This Is The End
Putting CS161 in Context: Nick's Self Defense Strategies...

• **How** and **why** do I protect myself online and in person...
  - **How** I decide what to prepare for (and what not to prepare for)
  - **Why** I've drunk the Apple Kool-Aid™
  - **Why** I use my credit card everywhere but not a debit card
  - **What** I would do as a real-world software engineer
  - And my future nightmares:
    - What do I see as the security problems of tomorrow...
My Personal Threats: The Generic Opportunist

- There are a **lot** of crooks out there
  - And they are rather organized...
- But at the same time, these criminals are generally economically rational
  - So **this** is a bear race: I don't need perfect security, I just need **good enough** security
- I use this to determine security/convenience tradeoffs all the time
  - So no password reuse (use a password manager instead)
  - Full disk encryption & passwords on devices: Mitigates the damage from theft
  - Find my iPhone turned on: Increases probability of theft recovery
My Personal Threats: The *Lazy* Nation State

- OK, I'm a high **enough** profile to have to worry about the "Advanced Persistent Threats"...
  - Trying for a reasonably high profile on computer policy issues
  - A fair amount of stuff studying the NSA's toys and other nation-state tools
  - But only at the Annoying Pestilent Teenager level: I'm worth some effort but not an extraordinary amount
- So its only **slightly** more advanced than the everyday attackers...

With one **huge** exception: Crossing borders
  - Every nation maintains the right to conduct searches of all electronic contents at a border checkpoint
My Border Crossing Policy: Low Risk Borders

• Not very sensitive borders: Canada, Europe, US, etc...
  • I use full disk encryption with strong passwords on all devices
    • Primary use is to prevent theft from also losing data
  • I have a very robust backup strategy
    • Time machine, archived backups in a safe deposit box, working sets under version control backed up to remote systems...

• So, as the plane lands:
  • Power off my devices
    • Device encryption is only robust when you aren't logged in
  • Go through the border

• If my devices get siezed...
  • "Keep it, we'll let the lawyers sort it out"
High Risk Borders

- Middle East or, if, god forbid, I visit China or Russia...
  - Need something that doesn't just resist compromise but can also **tolerate compromise**

- A "burner" iPhone SE with a Bluetooth keyboard
  - The cheapest secure device available
  - Set it up with **independent** computer accounts for both Google and Apple
    - Temporarily forward my main email to a temporary gmail account
    - All workflow accessible through Google apps on that device
  - Bluetooth keyboard does leak keystrokes, so don’t use it for passwords but its safe for everything else

- Not only is this device very hard to compromise...
  - But there is very low value in **successfully compromising it**: The attacker would only gain access to dummy accounts that have no additional privileges

- And bonus, I'm not stuck dragging a computer to the ski slopes in Dubai...
  - Since the other unique threat in those environments is the "Evil maid" attack
My Personal Threats: The Russians... Perhaps

- This is the paper that killed the Viagra® Spam business
- A $100M a year set of organized criminal enterprises in Russia...
  And they put the *organized* in organized crime...
- I've adopted a *detection and response* strategy:
  - The Russians have higher priority targets: The first authors, the last authors, and Brian Krebs
  - If anything suspicious happens to Brian, Kirill, or Stefan, *then* I will start sleeping with a rifle under my bed
Excluded Threats:
Sorta…

- Intimate Partner Threats…
  - But I’ve had at least one colleague caught up with that.

- Aggressive Nation States…
  - $50M will buy the latest version of Pegasus malcode

- The US government…
  - The surveillance powers of the US government are awesome and terrifying to behold…
Passwords and 2-Factor....

- I love security keys:
  - I have one in each of my main computers...
    and one on the keychain

- ANY site that supports multiple security keys has that as the primary 2-factor method
  - Both more convenient and more secure than the alternatives...

- I also religiously use a password manager
  - "Credential stuffing" is the biggest threat individuals face

- I personally use 1password, but others are equally good
  - In particular you can get LastPass premium through software@berkeley
The Apple Kool-Aid...

