This Is The End

All these are Blue Slides...

- There may be important takeaways however...
  - Both in class and in life
- Part 1: Plane Crashes
  - A very interesting story of safety failures
- Part 2: Nukes
  - Big and scary and interesting security properties
- Part 3: Putting 161 into practice

Safety and Security

- Safety and Security are closer than two sides of the same coin...
  - Both have the objective of ***maintaining system properties*** under all conditions
- The only real difference are the source of deviance
  - Security we deviate because of ***deliberate action by an adversary***
  - Safety we deviate because of ***chance, failure, and inadvertent actions***

The Airline Industry...

- A rough rule of thumb I once heard about an airline's costs:
  - 1/3 for fuel
  - 1/3 for people
  - 1/3 for the aircraft
- And the business is brutally competitive
  - Warren Buffett once joked that if he had a time machine he'd take a shotgun to the runway at Kitty Hawk to save subsequent investors a huge amount of money
- So when developing a new aircraft...
  - Make it ***cheap***:
    Limit the necessary retraining
    Limit the fuel costs

The Boeing 737...

- Probably the most successful commercial airliner
  - First flown in 1967, over 10,000 of various types sold!
- The first version: 737-100 and 737-200
  - Notice the relatively tiny jet engine...
    We will get back to that later
- Consequence of a design choice:
  - Wing mounts to the low part of the plane...
  - And can't have the plane too high off the ground because you needed to unload luggage on unimproved airfields
  - But at the same time, under-wing engines allowed the airframe to evolve:
    Stretch it to make it longer with each revision

Then the "737-Classic": -300, -400, -500

- First major revision
  - Sold from 1984-2000
- Bigger, Better, More Efficient
  - Major change in the concept of how the engines are mounted...
- Not quite a "separate plane"
  - But substantial retraining necessary for pilots & crew to shift from the original to the "classic"

Then the 737-NG -600, -700, -800, -900

- Almost a new plane
  - Bigger wings, new cockpit, new engines, more people etc...
  - Notably the "flat bottomed" engines to get them to fit!
- First on sale in 1997
- Really a "new plane"
  - Completely different cockpit for the pilots

In The Meantime: Enter Airbus

- The A320 family
  - Entered service in 1987...
- Slightly bigger than a 737
  - And claimed to be cheaper...
  - And not so close to the ground on the wings/engines
- A major new version entered service in 2016: the A320neo (New Engine Option)
  - Moderate pilot retraining necessary:
    it flies different from the A320 due to significantly larger engines
  - But they had higher wings to begin with so it was easier to put on bigger engines

Why Larger Engines?

- Bigger engines that burn hotter are ***much*** more fuel efficient
  - Thermodynamic efficiency of the engine core
  - Bigger bypass fans move more air
- Core problems:
  - Efficiency of the core is improved by making it bigger
  - Thrust goes up by moving a bigger volume of air ("high bypass")
    - $E=mv^2$, but $p=mv$
  - And the area of the engine is ~r^2

The 737-MAX program

- In 2011, Boeing responded to the A320...
  - American Airlines just ordered a bunch of A320ceo and A320neo planes
- Effectively sidelined the planned 737 replacement...
  - It would have been close to a "baby Dreamliner (787)"
  - And instead decided to "re-engine" and improve the 737-NG in other ways
  - Goal was 14% improvement in efficiency
- Fatal Decision #1:
  - Unlike the A320neo, there must be ***no significant pilot retraining***:
    If a pilot is certified for a 737-NG,
    the pilot should be able to fly the
    737-MAX with just a bit of written material

Fatal Decision #2: Larger Engines

- Went from a 61" engine to a 69" engine
  - But the previous 61" engine already had the minimum available ground clearance!
  - Oh, and still not as good as the A320neo, which has 20% higher bypass
- Forced to move the engines further forward and upward
  - Which changes the dynamic balance of the aircraft
  - Other option would have required effectively reengineering the entire wing setup
    - At which point, why not just design a new plane from scratch: the initial 737 design had much much smaller engines
- Dynamic balance changes are significant
  - Significantly higher tendency to want to pitch the nose up under acceleration

Fatal Decision #3:
The "Software" Fix

- If the plane goes too nose-up, it wants to stall
  - aka, "just drop from the sky", major not-good
- The larger nacels for the engines also act like wings
  - Even further increasing the propensity to stall
- "Hey, we have a computer that can fly the plane..."
  - So lets modify the computer to have the plane try to adjust itself so it flies like the 737-NG:
    MCAS: Maneuvering Characteristics Augmentation System

