CE 815 - Secure Software Systems

Lecture 0

Mehdi Kharrazi
Department of Computer Engineering
Sharif University of Technology

Acknowledgments: Some of the slides are fully or partially obtained from other sources. Reference is noted on the bottom of each slide, when the content is fully obtained from another source. Otherwise a full list of references is provided on the last slide.

Building secure computer systems

- Secure = achieves some property despite attacks by adversaries.
- Systematic thought is required for successful defense.
  - Details matter!

Devil is in the details!

High-level plan for thinking about security

- Goal: what your system is trying to achieve.
  - e.g. only Alice should read file F.
  - Common goals: confidentiality, integrity, availability.
- Policy: some plan (rules) that will get your system to achieve the goal.
  - e.g. set permissions on F so it's readable only by Alice's processes.
  - e.g. require a password and two-factor authentication.
- Threat model: assumptions about what the attacker can do.
  - e.g. can guess passwords, cannot physically steal our server.
- Mechanism: software/hardware that your system uses to enforce policy.
  - e.g. user accounts, passwords, file permissions, encryption.
  - policy might include human components (e.g., do not share passwords) that's outside of the scope of the security mechanisms

Building secure systems is hard -- why?

- Example: grade files are stored on a university server.
  - Policy: only TAs should be able to read and write the grades file.
- Easy to implement the *positive* aspect of the policy:
  - There just has to be one code path that allows a TA to get at the file.
- But security is a *negative* goal:
  - We want no tricky way for a non-TA to get at the file.
- There are a huge number of potential attacks to consider!
  - Exploit a bug in the server's code.
  - Guess a TA's password.
  - Steal a TA's laptop, maybe it has a local copy of the grades file.
  - Intercept grades when they are sent over the network to the registrar.
  - Get a job in the registrar's office, or as a class TA.

Building secure systems is hard -- why?

- One cannot get policies/threats/mechanisms right on the first try and must usually iterate:
  - Design, watch attacks, update understanding of threats and policies.
  - Post-mortems important to understand
    - Public databases of vulnerabilities (e.g., https://cve.mitre.org/)
    - Encourage people to report vulnerabilities (e.g., bounty programs)
- Defender is often at a disadvantage in this game.
  - Defender usually has limited resources, other priorities.
  - Defender must balance security against convenience.
- A determined attacker can usually win!
  - Defense in depth
  - Recovery plan (e.g., secure backups)
- That is why we review failures to make you start thinking in this way

What's the point if we can't achieve perfect security?

- Perfect security is rarely required.
- Make cost of attack greater than the value of the information.
  - So that perfect defenses aren't needed.
- Make our systems less attractive than other peoples'.
  - Works well if attacker e.g. just wants to generate spam.
- Find techniques that have big security payoff (i.e. not merely patching holes).
  - We'll look at techniques that cut off whole classes of attacks.
  - Successful: popular attacks from 10 years ago are no longer very fruitful.

What's the point if we can't achieve perfect security?

- Sometimes security *increases* value for defender:
  - VPNs might give employees more flexibility to work at home.
  - Sandboxing (JavaScript, Native Client) might give me more confidence to run software I don't fully understand.
- No perfect physical security either.
  - But that's OK: cost, deterrence.
  - One big difference in computer security: attacks are cheap.

What goes wrong #1:

- Problems with the policy.
  - i.e. system correctly enforces policy -- but policy is inadequate.
- Example: Business-class airfare.
  - Airlines allow business-class tickets to be changed at any time, no fees.
  - Is this a good policy?
    - Turns out, in some systems ticket could have been changed even AFTER boarding.
    - Adversary can keep boarding plane, changing ticket to next flight, ad infinitum.
    - Revised policy: ticket cannot be changed once passenger has boarded the flight.
      - Sometimes requires changes to the system architecture.
      - Need computer at the aircraft gate to send updates to the reservation system.

Examples of Policy Failure

- Example: Verifying domain ownership for TLS certificates.
  - Browser verifies server's certificate to ensure talking to the right server.
  - Certificate contains server's host name and cryptographic key, signed by some trusted certificate authority (CA).
  - Browser has CA's public key built in to verify certificates.
  - CA is in charge of ensuring that certificate is issued only to legitimate domain (hostname) owner.
  - Typical approach: send email to the contact address for a domain.

