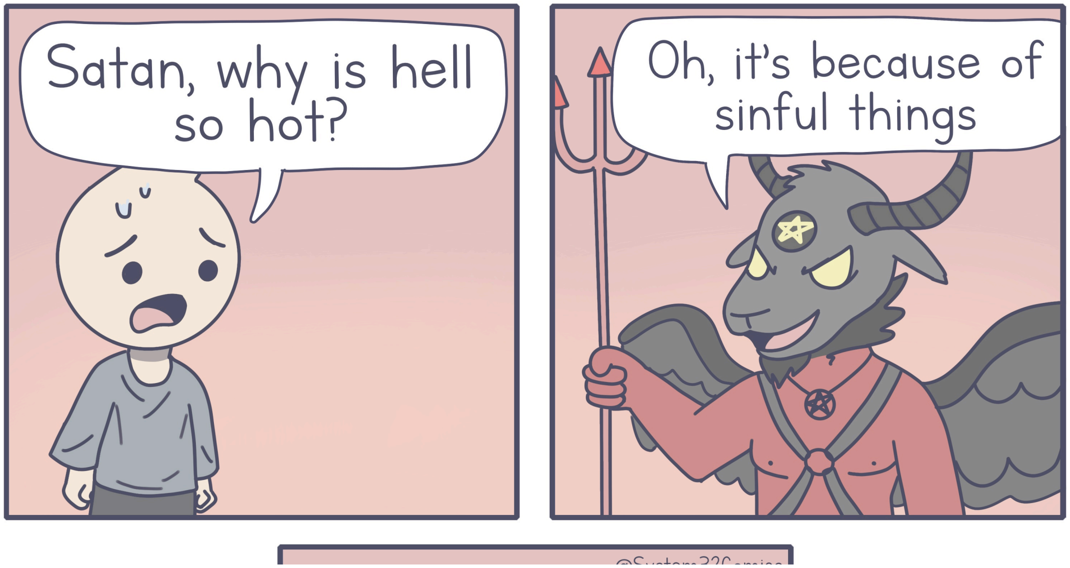
More Dealer's Choice



Announcements for the Last Lecture...

- The last lecture will be special...
- The first fraction will ***not*** be recorded
  - Instead it is how I would design a solution for Project 3:
    Based on my previous solution and tweaks for new requirements
- The rest will be an "Ask Me Anything"

Funky Hardware SideChannels...

- The Meltdown and Spectre Intel bugs...
  - Both were effectively side-channels
- The key idea:
  - You could trick the speculative execution engine to compute on memory that you don't own
  - And that computation will take a different amount of time depending on the memory contents
- So between the two, you could read past isolation barriers
  - Meltdown: Read operating system (and other) memory from user level
  - Spectre: Read in JavaScript from other parts of the web browser

How Meltdown Works...

- In a CPU, precise exceptions are hard: that is, stopping things when something "happens" at a specific instruction
- x86 actually provides two page table hardware pointers
  - One for the current user program, one for supervisor mode
  - Allows the OS to have virtual memory for the interrupt handler and other things
- Concept behind meltdown:
  - x86 allows "load whatever that memory location points to + base register"
- Do a bunch of loops that are always taken
  - Now the CPU will predict that the next time this loop is taken...
- Now do a load of memory you aren't supposed to read belonging to the OS
  - CPU guesses branch will taken, so is just going to do it speculatively.
    Only when it finally writes to a register will the exception be checked
- Now have the results of that load do a load to memory you are supposed to read
  - But dependent on what was in the memory you weren't supposed to read
- Now CPU finds that branch wasn't taken after all
  - And so nothing happens, neither the illegal load nor the "load not taken"

But Something *Did* Happen!