Fatal Decision #4: Engineering the software fix

- In an Airbus, the computer is the boss
  - So the computer design is very paranoid: Each computer can listen to all relevant sensors
- In a Boeing, the ***pilot*** is supposed to be the boss
  - So although there are two flight computers, each one only listens to its ***own set of sensors***...
  - Because on all previous 737s, the computer ***mostly*** acted as an advisor
    - Which means you can be fairly slack with things
- MCAS program stuck with the 737 design
  - So if the computer saw that ***it's*** pitch sensor said the nose was too high, it would act
- Plus other factors:
  - If you fight the computer on the 737-NG, the computer gives up
  - But on MCAS, it just tries again... and again... and again...

Fatal Decision #5: Regulatory Capture

- In the old days, the FAA certified planes...
  - But this requires significant expertise
  - And the government can't pay nearly as much as Boeing
- Now, the aircraft is ***mostly*** self certified by the company...
  - And even here they screwed up!
- MCAS was determined to create a "hazardous" condition if it erroneously activated at the wrong time...
- ***Yet they kept the single-sensor design!***

So How To Crash a 737-Max....

- Have the angle of attack sensor on one side of the plane break
  - On the same side as the currently active flight computers
- Makes the plane think the nose is pitching up
  - So MCAS pitches the plane down...
- The pilot fights MCAS to pitch back up...
  - So MCAS pitches the plane ***further down...***
- Lather/Rinse/Repeat...
  - Until the plane goes nose-first into the earth

Magnifying Culpability: Blaming the user...

- After the first crash, Boeing blamed the pilots
  - "Yeah, we didn't tell them about MCAS, but it should have been treated just like a runaway stabilizer, where the autopilot goes wonky..."
- But that wasn't true!
  - Runaway trim, you fight it and it stops fighting
- And they are ***still*** blaming the pilots!

Asked about what led to the safety flaws in the 737 Max, Muilenburg said Boeing didn't make any mistakes in its design of the planes. "There was no surprise or gap or unknown here or something that somehow slipped through the certification," Muilenburg said. "We know exactly how the airplane was designed, and we know exactly how the airplane was certified."

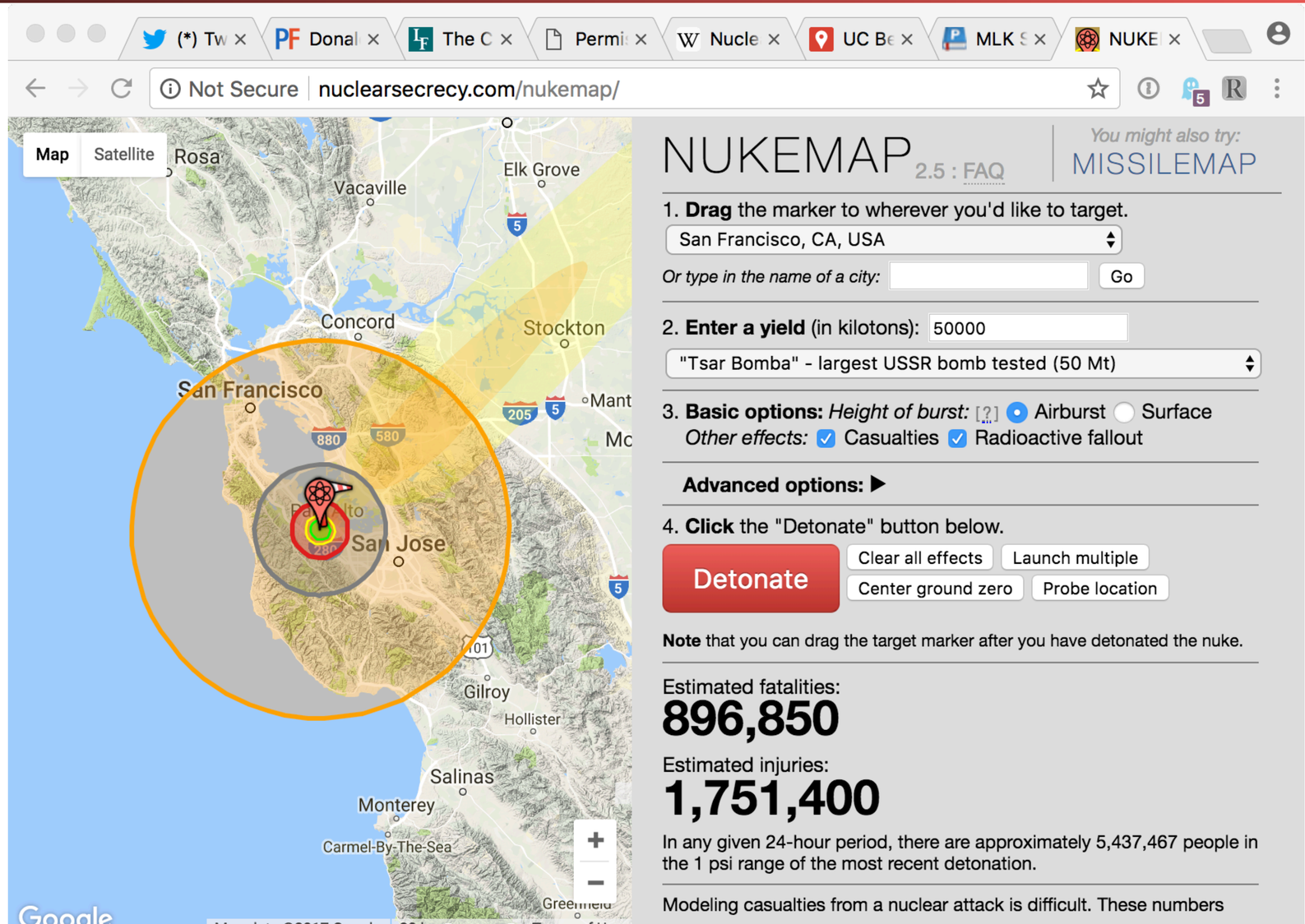
The CEO said both crashes were caused by a "series of events" that included erroneous sensor data being fed into the maneuvering characteristics augmentation system, or MCAS, an anti-stall system that played a role in both crashes. "There were actions — or actions not taken — that contributed to the final outcome," he said, alluding to the role of the pilots.