Examples of Policy Failure

- Example: Verifying domain ownership for TLS certificates (con’t)
  - Some TLDs (like .eu) do not reveal the contact address in ASCII text.
    - Most likely to prevent spam to domain owners.
  - Instead, they reveal an ASCII image of the email address.
  - One CA (Comodo) decided to automate this by OCR'ing the ASCII image.
  - Turns out, some ASCII images are ambiguous!
    - E.g., foo@a1telekom.at was mis-OCRed as foo@altelekom.at
    - Adversary can register mis-parsed domain name, get certificate for someone else's domain.
    - [Ref: https://www.mail-archive.com/dev-security-policy@lists.mozilla.org/msg04654.html]

Examples of Policy Failure

- Example: Fairfax County, VA school system.
  - [Ref: http://catless.ncl.ac.uk/Risks/26.02.html#subj7.1]
  - Student can access only his/her own files in the school system.
  - Superintendent has access to everyone's files.
  - Teachers can add new students to their class.
  - Teachers can change password of students in their class.
  - What's the worst that could happen if student gets teacher's password?
    - Student adds the superintendent to the compromised teacher's class.
    - Changes the superintendent's password, since they're a student in class.
    - Logs in as superintendent and gets access to all files.
  - Policy amounts to: teachers can do anything.

Examples of Policy Failure

- Example: Sarah Palin's email account.
  - [Ref: http://en.wikipedia.org/wiki/Sarah_Palin_email_hack]
  - Yahoo email accounts have a username, password, and security questions.
  - User can log in by supplying username and password.
  - If user forgets password, can reset by answering security Qs.
  - Some adversary guessed Sarah Palin's high school, birthday, etc.
  - Policy amounts to: can log in with either password *or* security Qs.
    - No way to enforce "Only if user forgets password, then ..."
  - Thus user should ensure that password *and* security Qs are both hard to guess.

Examples of Policy Failure

- Example: Mat Honan's accounts at Amazon, Apple, Google, etc.
  - [Ref: http://www.wired.com/gadgetlab/2012/08/apple-amazon-mat-honan-hacking/all/]
  - Honan an editor at wired.com; someone wanted to break into his gmail account.
  - Gmail password reset: send a verification link to a backup email address.
    - Google helpfully prints part of the backup email address.
    - Mat Honan's backup address was his Apple @me.com account.
- Apple password reset: need billing address, last 4 digits of credit card.
  - Address is easy, but how to get the 4 digits?
- How to get hold of that e-mail?
  - Call Amazon and ask to add a credit card to an account.
- No authentication required,
  - presumably because this didn't seem like a sensitive operation.

Examples of Policy Failure

- Example: Mat Honan's accounts at Amazon, Apple, Google, etc. (con't)
  - Call Amazon tech support again, and ask to change the email address on an account.
    - Authentication required!
    - Tech support accepts the full number of any credit card registered with the account.
      - Can use the credit card just added to the account.
  - Now go to Amazon's web site and request a password reset.
    - Reset link sent to the new e-mail address.
  - Now log in to Amazon account, view saved credit cards.
    - Amazon doesn't show full number, but DOES show last 4 digits of all cards.
    - Including the account owner's original cards!

Examples of Policy Failure

- Example: Mat Honan's accounts at Amazon, Apple, Google, etc. (con't)
  - Now attacker can reset Apple password, read gmail reset e-mail, reset gmail password.
  - Lesson: attacks often assemble apparently unrelated trivia.
  - Lesson: individual policies OK, but combination is not.
    - Apple views last 4 as a secret, but many other sites do not.
- Lesson: big sites cannot hope to identify which human they are talking to;
  - at best "same person who originally created this account".
  - security questions and e-mailed reset link are examples of this.

Policy Failures

- Policies typically go wrong in "management" or "maintenance" cases.
  - Who can change permissions or passwords?
  - Who can access audit logs?
  - Who can access the backups?
  - Who can upgrade the software or change the configuration?
  - Who can manage the servers?
  - Who revokes privileges of former admins / users / ...?