- The iPhone is perhaps the most secure commodity device available...
  - Not only does it receive patches but since the 5S it gained a dedicated cryptographic coprocessor
- The **Secure Enclave Processor** is the trusted base for the phone
  - Even the main operating system isn't fully trusted by the phone!
- A dedicated ARM v7 coprocessor
  - Small amount of memory, a true RNG, cryptographic engine, etc...
  - Important: A collection of *randomly* set fuses
    - Should not be able to extract these bits without taking the CPU apart:
      - Even the Secure Enclave can only use them as keys to the AES engine, not read them directly!
  - But bulk of the memory is shared with the main CPU
- GOOD documentation:
  - The iOS security guide is something you should at least skim....
    I find that the design decisions behind how iOS does things make *great* final exam questions
- But it isn't perfect: Nation-state actors will pay big $ for exploits
  - So keep it patched
  - And iOS 14.5: New Emoji and *turning on PAC all over the place!*
The Roll of the SEP...
The Things *too important* to allow the OS to handle

- Key management for the encrypted data store
  - The CPU has to ask for access to data!
- Managing the user's passphrase and related information
- User authentication:
  - *Encrypted* channel to the fingerprint reader/face recognition camera
- Storing credit cards
  - ApplePay is cheap for merchants *because it is secure*: Designed to have very low probability of fraud!
AES-256-XEX mode

- A **confidentiality-only** mode developed by Phil Rogaway...
  - Designed for encrypting data within a filesystem block \( i \)
    - Known plaintext, when encrypted, can't be replaced to produce known output, only "random" output
  - Within a block: Same cypher text implies different plaintext
  - Between blocks: Same cypher text implies nothing!
  - \( \alpha \) is a galios multiplication and is very quick:
    In practice this enables parallel encryption/decryption

- Used by the SEP to encrypt its own memory...
  - Since it has to share main memory with the main processor

- Opens a limited attack surface from the main processor:
  - Main processor can replace 128b blocks with **random** corruption
User Passwords...

- Data is encrypted with the user’s password
  - When you power on the phone, most data is completely encrypted
- The master key is PBKDF2(password || on-chip-secret)
  - So you need both to generate the master key
  - Some other data has the key as F(on-chip-secret) for stuff that is always available from boot
- The master keys encrypt a block in the flash that holds all the other keys
  - So if the system can erase this block effectively it can erase the phone by erasing just one block of information
- Apple implemented effaceable storage:
  - After x failures, OS command, whatever...
    Overwrite that master block in the flash securely
  - Destroy the keys == erase everything!
Background: FBI v Apple

- A "terrorist" went on a rampage with a rifle in San Bernardino...
  - Killed several people before being killed in a battle with police
- He left behind a work-owned, passcode-locked iPhone 5 in his other car...
- The FBI knew there was no valuable information on this phone
  - But never one to refuse a good test case, they tried to compel Apple in court to force Apple to unlock the phone...
- Apple has serious security on the phone
  - Effectively everything is encrypted with PBKDF2(PW||on-chip-secret):
    - >128b of randomly set microscopic fuses
    - Requires that any brute force attack either be done on the phone or take apart the CPU
  - Multiple timeouts:
    - 5 incorrect passwords -> starts to slow down
    - 10 incorrect passwords -> optional (opt-in) erase-the-phone
What the FBI wanted...

- Apple provides a modified version of the operating system for the Secure Enclave which...
  - Removes the timeout on all password attempts
  - Enables password attempts through the USB connection
  - Enables an on-line brute force attack...
    but with a 4-digit PIN and 10 tries/second, you do the math...

- Apple cryptographically signs the rogue OS version!
  - A horrific precedent:
    This is requiring that Apple both create a malicious version of the OS and sign it
    - If the FBI could compel Apple to do this, the NSA could too...
      It would make it impossible to trust software updates!
Updating the SEP To Prevent This Possibility...

- The SEP will only accept updates signed by Apple
- The FBI previously asked for this capability against a non-SEP equipped phone
  - "Hey Apple, cryptographically sign a corrupted version of the OS so that we can brute-force a password"
- How to prevent the FBI from asking again?
- Now, an OS update (either to the base OS and/or the SEP) requires the user to be logged in and input the password
  - "To rekey the lock, you must first unlock the lock"
  - The FBI can only even attempt to ask before they have possession of the phone since once they have the phone they must also have the passcode
  - So when offered the chance to try again with a "Lone Wolf's" iPhone in the Texas church shooting, they haven't bothered
- At this point, Apple has now gone back and allows auto-updates for the base OS
  - (but probably not the SEP)
The Limits of the SEP...