Conclusions and non-tested Takeaways...

- It is a massive Charlie Foxtrot of epic proportions
- If it was an American or Southwest plane involved, there would already be indicted ***executives*** rather than just the single chief test-pilot
- Every system on the 737-Max that changed needs to be viewed with suspicion
  - And I won't fly on one for at least 3 years post recertification
- Oh, and don't let the "Business" types take over an engineering firm
- Oh, and never buy a Tesla:
  Their software development flow and autonomous concept is similarly fatally flawed, just with a lower body count

Why talk about nukes?

- Nukes are big and scary and in the news...
  - But have interesting security and safety properties
- Lots of material ~~stolen~~ borrowed from Steve Bellovin's excellent talk on PALs



How a Nuclear Weapon Works...

- 1940s-level technology...
  - A hollow sphere of fissile material
    - Plutonium and/or Plutonium + Uranium
  - Use this as a primary to ignite a Teller/Ulam secondary to make it a hydrogen bomb...
- Very careful sequencing needed
  - D/T pump to fill the hollow with Deuterium & Tritium ("Boost gas")
    - Not needed for the earliest bombs, but most modern bombs need boosting to work
  - Initiator sprays neutrons to start the chain reaction
  - Detonator needs to trigger multiple points on the explosive shell
    - Squiggly-traces of explosive so that all around the shell everything detonates at once
- We could make a 194 "design a nuke" class and probably get something that would work! Even without physics students!
  - Physics 111 electronics lab: "All the EE a physicist needs to build his nuclear weapons"

