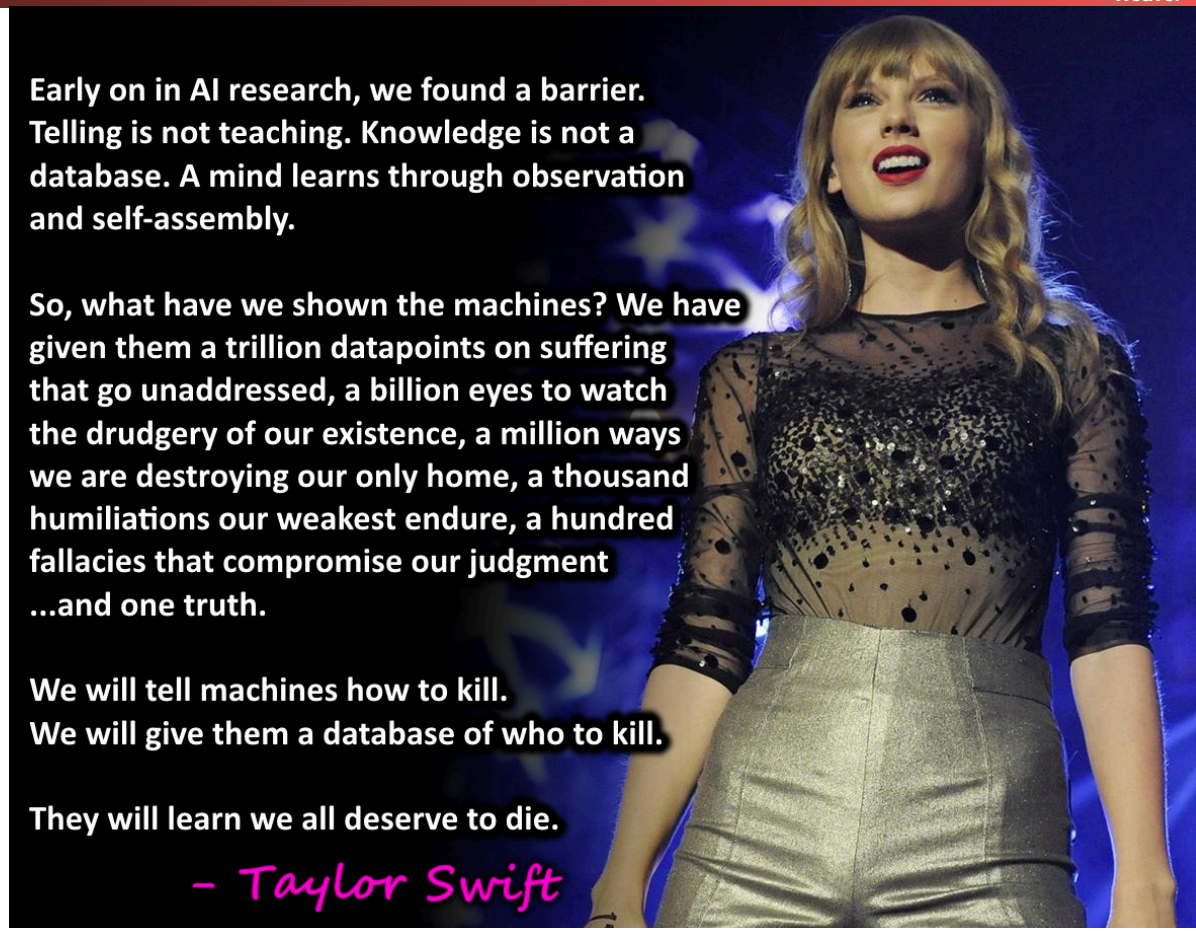
Nuke & Malcode

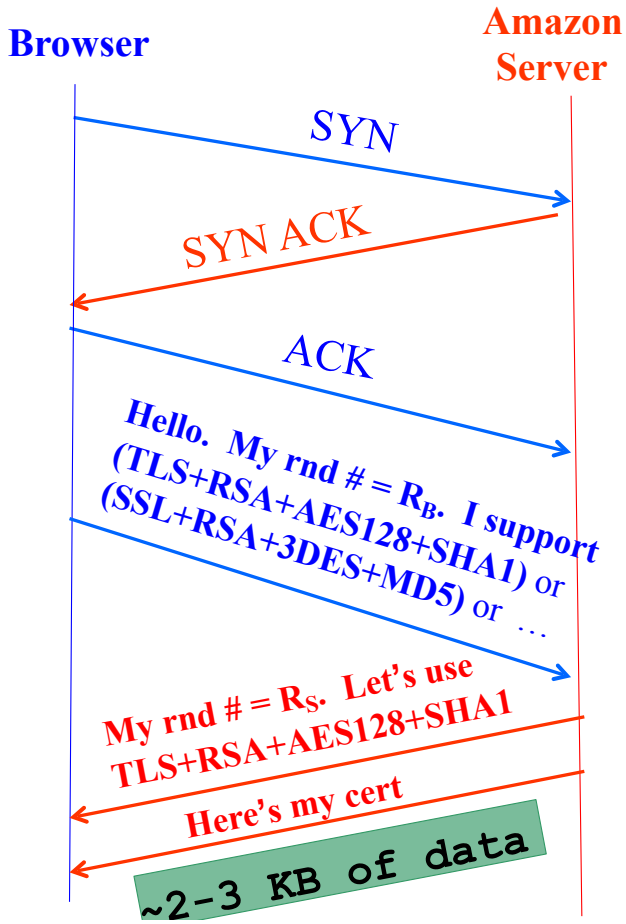


Exam Problem: TLS Fuckups...