The host O/S

• The SEP can keep the host OS from accessing things it shouldn't...
  • Credit cards stored for ApplePay, your fingerprint, etc...

• The SEP can \textit{use} the random secret but not read it...
  • Can encrypt with it but can't read it

• But it can't keep the host OS from things it is supposed to access
  • All the user data when the user is logged in...

• So do have to rely on the host OS as part of \textit{my} TCB
  • Fortunately it is updated continuously when vulnerabilities are found
    • Apple has responded to the discovery of very targeted zero-days in <30 days
  • And Apple has both good sandboxing of user applications and a history of decent vetting
    • So the random apps are \textit{not} in the Trusted Base.
The SEP and Apple Pay

• The SEP is what makes ApplePay possible
  • It handles the authentication to the user with the fingerprint reader/face reader
    • Verifies that it is the user not somebody random
  • It handles the emulation of the credit card
    • A "tokenized" Near Field Communication (NFC) wireless protocol
    • And a tokenized public key protocol for payments through the app

• Very hard to conduct a fraudulent transaction
  • Designed to enforce user consent at the SEP

• Disadvantage: The fingerprint reader is part of the trust domain
  • Which means you need special permission from Apple to replace the fingerprint reader when replacing a broken screen
I love ApplePay...

- It is a **faster** protocol than the chip-and-signature
  - NFC protocol is designed to do the same operation in less time because the protocol is newer
- It is a **more secure** protocol than NFC on the credit card
  - Since it actually enforces user-consent
- It is more **privacy sensitive** than standard credit card payments
  - Generates a unique token for each transaction: Merchant is not supposed to link your transactions
- Result is its low cost:
  - Very hard to commit fraud -> less cost to transact
- I use it on my watch all the time
Transitive Trust in the Apple Ecosystem...

- The most trusted item is the iPhone SEP
  - Assumed to be rock-solid
  - Fingerprint reader/face reader allows it to be convenient
- The watch trusts the phone
  - The pairing process includes a cryptographic key exchange mediated by close proximity and the camera
  - So Unlock the phone -> Unlock the watch
- My computer trusts my watch
  - Distance-bounded cryptographic protocol
  - So my watch unlocks my computer
- Result? I don't have to keep retyping my password
  - Allows the use of strong passwords everywhere without driving myself crazy!
Credit Card Fraud

• Under US law we have very good protections against fraud
  • Theoretical $50 limit if we catch it quickly
  • $0 limit in practice

• So cost of credit card fraud for me is the cost of recovery from fraud
  • Because fraud _will happen_: 
  • The mag stripe is all that is needed to duplicate a swipe-card 
    • And you can still use swipe-only at gas pumps and other such locations
  • The numbers front and back is all that is needed for card-not-present fraud 
    • And how many systems

• What are the recovery costs? 
  • Being without the card for a couple of days... 
    • Have a second back-up card
  • Having to change all my autopay items... 
    • Grrrr....
But What About "Debit" Cards?

• Theoretically the fraud protection is the same...

• But two caveats...
  • It is easier to not pay your credit card company than to claw money back from your bank...
  • Until the situation is resolved:
    • Credit card? It is the credit card company's money that is missing
    • Debit card? It is your money that is missing

• Result is debit card fraud is more transient disruptions...
So Two Different Policies...

- Credit card: Hakunna Matata!
  - I use it without reservation, just with a spare in case something happens
  - Probably 2-3 compromise events have happened, and its annoying but ah well
    - The most interesting was $1 to Tsunami relief in 2004...
      was a way for the attacker to test that the stolen card was valid

- Debit card: Paranoia-city...
  - It is an ATM-ONLY card (no Visa/Mastercard logo!)
  - It is used ONLY in ATMs belonging to my bank
    - Reduce the risk of "skimmers": rogue ATMs that record cards and keystrokes
And Banking Information...