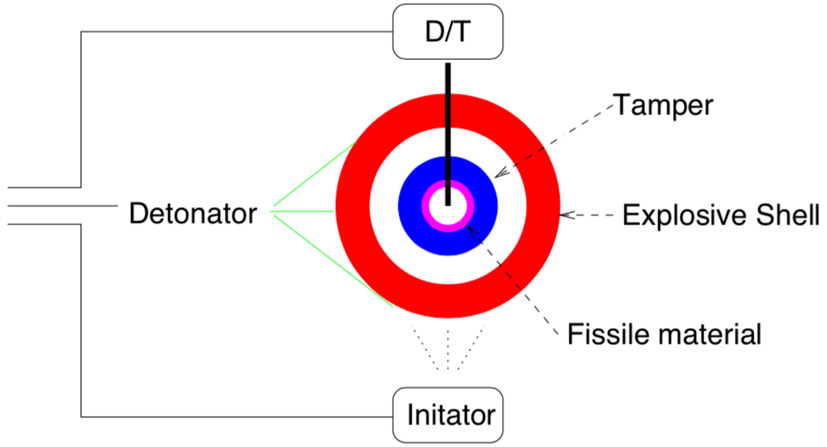
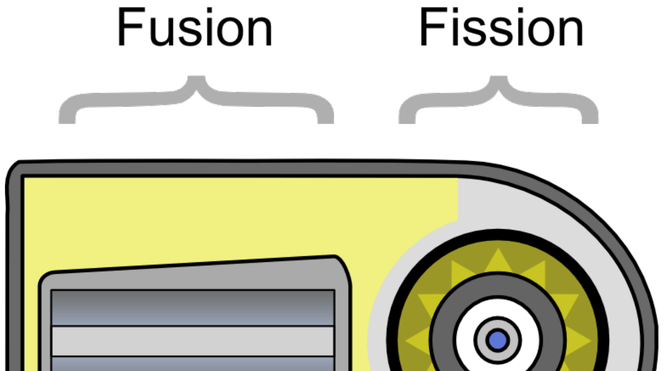


Diagram by Steve Bellovin

And H-Bombs...

- A "Tellar/Ulam" 2-stage device:
  A A-bomb ignites a fusion stage
  - Fusion stage has Lithium Deuteride...
    - Neutrons and pressure from the A-bomb convert the Lithium to Tritium
    - Then Deuterium/Tritium fusion makes it go boom!
- Still 1960s technology!
  - Biggest issue overall is ***materials***:
    6 or 7 countries have built H-Bombs



And How To Deliver Them...

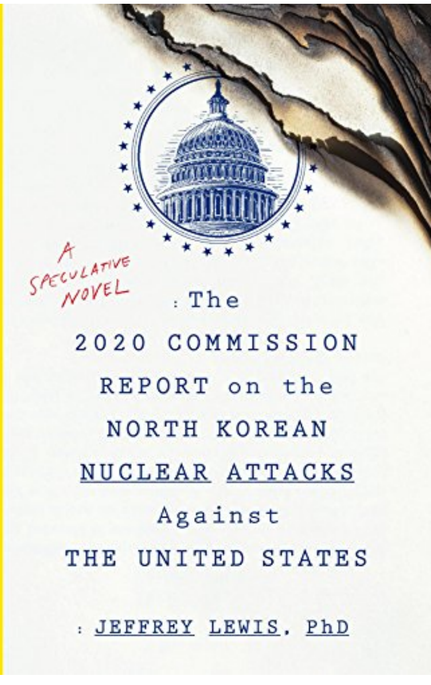
- Stick em on a rocket
  - This ***is*** rocket science: It is far easier to build the nuke than build the ICBM...
  - Alternatively, stick it on an unmanned miniature airplane ("Cruise Missile") or just hang it under a plane as a old-fashioned bomb
- Then stick the rocket on something
  - In a hardened silo
    - But the other side can drop a nuke on it...
  - On a truck
  - In a sub
  - On a plane...

The Problem:
When To Use Nukes...

- Nuclear weapon systems can fail in two ways:
  - Launch the nukes when you shouldn't...
  - Fail to launch the nukes when you should...
- The latter is (badly) addressed by how our nuclear decision making happens
  - "Launch on warning": If we ***think*** we are under attack, the President has a couple minutes to decide to order a nuclear strike before the attacker hits our ICBMs!
    - This is often regarded as ***insanely*** stupid: We have both nuclear bombers with long-range cruise missiles and nuclear armed submarines, both of which ***will*** be able to launch enough retaliatory hellfire
  - Far better is the "French model" (cite @armscontrolwonk):
    "We have subs. You nuke us ***or*** attack our strategic weapons and we nuke you":
    - This removes the time pressure which can cause errors

"Launch on Warning" and North Korea...