- The final "load not taken" got taken!
  - So it will be cached
- And that load was dependent on the illegal load
- So we can discover which "load not taken" got actually taken!
  - Allowing us to read memory we aren't supposed to!
- Fix involves the OS flushing the TLB and presenting a dummy OS page-table when returning to a user process
  - Greatly increasing the cost of a context switch or interrupt

Countering Meltdown and Spectre...

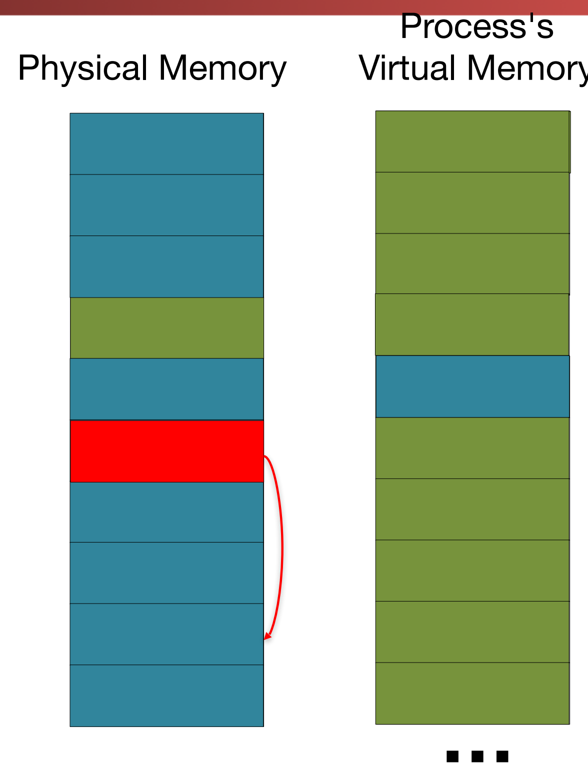
- Meltdown was really a bug...
  - TLB check not acted on right away
- Spectre and variants are really features of caches
  - You could train a branch-prediction buffer that you won't do it..
    Then you did it anyway
- Countering Spectre requires flushing ***all*** caches on ***every*** context switch
  - No such thing as a lightweight isolation barrier
  - This is why chrome & firefox eat ram with abandon:
    Every web origin runs in a different OS process

The Ultimate Page-Table Trick: Rowhammer

- An ***unspeakably cool*** security vulnerability…
- DRAM (unless you pay for error correcting (ECC) memory) is actually unreliable
  - Can repeatedly read/write the same location ("hammer the row" and eventually cause an error in ***some physically distinct memory location***
- Can tell the OS "I want to map this same block of memory at multiple addresses in my process…"
  - Which creates additional page table entries, lots of them. Lots and lots of them. Lots and lots and lots and lots and lots of them...
- Enter ***Rowhammer***
  - It seems all vunerabilities get named now, but this one is cool enough to deserve a name!
- Touches on virtual memory, hardware failures, and breaks security