• **Watch** your bank account transactions
  • In case of fraud, you have protection but you need to notice
  • Bank accounts are particularly vulnerable:
    • The information on a cheque is all the data needed to transfer to/from an account!
Assume *GASP* I have to Work for a Living...

- I get to my new work environment and have to adopt/start a project...
  - I am going to want to prevent as many security problems as possible before they become problems...

- Two options:
  - New Project
  - Existing Project
New Project: Chose Your Language...

• Question: "Do I need real-time (<10ms) response?"
  • More precisely: If my program pauses for 10-50ms does anyone care?
  • If the answer is "NO", I can use a garbage collected language
    • My personal preference will be go:
      The concurrency model is such that I can easily take advantage of modern multicore systems
    • And how many bugs did the type-system catch?

• If the answer is "Yes"
  • The old answer would be "use C/C++": it is the lack of a GC that tended to require C/C++ here...
  • But today: Rust. Learning curve is a @#)(*#)(* (So I haven't learned it yet)
Existing Project: C/C++

- **Step 1: Turn on all compiler and OS mitigations in the build flow**
  - Stack canaries: Stop simple stack overflows
    - There are "security" appliances from major vendors like Cisco that don't do this!
  - ASLR: adds defense in depth
    - Need two vulnerabilities: one which allows reading memory in order to break randomization
  - If possible: Run on a 64b platform & OS
  - If the stars align: Run on Arm 8.3 and turn on PAC

- **Step 2: Add rate limits**
  - Change any auto-restart after crash to add an increasing delay:
    - First 10 immediate
    - Later an exponential increase (1, 2, 4, 8, 16 minutes...)
  - Seriously disrupts brute-force attacks
Existing Project: C/C++

- Look at the continuous integration testing flow...
  - If you don't have such a testing flow already, create one!
    - Both explicit test cases, testing code, and fuzz testing... Do it all!

- Once you do, add a few more machines to your test infrastructure...
  - Now on those machines run the same tests but within `valgrind` or a similar tool
  - Valgrind slows down the program by an order of magnitude so you can't run it on your main flow, but it will catch a lot of memory problems before they become problems!

- Aside: Computers are cheap!
  - "Oh, to do this I need a 8-core computer with 32 GB of RAM and a 1 TB ssd. And ideally quiet because I need to stick it under my desk? Hey boss, I need an $850 computer..."
Existing Project: Command Injection

- Grep for every call to `system` and direct SQL
  - Search the entire code-base
- Now refactor *every instance* into a call to `execve` or prepared statements
  - Do *not accept the excuse* that "this particular invocation is known safe" because...
- Now do some include/compiler/language tricks so any code which calls `system` etc fails to build!
  - And if you are doing a new project, make sure that is already in place!
New or Existing Project: Web Security...

- Time to block common web exploits:
  - Turn on HTTPS only: Use LetsEncrypt

- Actually require a modern browser to access:
  - Enables mitigations not otherwise possible

- Set *all* cookies right:
  - Every one should have *secure* and *same-site* set
    - Don't want to have to distinguish between "important" and "unimportant"!

- Ensure that all toolkits have CSRF protection as well
  - Because the boss may overrule the "only modern browsers" restriction!
New or Existing Project: Content Security Policy...

- Now the annoying part: Enabling a content-security-policy!
  - Well, annoying if an existing project
- For CSP to prevent XSS we can't have *any* inline JavaScript!
  - All JavaScript needs to be in separate files not inline in order to allow CSP to prevent xss attacks
- Also make sure *all* user input passes through the XSS-busting filter rules
  - It may be a "denylist" rule but it is a well structured one
And a bit more hardening...

Containment and sandboxing

- All servers should run in a **chroot** jail and execute as a minimum privileged user process after that
  - Limits access to a subsection of the filesystem:
    No more path traversal problems and a lot of mitigation
  - Limit the rights of the user account:
    Even a compromise now has to work within bounds
- Even better, can you use the chrome sandbox?
  - Limits a program to only a fixed set of defined capabilities
  - Idea is even if you exploit the program the attacker has to escape the sandbox as well
And Now:
Ask Me Anything!