- First four were a particularly hard and subtle problem...
  - Designed to test that you ***understood*** the TLS handshake
- I ***did*** warn people in lecture that drawing the handshakes for TLS would be a good thing for your cheat sheet...
- Basic framework: Either the client or the server has a bad pRNG
  - With a public disclosure of pRNG state, the attacker can predict all other values from that pRNG
  - A very very common problem: both fuckups and sabotage

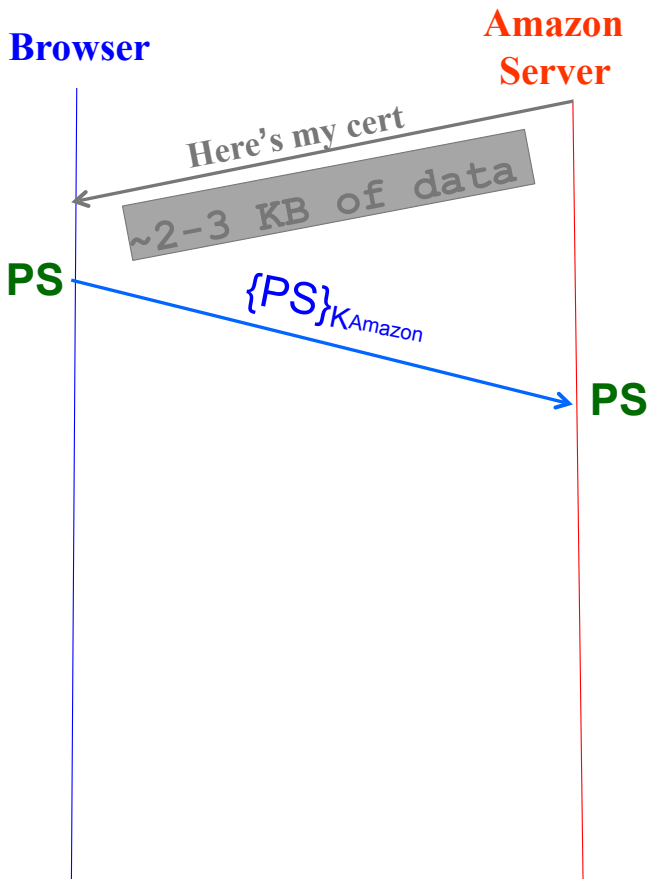
Reminder: The common part

- Browser (client) connects via TCP to Amazon's HTTPS server
- Client picks 256-bit random number R_B, sends over list of crypto protocols it supports
- Server picks 256-bit random number R_S, selects protocols to use for this session
- Server sends over its certificate
  - (all of this is in the clear)
- Client now ***validates*** cert
- The actual keys are F(R_B, R_S, PS)



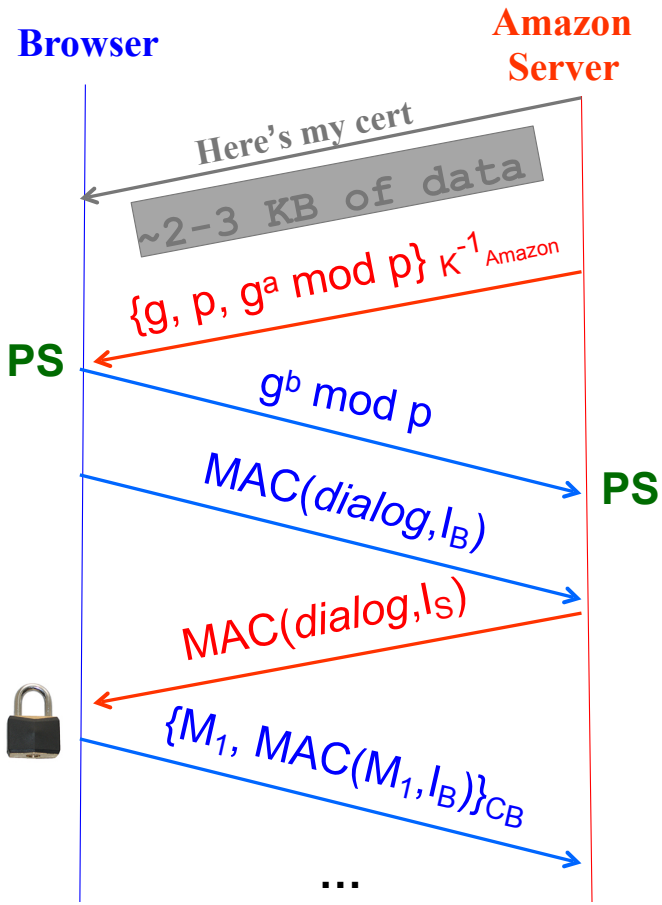
RSA...

