except its caller. A set of examples attempts to stake out the boundaries of the problem. Necessary conditions for a solution are stated and informally justified.

Key Words and Phrases: protection, confinement, proprietary program, privacy, security, leakage of data
CR Categories: 2.11, 4.30

Introduction

Designers of protection systems are usually preoccupied with the need to safeguard data from unauthorized access or modification, or programs from unauthorized execution. It is known how to solve these problems well enough so that a program can create a controlled environment within which another, possibly untrustworthy program, can be run safely [1, 2]. Adopting terminology appropriate for our particular case, we will call the first program a *customer* and the second a *service*.

The customer will want to ensure that the service cannot access (i.e. read or modify) any of his data except those items to which he explicitly grants access. If he is cautious, he will only grant access to items which are needed as input or output for the service program. In general it is also necessary to provide for smooth transfers of control, and to handle error conditions. Furthermore, the service must be protected from intrusion by the customer, since the service may be a proprietary program or may have its own private data. These things, while interesting, will not concern us here.

Even when all unauthorized access has been prevented, there remain two ways in which the customer may be injured by the service: (1) it may not perform as advertised; or (2) it may leak, i.e. transmit to its owner the input data which the customer gives it. The former problem does not seem to have any general technical solution short of program certification. It does, however, have the property that the dissatisfied customer is left with evidence, in the form of incorrect outputs from the service, which he can use to support his claim for restitution. If, on the other hand, the service leaks data which the customer regards as confidential, there will generally be no indication that the security of the data has been compromised.

There is, however, some hope for technical safeguards which will prevent such leakages. We will call the problem of constraining a service in this way the *confinement* problem. The purpose of this note is to characterize the problem more precisely and to describe methods for blocking some of the subtle paths by which data can escape from confinement.

The Problem

We want to be able to confine an arbitrary program. This does not mean that any program which works when free will still work under confinement, but that any program, if confined, will be unable to leak data. A misbehaving program may well be trapped as a result of an attempt to escape.

A list of possible leaks may help to create some intuition in preparation for a more abstract description of confinement rules.

0. If the service has memory, it can collect data, wait for its owner to call it, and then return the data to him.
1. The service may write into a permanent file in its owner's directory. The owner can then come around at his leisure and collect the data.
2. The service may create a temporary file (in itself a legitimate action which cannot be forbidden without imposing an unreasonable constraint on the computing which a service can do) and grant its owner access to this file. If he tests for its existence at suitable intervals, he can read out the data before the service completes its work and the file is destroyed.
3. The service may send a message to a process controlled by its owner, using the system's interprocess communication facility.
4. More subtly, the information may be encoded in the bill rendered for the service, since its owner must get a copy. If the form of bills is suitably restricted, the amount of information which can be transmitted in this way can be reduced to a few bits or tens of bits. Reducing it to zero, however, requires more far-reaching measures.

Author's address: Xerox Palo Alto Research Center, 3180 Porter Drive, Palo Alto, CA 94304.

A note on the confinement problem

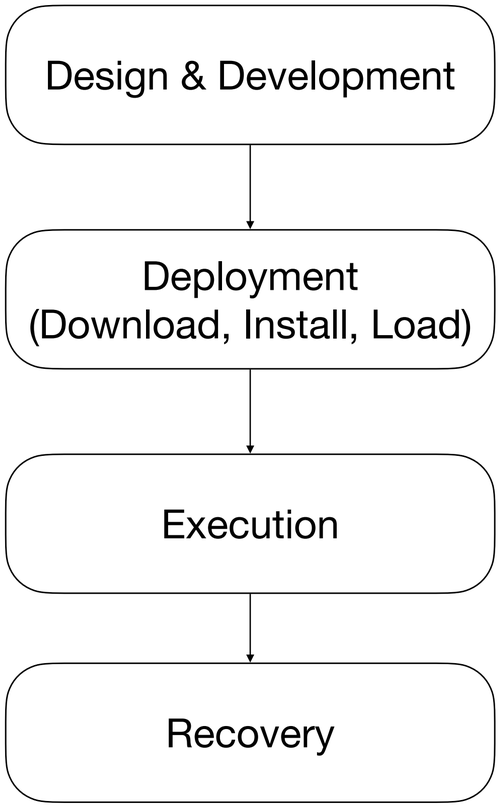
How can a process leak information?