What goes wrong #2:

- Problems with threat model / assumptions.
  - i.e. designer assumed an attack wasn't feasible (or didn't think of the attack).

- Example: assume the design/implementation is secret
  - "Security through obsecurity"
  - Clipper chip (https://en.wikipedia.org/wiki/Clipper_chip)
  - Broken secret crypto functions
- Example: most users are not thinking about security.
  - User gets e-mail saying "click here to renew your account",
    - then plausible-looking page asks for their password.
  - Or dialog box pops up with "Do you really want to install this program?"
  - Or tech support gets call from convincing-sounding user to reset password.

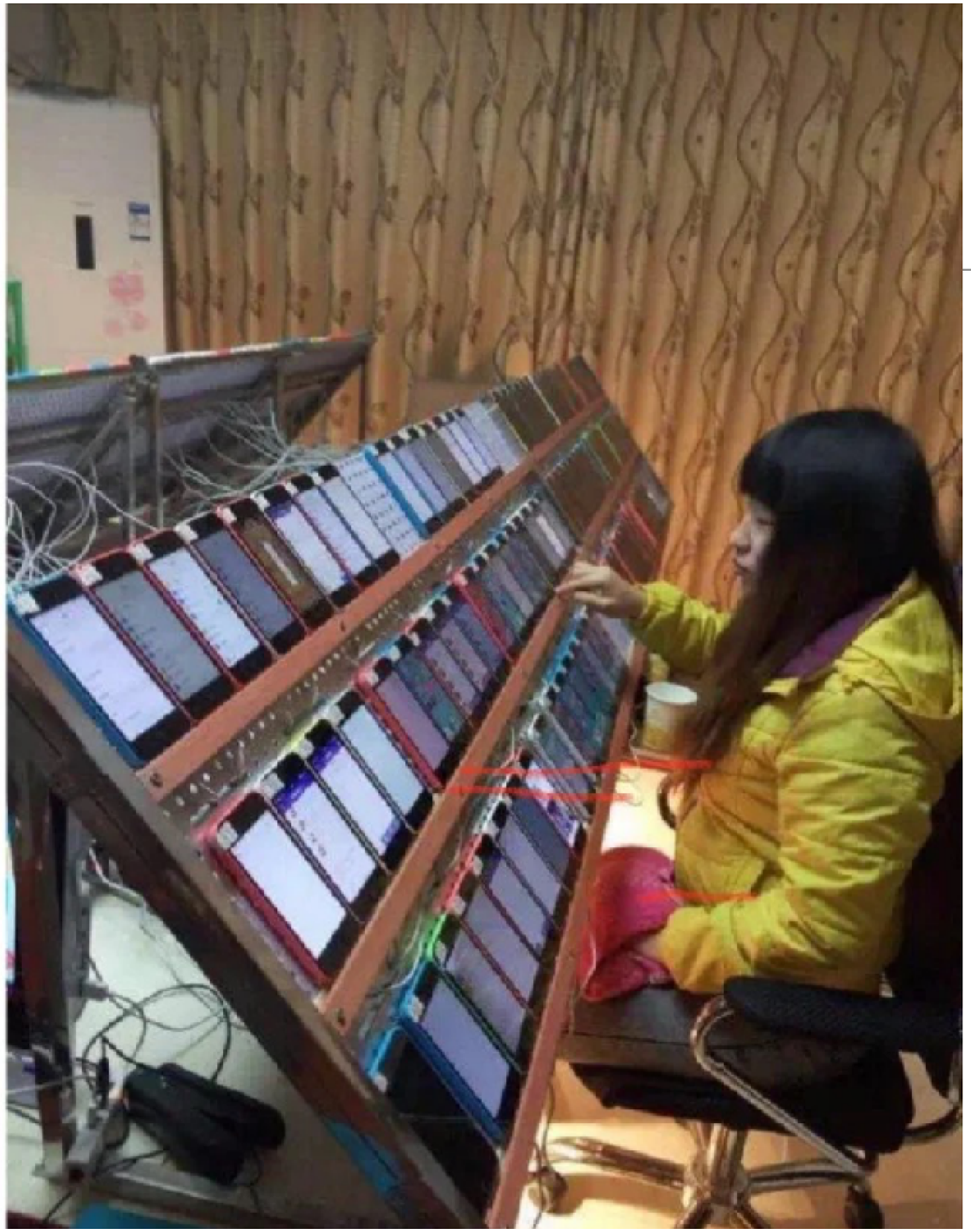
Examples of Incorrect Assumptions

- Example: computational assumptions change over time.
  - MIT's Kerberos system used 56-bit DES keys, since mid-1980's.
  - At the time, seemed fine to assume adversary can't check all 2^56 keys.
  - No longer reasonable: now costs about $100.
    - [Ref: https://www.cloudcracker.com/dictionaries.html]
    - Several years ago, a class final project showed can get any key in a day.

Examples of Incorrect Assumptions

- Example: assuming a particular kind of a solution to the problem.
  - Many services use CAPTCHAs to check if a human is registering for an account.
    - Requires decoding an image of some garbled text, for instance.
  - Goal is to prevent mass registration of accounts to limit spam, prevent high rate of password guessing, etc.
  - Assumed adversary would try to build OCR to solve the puzzles.
    - Good plan because it's easy to change image to break the OCR algorithm.
    - Costly for adversary to develop new OCR!
- Turns out adversaries found another way to solve the same problem.
  - Human CAPTCHA solvers in third-world countries.
  - Human solvers are far better at solving CAPTCHAs than OCRs or even regular users.
- Cost is very low (fraction of a cent per CAPTCHA solved).
  - [Ref: https://www.cs.uic.edu/pub/Kanich/Publications/re.captchas.pdf]

A still from HBO’s Silicon Valley depicting a click farm



A click farm in Asia where low paid workers are used to inflate video views and trick advertisers to believe that their ads are running successfully.