- Let us assume that North Korea's leadership are ***rational*** actors
  - They act in what they perceive as their self interest: survival!
- North Korean leadership ***will eventually lose*** a war with South Korea and the US
  - So they may be provocative, but they want to make ***sure*** the US and South Korea won't start a war
- Nukes are a critical deterrent for them
  - Especially when Donald Trump didn't seem to care that a war would kill hundreds of thousands in South Korea
- IRBMs and ICBMs are as important as the nukes themselves!
  - Need to be able to hit the US bases in Okinawa and Guam as military targets
  - And last year Mar-a-lago and Washington DC to dissuade Trump personally: The Hwasong-15 ICBM can just barely range South Florida.
- "***Empathy*** for the devil"
  - Computer security is adversarial, think about your adversary's needs, wants, and desires



Launch on Warning and the US C&C Structure

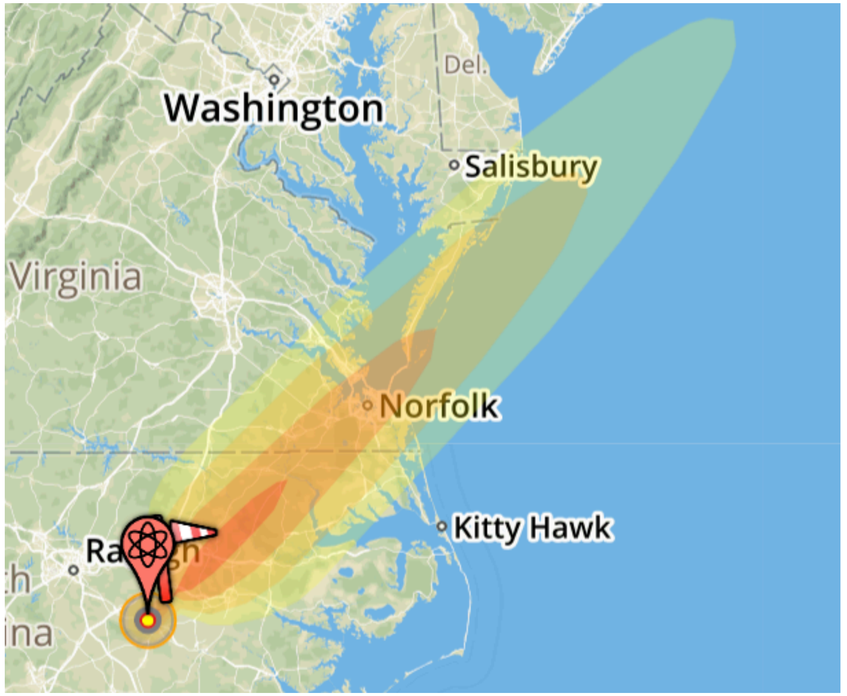
- The President has three items:
  - A “biscuit” of authentication codes kept on his person
  - The “football”: containing a menu of options for ordering a nuclear strike
  - An encrypted secure phone
- The President has a bad day…
  - He calls over the football
    - Picks out the menu option he wants to use..
  - He calls NORAD on the phone
    - Taking out the biscuit, opening it, and getting the authentication code of the day
    - Saying what menu option he wants
  - < 5 minutes later, the ICBMs leave their silos
    - And there is no “recall code”

The Interesting Problem: Limiting Use

- Who might use a nuke without authorization?
  - Our "allies" where we station our nukes
    - Original motivation: Nukes stored in Turkey and Greece...
      And we still probably have nukes stored in Turkey!
  - Someone who can capture a nuke
    - This is what sold the military on the need for the problem:
      We had nukes in Germany which ***would*** be overrun in case of a war with the USSR
  - Our own military
    - General Jack D Ripper scenario
- The ***mandated*** solution:
  - Permissive Access Link (PAL)