- For RSA, browser constructs "Premaster Secret" PS
- Browser sends PS encrypted using Amazon's public RSA key K_{Amazon}
- Under the "fuckup" scenarios...
  - Attacker ***can*** take R_b and get PS if the browser uses the bad pRNG
  - Attacker ***can not*** get PS if the server uses the bad pRNG



Diffie-Hellman

- For DHE, PS is $g^{ab} \bmod p$
  - And the signature is to prevent the active MitM attack on DH and to validate the server
- So the attacker needs to know ***a*** or ***b***
  - For both, the associated R_s or R_b is in the clear
  - So for ***both***, the attacker can determine the necessary secret value
- This made the NSA pRNG sabotage with Dual_EC particularly interesting...
  - If you used the "less secure" (no forward secret) RSA mode, server corruption didn't matter
  - If you used the forward secret mode, it did!



And the "WPA-Enterprise Cert Problem"

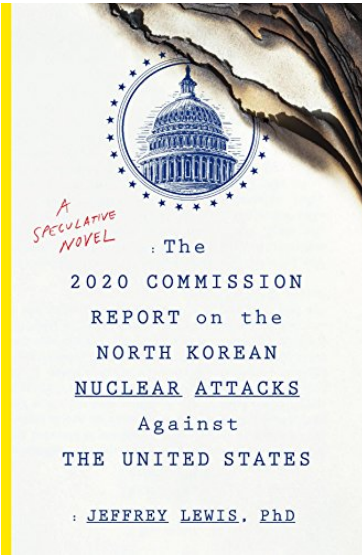
- Again, a subtle problem to get if you ***understand*** certificates
- For iPhone: You have "trust on first use"
  - So first time you can be MitM'ed, but if the first time is good, cool!
- For Android instructions: You have "Uh, just don't check"
  - So you can always be MitM'ed! 
- In the changes that Outis did...
  - His Linux laptop now trusts ***all*** CAs.
  - His Android phone does not...
    - So more people can potentially act as a MitM on his laptop

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...
  - "Do you want your system to fail safe or fail deadly?"
- 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 since Donald Trump doesn'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 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

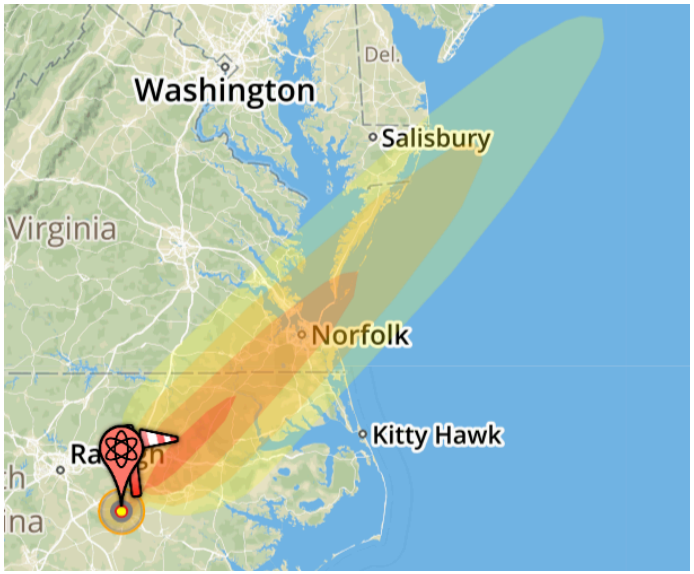


The Interesting Problem: Limiting Use

- Who might use a nuke without authorization?
  - Our "allies" where we station our nukes
    - Original motivation: Nukes stored in Turkey and Greece
  - 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)