Examples of Incorrect Assumptions

- Example: XcodeGhost.
  - Apple's development tools for iPhone applications (Xcode) are large.
  - Downloading them from China required going to Apple's servers outside of China.
  - Takes a long time.
  - Unofficial mirrors of Xcode tools inside China.
  - Some of these mirrors contained a modified version of Xcode that injected malware into the resulting iOS applications.
  - Found in a number of high-profile, popular iOS apps!
    - [Ref: https://en.wikipedia.org/wiki/XcodeGhost]
  - Classic paper: Reflections on Trusting Trust.

Examples of Incorrect Assumptions

- Example: decomissioned disks.
  - Many laptops, desktops, servers are thrown out without deleting sensitive data.
  - One study reports large amounts of confidential data on disks bought via ebay, etc.
  - [Ref: https://simson.net/page/Real_Data_Corpus]

What goes wrong #3: problems with the mechanism -- bugs

- Bugs routinely undermine security.
  - Rule of thumb: one bug per 1000 lines of code.
  - Bugs in implementation of security policy.
  - But also bugs in code that may seem unrelated to security, but they are not
    - Good mindset: Any bug is a potential security exploit
- Example: Apple's iCloud password-guessing rate limits.
  - [Ref: https://github.com/hackappcom/ibrute]
  - People often pick weak passwords; can often guess w/ few attempts (1K-1M).
  - Most services, including Apple's iCloud, rate-limit login attempts.
  - Apple's iCloud service has many APIs.
  - One API (the "Find my iPhone" service) forgot to implement rate-limiting.
  - Attacker could use that API for millions of guesses/day.
  - Lesson: if many checks are required, one will be missing.

Example Bugs

- Example: Mis-handling of error codes.
  - [Ref: https://www.mail-archive.com/dev-security-policy@lists.mozilla.org/msg05398.html]
  - Some certificate authorities rely on checking for a particular challenge file on a web server to prove domain ownership.
  - GoDaddy mis-handled error cases (when web server responded with an error to the request AND printed the requested URL as part of the error reply).
  - Treated such requests as passing.