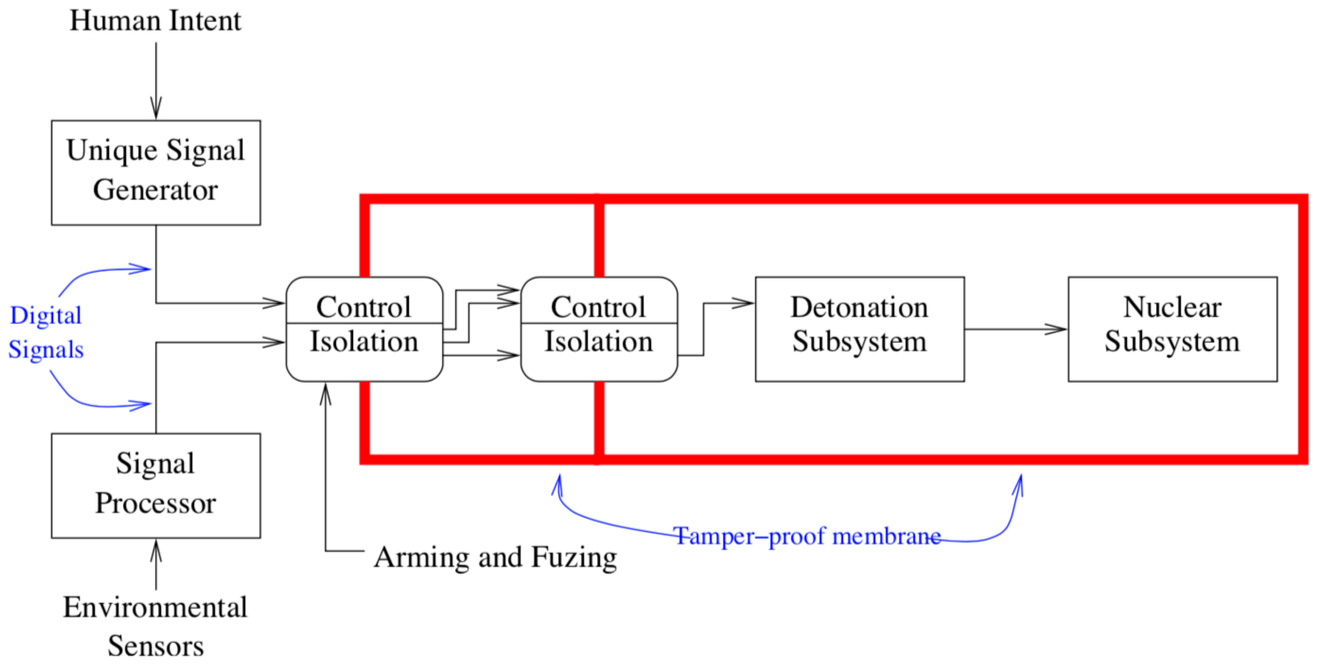
Nuke Safety Features

- One-point safety – no nuclear yield from detonation of one explosive charge.
- Strong link/weak link –
  - strong link provides electrical isolation;
  - weak link fails early under stress (heat, etc.)
- Environmental sensors – detect flight trajectory.
- Unique signal generator – digital signal used for coupling between stages.
- Insulation of the detonators from electrical energy.
- “Human intent” input.
- Tamper-resistant skin (Personal bet: dump the boost gas)
- Use Control Systems
- Not always the case: In 1961 in South Carolina a B52 broke up
  - One of the two 4MT bombs ***almost*** detonated on impact, since it thought it was being dropped!



Bomb Safety Systems

- We have a "trusted base"
  - Isolated inside a tamper-detecting membrane
    - Breach the membrane -> disable the bomb
- We have human input
  - Used to generate a signal saying "its OK to go boom"
    - The user interface to the PAL can follow the same path/concepts
- We have critical paths that we can block
  - Complete mediation of the signal to go boom!



Unique Signal Generator

- Part of the strong link
  - Prevent any detonation without clear, unambiguous showing of “human intent”
- A ***safety*** system, not a security system
- Looks for a 24-bit signal that is extremely unlikely to happen during any conceivable accident. (Format of input bits not safety-critical)
  - Accidents can generate random or non-random data streams
  - Desired signal pattern is unclassified!
- Unique signal discriminator locks up on a ***single*** erroneous bit
- At least partially mechanical

PALs

- Originally electromechanical. (Some weapons used combination locks!)
- Newest model is microprocessor-based. There may still be a mechanical component.
  - Recent PAL codes are 6 or 12 digits.
- The weapon will permanently disable itself if too many wrong codes are entered.
- PALs respond to a variety of codes – several different arming codes for different groups of weapons, disarm, test, rekey, etc.
- It was possible, though difficult, to bypass early PALs.
  - Some even used false markings to deceive folks who didn't have the manual.
- It does not appear to be possible to bypass the newest "CAT F" PAL.
  - Modern bombs don't work without the tritium boost-gas:
    If you blow the gas you disable the nuke. Don't know if this is done or not

How are PALs built?