0. return data read from memory directly
1. store data in file
2. create temp files that are destroyed later
3. message passing to another process
4. encode data in a bill
5. emulate shared booleans via interlocks
6. vary the ratio of I/O

Language-Based Security

- Using ideas and techniques from PL to reason and enforce program security
- What are the tools?
  - semantics
  - type systems
  - runtime monitoring
  - automatic code transformations for detection of runtime errors
  - static analysis
  - formal verification
  - compiler techniques

Software lifecycle



- Securing non-malicious code
  - Type-safe languages
  - Formal verification
  - Program synthesis
  - Verified compilation
- Malicious code (viruses, worms, etc
  - Obfuscation

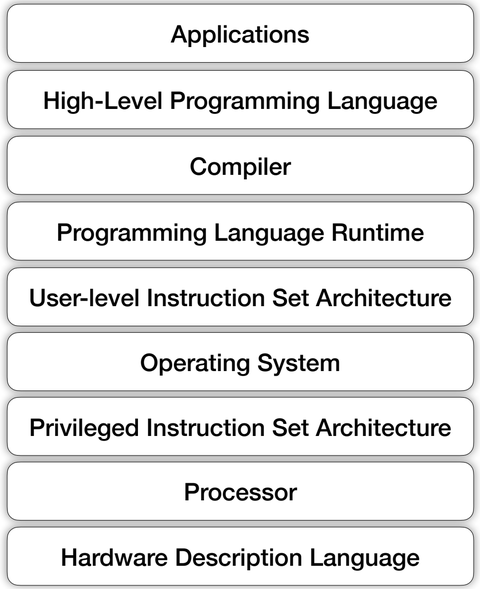
- Code analysis (antivirus scanning)
- Code-signing
- Type-safe target code (e.g., bytecode)
- Independently verified certificates

- Runtime monitoring
- Automatically generated self-monitoring code

- Auditing (logging)
- Rollback (reversible computation, restore points)
- Legal actions

acks: Kevin Hamlen

Security across abstraction layers



A stack of abstraction layers
(adapted from [Piessens'20])

- Creating abstraction layers is the bread and butter of computer scientists
  - Yet abstractions are often leaky
- Many realistic attacks exploit implementation details of lower layers to attack upper layers
  - Memory corruption exploit implementation details of the compiler and the OS
  - Covert channels exploit shared resources (processor caches, schedulers, PL runtime)
  - Hardware-level fault injection exploit physical properties of the chips (e.g., high probability of bit flips under low voltage)

Reading

- The Protection of Information in Computer Systems. [Saltzer & Schroder 1975] Sec 1.
- A Note on the Confinement Problem. B. Lampson. 1973
- Reflections on Trusting Trust. K. Thompson. 1984
- The Research Value of Publishing Attacks by D. Basin and S. Capkun.

- Bonus:
  - Some thoughts on security after ten years of qmail 1.0. [D. Bernstein 2007]

Assignment 1

- Scenario:
  - There is a web service that has a private encryption key stored in its database.
  - Extract as much of the key as you can.
  - If you get the whole key, decrypt the file from Brightspace.
- Work in groups
- Hack, but don’t break, please.
  - If you think the assignment server has blatant holes that I did not think of, kindly let me know

Conclusion

- A successful attack is the one you don’t know about
  - Security attacks often break the abstraction stack
- Security models allow us to use principled approaches to security
  - Policies/Mechanisms/Assurance
    - CIA
- Language-based security: using PL techniques for specifying and reasoning about security