Example Bugs

- Example: Missing access control checks in Citigroup's credit card web site.
  - [Ref: http://www.nytimes.com/2011/06/14/technology/14security.html]
  - Citigroup allowed credit card users to access their accounts online.
  - Login page asks for username and password.
  - If username and password OK, redirected to account info page.
  - The URL of the account info page included some numbers.
    - e.g. x.citi.com/id=1234
  - The numbers were (related to) the user's account number.
  - Adversary tried different numbers, got different people's account info.
  - The server didn't check that you were logged into that account!
  - Lesson: programmers tend to think only of intended operation.

Example Bugs

- Example: poor randomness for cryptography.
  - Need high-quality randomness to generate the keys that can't be guessed.
  - Android's Java SecureRandom weakness leads to Bitcoin theft.
  - [Ref: https://bitcoin.org/en/alert/2013-08-11-android]
  - [Ref: https://www.nilsschneider.net/2013/01/28/recovering-bitcoin-private-keys.html]
  - Bitcoins can be spent by anyone that knows the owner's private key.
  - Many Bitcoin wallet apps on Android used Java's SecureRandom API.
  - Turns out the system sometimes forgot to seed the PRNG!
    - A Pseudo-Random Number Generator is deterministic after you set the seed.
    - So the seed had better be random!

Example Bugs

- Example: poor randomness for cryptography (con’t)
  - As a result, some Bitcoin keys turned out to be easy to guess.
  - Adversaries searched for guessable keys, spent any corresponding bitcoins.
  - Really it was the nonce in the ECDSA signature that wasn't random and repeated nonce allows private key to be deduced.
- Lesson: be careful
  - Embedded devices generate predictable keys.
    - Problem: embedded devices, virtual machines may not have much randomness.
    - As a result, many keys are similar or susceptible to guessing attacks.
    - [Ref: https://factorable.net/weakkeys12.extended.pdf]

Example Bugs

- Example: Moxie's SSL certificate name checking bug
  - [Ref: http://www.wired.com/2009/07/kaminsky/]
  - Certificates use length-encoded strings, but C code often is null-terminated.
  - CAs would grant certificate for amazon.com\0.nickolai.org
  - Browsers saw the \0 and interpreted as a cert for amazon.com
  - Lesson: parsing code is a huge source of security bugs.

Example Bugs

- buffer overflows
- format string attacks
- return oriented programming
- etc.

Details on the Course

Content

- Part 1: Classical Attacks and Defensive Mechanisms
  - Classical Attacks
    - Buffer Overflow, Format String, ROP, etc.
  - Fundamentals
    - Taint tracking, CFI,
  - Code Analysis
    - Static analysis, Symbolic execution, fuzzing
- Part 2: Part 2: AI in Vulnerability Detection and Causal Analysis
  - AI driven code vulnerability detection
  - AI driven APT detection (i.e. causal analysis)
- Part 3: AI-Driven Advancements in Secure Software Systems

Administrivia

- Website:
  - sharif.edu/~kharrazi/courses/40815-031/
  - You are expected to check the website regularly

- Discord server

- Grading (tentatively)
  - 5% Class Participation
  - 45% HWs
  - 20% Class Project
  - 30% Final Exam

Administrivia

- Prerequisites
  - Motivation to learn
  - Motivation to learn
  - Motivation to learn
  - Motivation to learn
  - Motivation to learn
  - Understand that the devil is in the details
  - Understand that the devil is in the details
  - Data and network security + OS
  - assuming you all know how to write code!! Or will learn on your own in the semester ;)

References

- Lot’s of research papers

Policies

- Late Homework
  - One day late will cost you 25%, two days 50%, and three days 75%.
  - No homework will be accepted after the third day.
- Cellphones
  - Please turn them off before entering class.
- Cheating and Copying
  - First time you are caught you will get a zero for the task at hand.
  - Second time you are caught you will fail the course.
  - Providing your assignment to someone else is considered cheating on your behalf.
- More detail on the course webpage.

Ethics of security

- Taking a network security class is not an excuse for hacking
- “Hacking” is any form of unauthorized access, including exceeding authorized permissions
- The fact that a file or computer is not properly protected is no excuse for unauthorized access
- Absolutely no Trojan horses, back doors, or other malicious code in homework assignments

Acknowledgments/References

- [CS6.858’19] 6.858: Computer Systems Security, Frans Kaashoek and Robert Morris, MIT, Spring 2019.