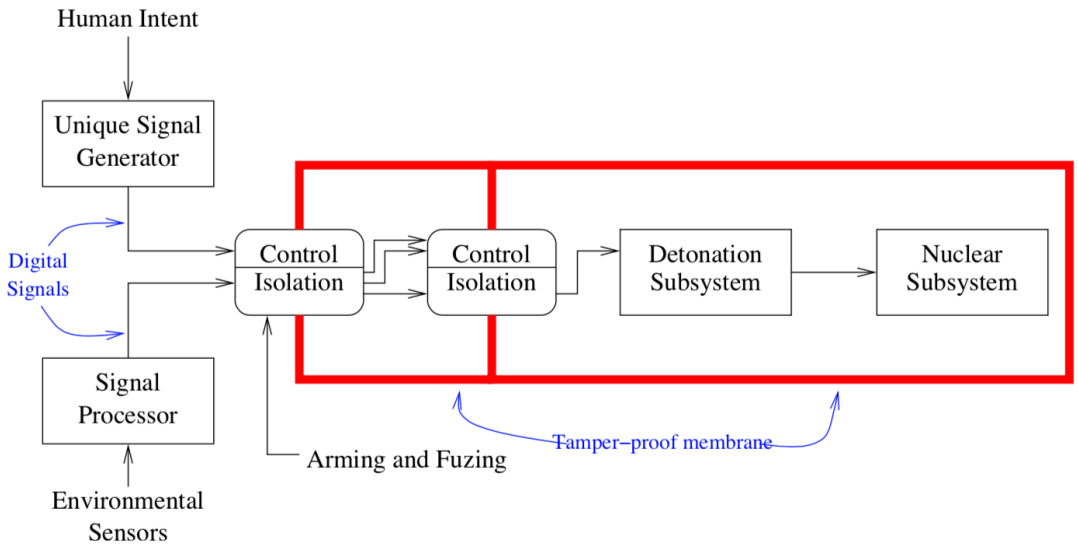
Nuke Safety Features

- 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
- 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:
      In fact, the name alone suggests how: You block a link unless you have permission!
- 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.

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

- 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?

Malware:
Catch-All Term for "Malicious Code"

- Attacker code running on victim computer(s)

What Can Malware Do?

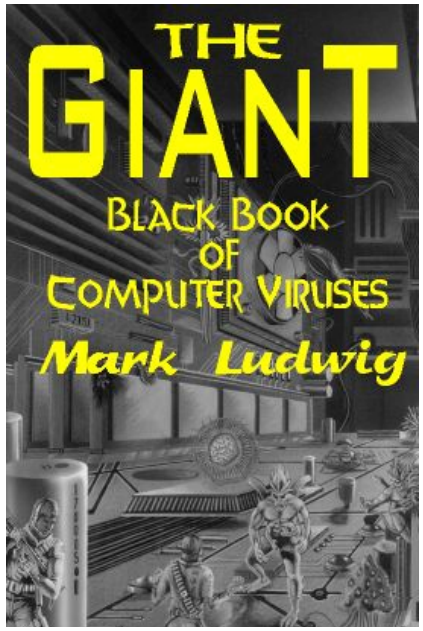
- Pretty much anything
  - Payload generally decoupled from how manages to run
  - Only subject to permissions under which it runs
- Examples:
  - Brag or exhort or extort (pop up a message/display)
  - Trash files (just to be nasty)
  - Damage hardware (!)
  - Launch external activity (spam, click fraud, DoS; banking)
  - Steal information (exfiltrate)
  - Keylogging; screen / audio / camera capture
  - Encrypt files (ransomware)
- Possibly delayed until condition occurs
  - “time bomb” / “logic bomb”

Malware That Automatically Propagates

- ***Virus*** = code that propagates (replicates) across systems by arranging to have itself eventually executed, creating a new additional instance
  - Generally infects by altering stored code
- ***Worm*** = code that self-propagates/replicates across systems by arranging to have itself immediately executed (creating new addl. instance)
  - Generally infects by altering running code
  - No user intervention required
- (Note: line between these isn't always so crisp; plus some malware incorporates both approaches)
- ***NO EXPERIMENTATION WITH SELF REPLICATING CODE!***