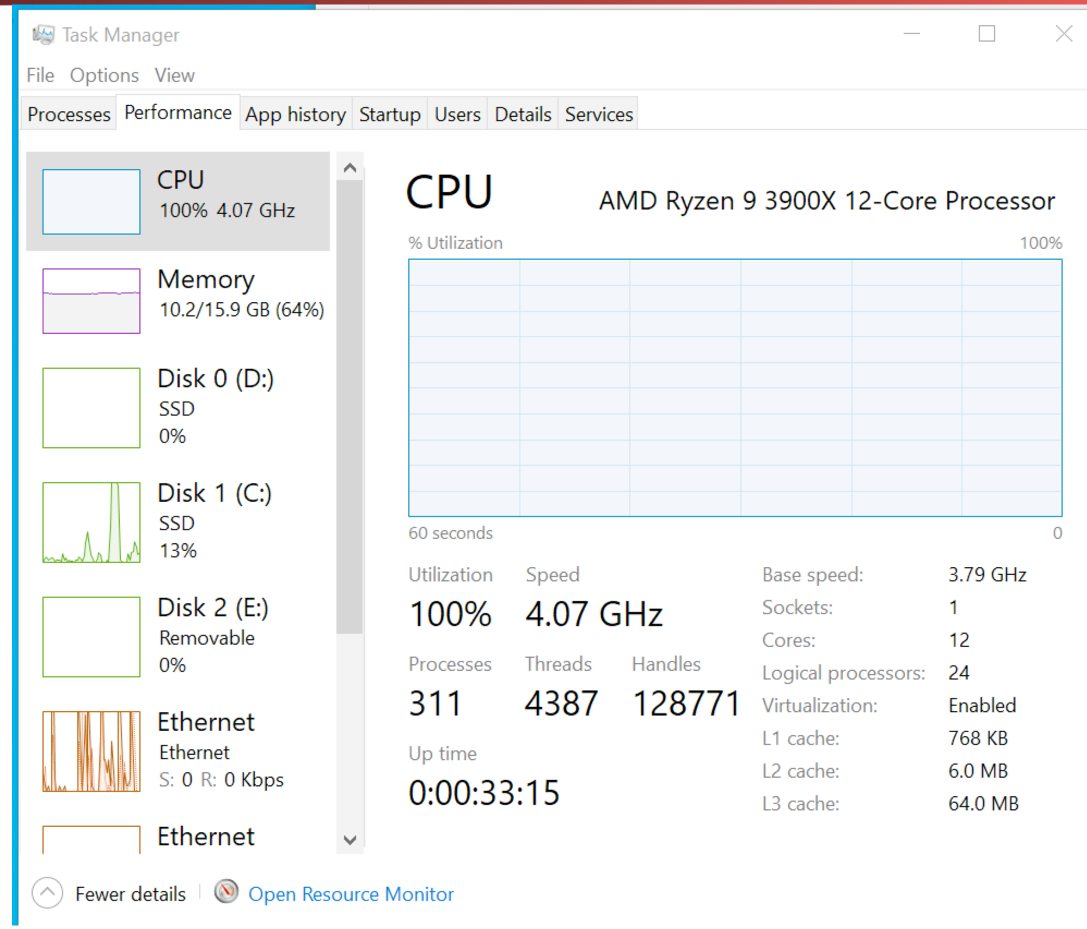
How RowHammer Works

- Step 1: Allocate a single page of memory
- Step 2: Make the OS make a gazillion page-table entries pointing to the same page
- Step 3: Hammer the DRAM until one of those entries gets corrupted
  - Now causes that memory page to point to a set of page table entries instead
- Step 4: ***Profit***
  - Well, the ability to read and write to any physical address in the system, same difference



Analyzing the Stolen UCOP Data

- Some bad actor stole >4 GB of data from UCOP
  - Basically everything that was on the "secure" file transfer server in December
- The bad actor released at least some of this data ***publicly***
  - As a 4GB compressed archive that anyone can download
- What information about ***me*** was in the archive?



What I need to know...

- I already know my social security # got breached
  - They told us that...
  - But I've got fraud alerts & freezes in place already
- But what other information?
  - Address? Phone #? Things I don't know about?
  - Tax information?
  - ***Banking information?***
    - The numbers on the bottom of a check are all an attacker needs to make fake checks

The Nature of the Dump...

- A ***lot*** of pdf files
  - PDFs are a pain to search, need to convert to text
- A ***lot*** of data tables
  - Some as comma-delimited text, some as excel spreadsheets, some in stada format
- Need to convert it to something reasonable
- Google around...
  - Nice linux OCR pipeline cobbled together:
    PDF -> images -> OCR text
  - pandas can read both xlsx and stada files

Step 1:
File Conversion

- Want to convert everything into text files
- Obscenely parallel problem:
  - For every PDF do X
- But with some gotchas...
  - I can't just spawn 700 PDF->txt conversion programs
    - That would grind my machine to a halt
  - And different invocations take a different amount of time
    - So I can't just spawn off tasks at a reasonable interval
- Two approaches
  - Dynamically tune based on load...
  - Or just say "F-it, and keep X jobs live"

Keeping X jobs live: Fork/join with a limiter

- Used a simple golang hack
- `capacity := make(chan bool, 10)`
  `done := make(chan bool)`
-
```
func run(txt string) {
    c <- True;
    ....
    <- c;
    done <- True;
}
```
- main just calls "`go func()`" on every line of stdin...
  and then an equal number of lines of "`<- done`"

A bit of tuning...