- We don't know, but some informed speculation from Steve...
- It is ***most likely*** based around the same basic mechanism as the unique signal generator
  - Gives a single point of control already in the system
  - Reports about it indicate that it was successfully evaluated in isolation
  - Take advantage of the existing trusted base of the tamper-resistant barrier around the warhead to protect the device

Deployment History

- Despite Kennedy's order, PALs were not deployed that quickly.
  - In 1974, there were still some unprotected nukes in Greece or Turkey
- PALs and use control systems were deployed on US-based strategic missiles by then
  - But the launch code was set to 00000000
  - Rational: the Air Force was more worried about failure to launch!
- A use control system was added to submarine-based missiles by 1997
- In 1981, half of the PALs were still mechanical combination locks

Steve Bellovin's Lessons Learned
Aka ***Takeaways***

- Understand what problem you're solving
- Understand ***exactly*** what problem you're solving
- If your abstraction is right:
  you can solve the key piece of the overall puzzle
- For access control, find the One True Mandatory Path — and block it.
  - And if there is more than one, you're doing it wrong!
- What is the real TCB of our systems?

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

Click Trajectories: End-to-End Analysis of the Spam Value Chain

Kirill Levchenko* Andreas Pitsillidis* Neha Chachra* Brandon Enright* Márk Félegyházi‡ Chris Grier†
Tristan Halvorson* Chris Kanich* Christian Kreibich†◇ He Liu* Damon McCoy*
Nicholas Weaver†◇ Vern Paxson†◇ Geoffrey M. Voelker* Stefan Savage*

- 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.
- Agressive 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...

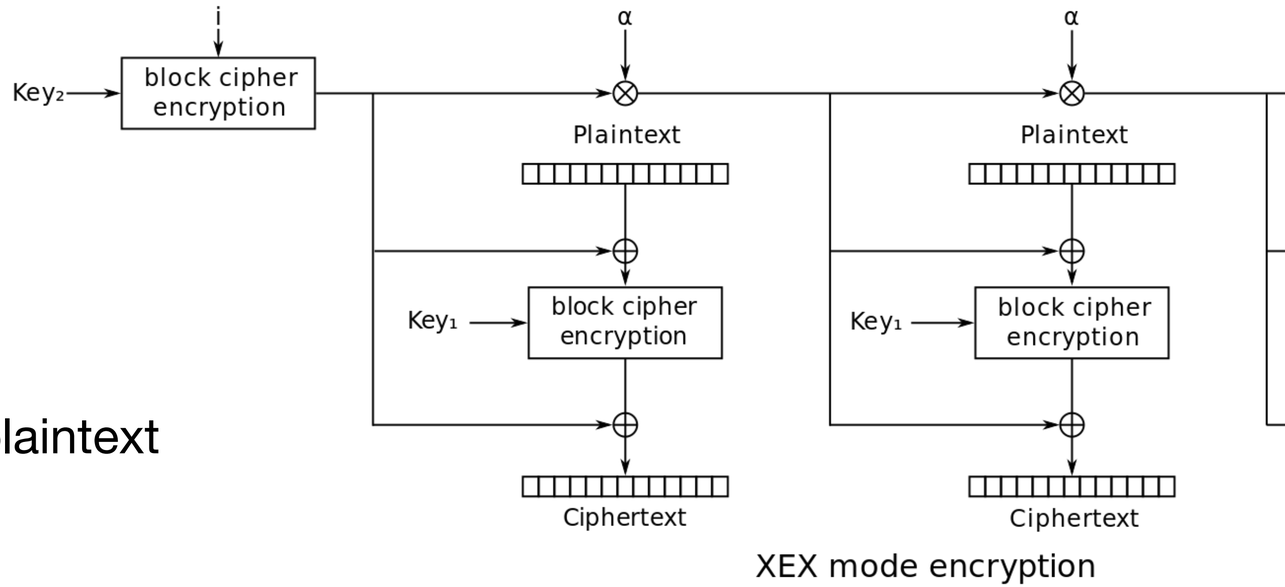
- The iPhone is perhaps the most secure commodity device available...
  - Not only does it receive patches but since the 5S it gained a dedicated cryptographic coprocessor
- The ***Secure Enclave Processor*** is the trusted base for the phone
  - Even the main operating system isn't fully trusted by 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...
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!
  - α is a galios multiplication and is very quick:
    In practice this enables parallel encryption/decryption



XEX mode encryption

- 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 ***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 ***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 ***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...