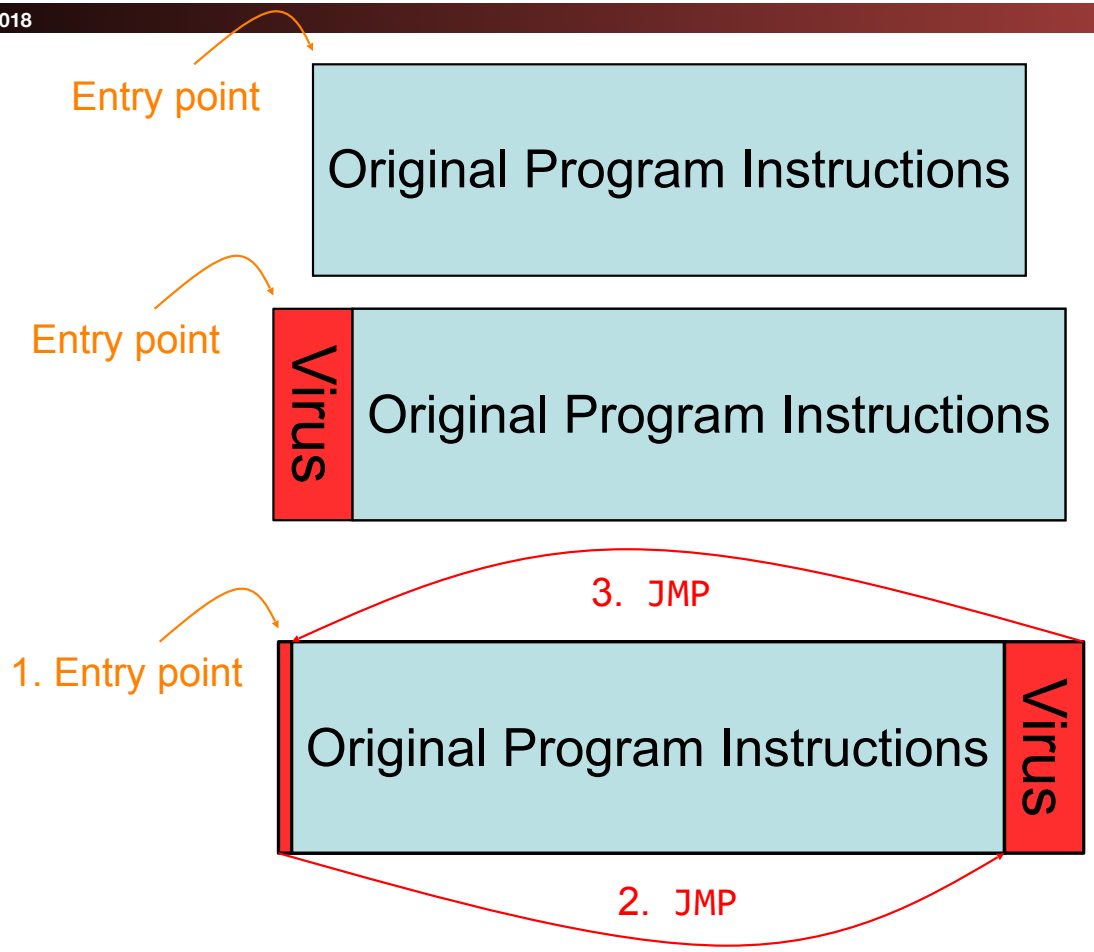
The Problem of Viruses

- Opportunistic = code will eventually execute
  - Generally due to user action
    - Running an app, booting their system, opening an attachment
- Separate notions: how it propagates vs. what else it does when executed (payload)
- General infection strategy: find some code lying around, alter it to include the virus
- Have been around for decades …
  - … resulting arms race has heavily influenced evolution of modern malware



Propagation

- When virus runs, it looks for an opportunity to infect additional systems
- One approach: look for USB-attached thumb drive, alter any executables it holds to include the virus
  - Strategy: when drive later attached to another system & altered executable runs, it locates and infects executables on new system's hard drive
- Or: when user sends email w/ attachment, virus alters attachment to add a copy of itself
  - Works for attachment types that include programmability
  - E.g., Word documents (macros)
  - Virus can also send out such email proactively, using user's address book + enticing subject ("I Love You")



Original program instructions can be:

- Application the user runs
- Run-time library / routines resident in memory
- Disk blocks used to boot OS
- Autorun file on USB device
- …

Other variants are possible; whatever manages to get the virus code executed

Detecting Viruses

- Signature-based detection
  - Look for bytes corresponding to injected virus code
  - High utility due to replicating nature
    - If you capture a virus V on one system, by its nature the virus will be trying to infect many other systems
    - Can protect those other systems by installing recognizer for V
- Drove development of multi-billion $$ AV industry (AV = “antivirus”)
  - So many endemic viruses that detecting well-known ones becomes a “checklist item” for security audits
- Using signature-based detection also has de facto utility for (glib) marketing
  - Companies compete on number of signatures …
    - … rather than their quality (harder for customer to assess)

virustotal

SHA256: 58860062c9844377987d22826eb17d9130dceaa7f0fa68ec9d44dfa435d6ded4

File name: cc8caa3d2996bf0360981781869f0c82.exe

Detection ratio: 11 / 62

Analysis date: 2017-04-18 22:28:27 UTC (56 minutes ago)

Analysis | File detail | Relationships | Additional information | Comments 4 | Votes | Behavioural information

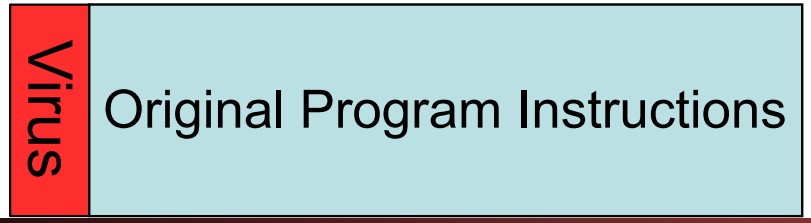
Antivirus	Result	Update
Avira (no cloud)	TR/Crypt.ZPACK.atbin	20170418
CrowdStrike Falcon (ML)	malicious_confidence_100% (W)	20170130
DrWeb	Trojan.PWS.Panda.11620	20170418
Endgame	malicious (moderate confidence)	20170413
ESET-NOD32	a variant of Win32/GenKryptik.ACKE	20170418
Invincea	virus.win32.ramnit.ah	20170413
Kaspersky	Trojan.Win32.Yakes.tavs	20170418
Palo Alto Networks (Known Signatures)	generic.ml	20170418