- ~10 jobs pegs all CPU cores on the OCR pipeline...
  - And it took a couple hours to PDF->txt the lot:
    Driver used multiple threads for single documents
- The .xlsx and stata conversion was a lot faster...
  - Set to ~25 (since python doesn't thread, especially on this task)
  - Took ~10 minutes or so
- Not fully efficient...
  - At the end of the PDF run there was no longer pegged CPU
  - 100% CPU utilization means efficiency loss due to context switching:
    Optimum would be ~95%
  - Also, really stressing the Windows virtualization...
    - I do all my work in "linux" under WSL

And Now To Search

- Just use the same pipeline with grep...
- But...
  - Ends up ***not*** pegging the CPUs...
    Instead I'm pegging the "disk"!
- OK for just searching for me, but...
  - Want to be able to do a "for anyone who wants" service
- So to do this, parallelize on an alternate axis:
  - Don't check one person at a time, check ***all*** people using a single program
  - And then invoke that in parallel across all files
- Gotcha problem: Need to make sure to synchronize writes well
  - Again, golang FTW:
    A channel for each user's results, the search does an atomic write to the channel

Results So Far...

- Lots of tabular data on everyone
  - For students it isn't just social security # but address, phone, self reported sexual orientation, self reported disability, etc...
- Fortunately no bulk banking or tax records
  - Did see a large number of individual records and documents concerning failed bank transactions
- Tabular data remarkably well organized
- Seeking to create a "Know Your Data" service
  - Request with a google form
  - Reply is a google doc
    - Allows UC/Google to handle all the access control
  - Need UC permission to do this, but I'm Trying!

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
- 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...

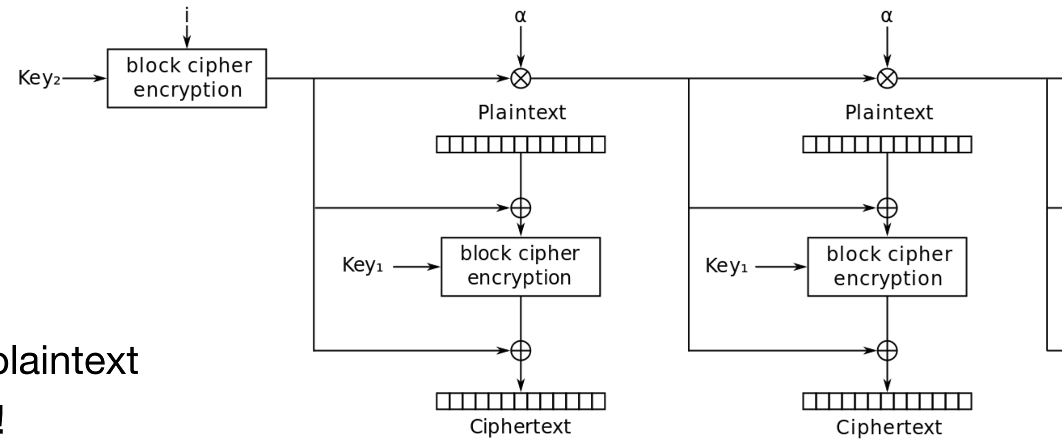
- The iPhone is perhaps the most secure commodity device available...
  - Not only does it receive patches but since the 5S it gained a dedicated cryptographic coprocessor
- The ***Secure Enclave Processor*** is the trusted base for the phone
  - Even the main operating system isn't fully trusted by 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 ***confidentality-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
- 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



XEX mode encryption

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!