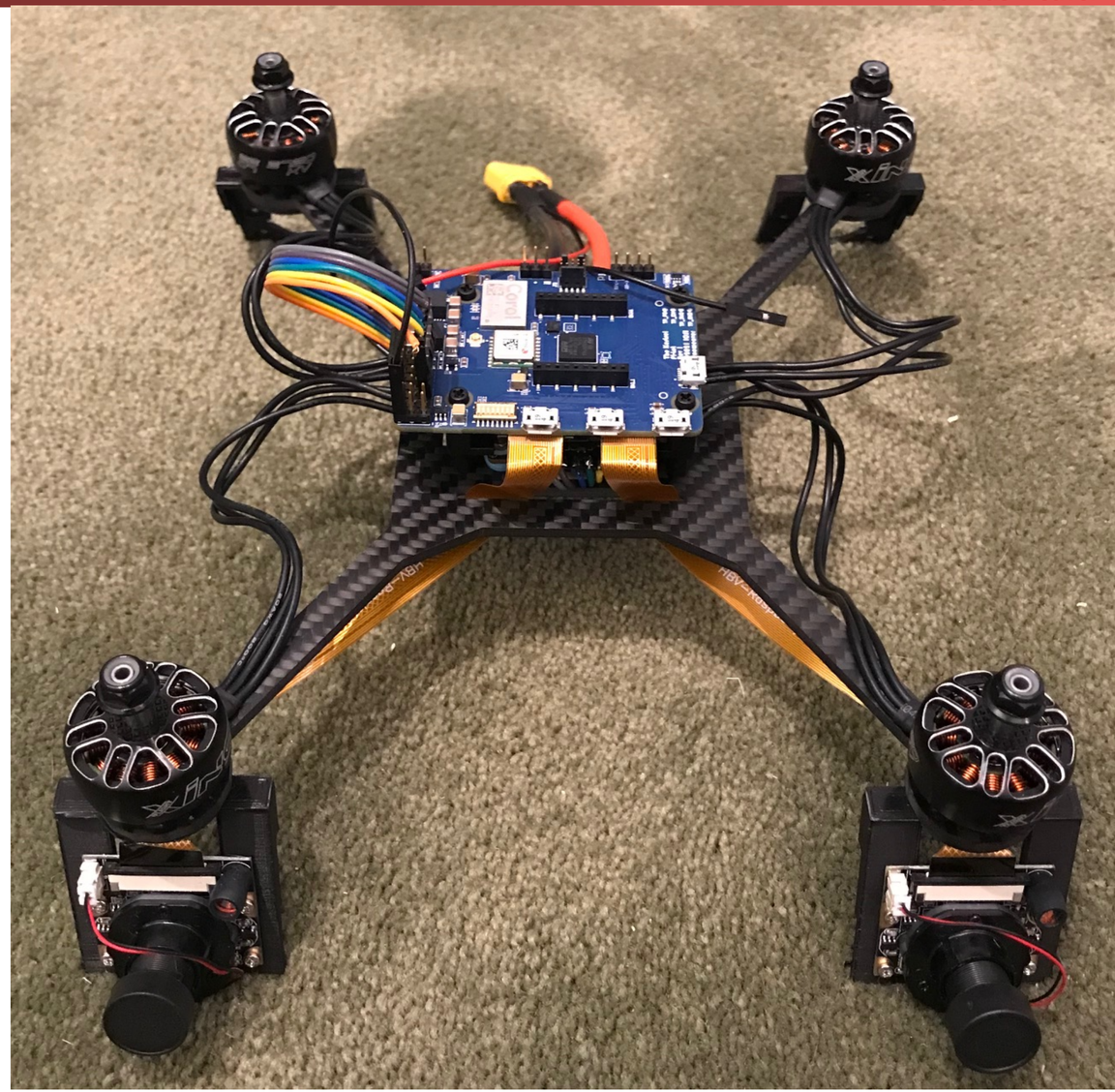
- I get to my new work environment and have to adopt/start a project...
  - I am going to want to prevent as many security problems as possible before they become problems...
- Two options:
  - New Project
  - Existing Project

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)

And Even If You Need C...

- I'm leaving Berkeley to found a startup: ***Skerry Technologies***
  - Focus is building small, cheap, autonomous drones
- The initial vision processing is ***very*** performance sensitive
  - I want to do stereo/optical flow at 1080p/30FPS
  - And the interfaces for the camera are very C-centric as well
- Going to do the front end vision in C++ but...
  - As a library from go so the rest of the process can be in go
- Even if you can't eliminate C, minimize it to the greatest degree possible!



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
  - Least Privilege is your friend!

And Now:
Ask Me Anything!