Virus Writer / AV Arms Race

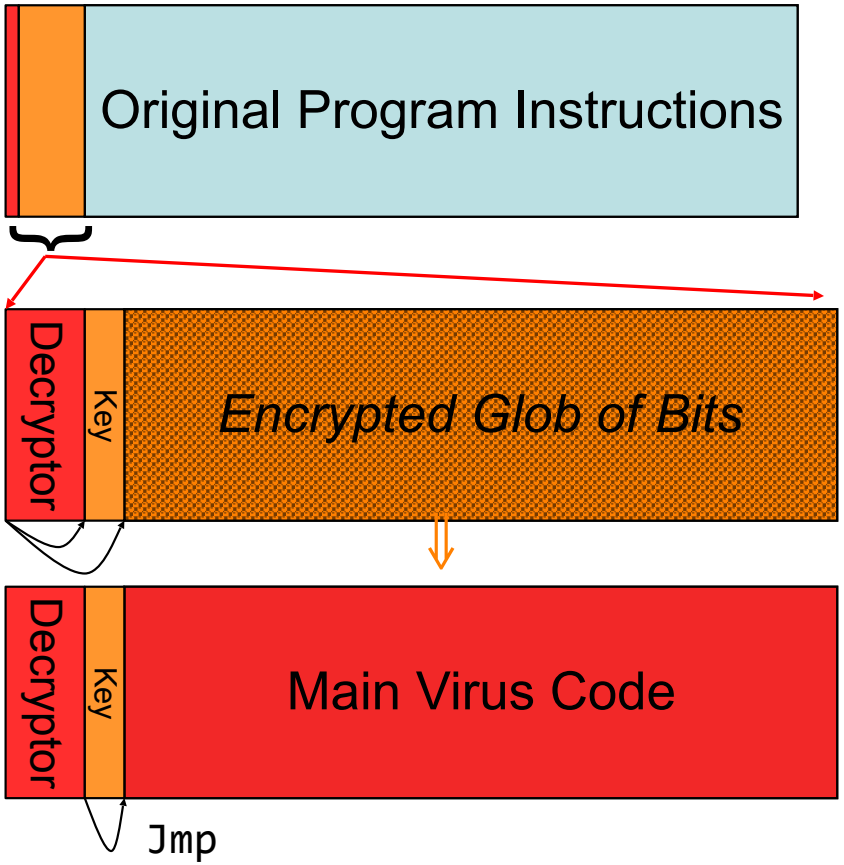
- If you are a virus writer and your beautiful new creations don't get very far because each time you write one, the AV companies quickly push out a signature for it ….
  - …. What are you going to do?
- Need to keep changing your viruses …
  - … or at least changing their appearance!
- How can you mechanize the creation of new instances of your viruses …
  - … so that whenever your virus propagates, what it injects as a copy of itself looks different?

Polymorphic Code

- We've already seen technology for creating a representation of data apparently completely unrelated to the original: encryption!
- Idea: every time your virus propagates, it inserts a ***newly encrypted*** copy of itself
  - Clearly, encryption needs to vary
    - Either by using a different key each time
    - Or by including some random initial padding (like an IV)
  - Note: weak (but simple/fast) crypto algorithm works fine
    - No need for truly strong encryption, just obfuscation
- When injected code runs, it decrypts itself to obtain the original functionality