Putting Everything Together In the Real World: The "Sad DNS" Attack...

- Over a decade after the Kaminski attacks, DNS cache poisoning returned to the news
- Reminder: Kaminski strategy...
  - You send glue records to actually poison the target:
    So to poison **`www.google.com`**, you create a query for **`a.google.com`**...
    And in the additional include **`www.google.com A 66.66.66.66`**
  - Still have to guess TXID (2^{16} work factor), but can keep trying!
- Defense was randomize the UDP source port as well...
  - So attacker has to guess the port and TXID at the same time (so 2^{32} work give-or-take)

Work by Keyu Man et al..
saddns.net (UC Riverside & Tsinghua University)

- Observation #1, can we detect what UDP port(s) are in use for a particular query?
  - If so, it turns the problem from expected 2^{32} work to $2^{16} + 2^{16}$ work!
  - You search for the open port, and if you get lucky, do the random TXIDs...
- Observation #2, can we cause the DNS authority for a domain to ***not respond?***
  - If so, enables us to have a lot more time for an attack
  - Which can make it far easier to be successful
- Answer to both is ***yes!***

Answer to 1:
Just Ask the DNS Resolver!

- By default you get a response if there is no open UDP port
  - "ICMP port unreachable"
- And UDP ports are not host specific by default...
  - So if you call **`sendto()`** and then **`recvfrom()`**...
    you won't send an ICMP back for that port
  - Behavior is not the same for **`connect()`** semantics:
    Connect will only not send back an ICMP if the UDP packet is from the remote IP
- So just scan all 2^{16} ports to see if you get a response!
- But there are gotchas...
  - ICMP packet sending has both a per-IP and global rate limit

So Enter Side Channels....

- Spoof a bunch of packets that will ***just*** trigger the global rate limit
  - Then send a packet from your IP to a port you ***know*** will trigger an ICMP response
- If you get a response...
  - One of the ports you checked was open!
  - So divide and conquer
- If you don't... Wait the short 20ms timeout and go onto the next block of ports to check
- Oh, and if they use **`connect()`** for UDP...
  - You only don't get an ICMP back if you are the IP that was connected to...
    So just spoof the real server with the side channel check!

And Now To Buy Time...
Another Rate Limit...

- DNS servers can be used for reflected DOS attacks
  - Spoof the IP address of the target and send a packet to the DNS server
  - DNS server then replies...
    Making the attack look like its coming from the DNS server
  - And since DNS replies are bigger, this is an amplifier for DOS attacks
- So DNS authority servers have their own rate limit
  - Too many requests from a single IP and they will start ignoring some request
- So use ***that*** to buy time...
  - Send just enough requests spoofing the target resolver's IP address for nonsense requests
  - Target resolver ignores the replies (after all, they were never made)
  - But the DNS authority server will now ignore the target resolver's DNS request!

Solution #1: DNSSEC

- If the resolver (or better yet client) validates DNSSEC...
  - Now it doesn't matter!
- Fortunately DNSSEC serving is getting easier
  - Most people are using a few outsourced DNS services
  - So they can easily add in DNSSEC if they aren't already
  - ***Any*** managed DNS service should use DNSSEC these days

Solution #2: Detection & Response

- Still relies on Kaminsky-style glue records for poisoning
  - Otherwise you can only race once on failure until the record's TTL expires
- This is ***VERY NOISY***
  - Hundreds or thousands of non-matching responses
  - This is even noisier than standard Kaminsky:
    Lots of bogus replies from the real server to suppress the legitimate reply
- So detect and respond
  - Don't query once, query multiple times and accept majority
  - Don't promote glue into the cache
  - Or just don't resolve the targeted name(s)
  - Nobody does this however