Instead of this …

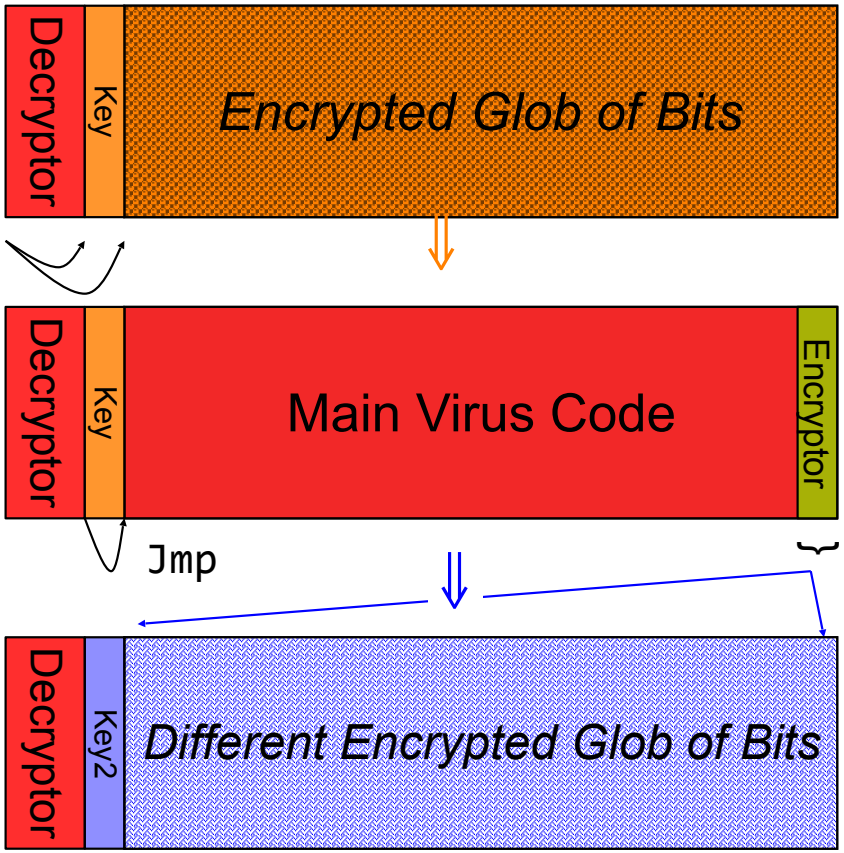


Virus has *this* initial structure

When executed, decryptor applies key to decrypt the glob …

… and jumps to the decrypted code once stored in memory

Polymorphic Propagation



Once running, virus uses an *encryptor* with a new key to propagate

New virus instance bears little resemblance to original

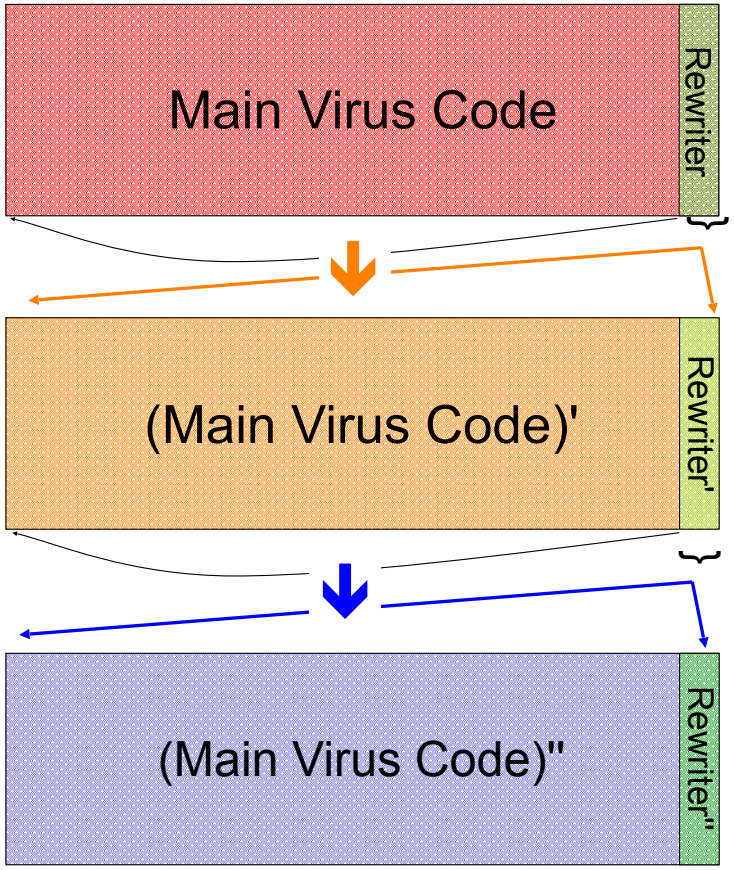
Arms Race: Polymorphic Code

- Given polymorphism, how might we then detect viruses?
- Idea #1: use narrow sig. that targets ***decryptor***
  - Issues?
    - Less code to match against ⇒ more false positives
    - Virus writer spreads decryptor across existing code
- Idea #2: execute (or statically analyze) suspect code to see if it decrypts!
  - Issues?
    - Legitimate "packers" perform similar operations (decompression)
    - How long do you let the new code execute?
      - If decryptor only acts after lengthy legit execution, difficult to spot
- Virus-writer countermeasures?

Metamorphic Code

- Idea: every time the virus propagates, generate semantically different version of it!
  - Different semantics only at immediate level of execution; higher-level semantics remain same
- How could you do this?
- Include with the virus a code rewriter:
  - Inspects its own code, generates random variant, e.g.:
    - Renumber registers
    - Change order of conditional code
    - Reorder operations not dependent on one another
    - Replace one low-level algorithm with another
    - Remove some do-nothing padding and replace with different do-nothing padding ("chaff")
      - Can be very complex, legit code … if it's never called!

Metamorphic Propagation



When ready to propagate, virus invokes a randomized *rewriter* to construct new but semantically equivalent code (*including the rewriter*)

Detecting Metamorphic Viruses?

- Need to analyze execution behavior
  - Shift from syntax (appearance of instructions) to semantics (effect of instructions)
- Two stages: (1) AV company analyzes new virus to find behavioral signature; (2) AV software on end systems analyze suspect code to test for match to signature
- What countermeasures will the virus writer take?
  - Delay analysis by taking a long time to manifest behavior
    - Long time = await particular condition, or even simply clock time
  - Detect that execution occurs in an analyzed environment and if so behave differently
    - E.g., test whether running inside a debugger, or in a Virtual Machine
- Counter-countermeasure?
  - AV analysis looks for these tactics and skips over them
- Note: attacker has edge as AV products supply an oracle

Malcode Wars and the Halting Problem...

- Cyberwars are not won by solving the halting problem...
  Cyberwars are won by making some other poor sod solve the halting problem!!!
  - In the limit, it is ***undecidable*** to know "is this code bad?"
- Modern focus is instead "is this code ***new?***"
  - Use a secure cryptographic hash (so sha-256 not md5)
  - Check hash with central repository: If ***not*** seen before, treat binary as inherently more suspicious
- Creates a bind for attackers:
  - Don't make your code *morphic:
    Known bad signature detectors find it
  - Make your code *morphic:
    It always appears as new and therefore inherently suspicious

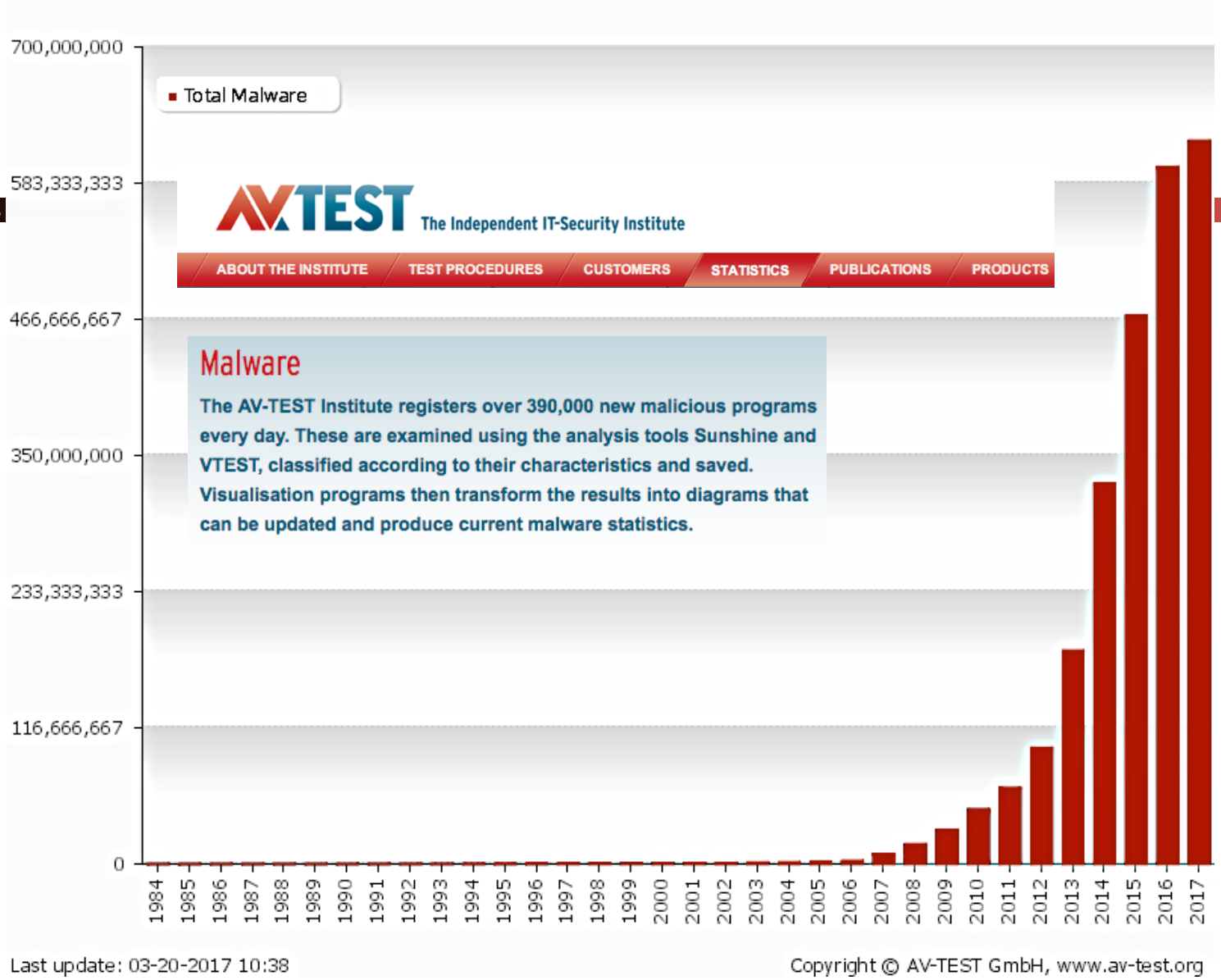
Creating binds is very powerful...

- You have a detector D for some bad behavior...
  - So bad-guys come up with a way of avoiding detector D
- So come up with a detection strategy for ***avoiding detector D***
  - So to avoid ***this*** detector, the attacker ***must not*** try to avoid D
- When you can do it, it is very powerful!

How Much Malware Is Out There?

- A final consideration re polymorphism and metamorphism:
  - Presence can lead to mis-counting a single virus outbreak as instead reflecting 1,000s of seemingly different viruses

- Thus take care in interpreting vendor statistics on malcode varieties
  - (Also note: public perception that huge malware populations exist is in the vendors' own interest)


700,000,000
583,333,333
466,666,667
350,000,000
233,333,333
116,666,667
0
Total Malware
AV-TEST The Independent IT-Security Institute
ABOUT THE INSTITUTE
TEST PROCEDURES
CUSTOMERS
STATISTICS
PUBLICATIONS
PRODUCTS
Malware
The AV-TEST Institute registers over 390,000 new malicious programs every day. These are examined using the analysis tools Sunshine and VTEST, classified according to their characteristics and saved. Visualisation programs then transform the results into diagrams that can be updated and produce current malware statistics.
1984 1985 1986 1987 1988 1989 1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017
Last update: 03-20-2017 10:38
Copyright © AV-TEST GmbH, www.av-test.org

Infection Cleanup

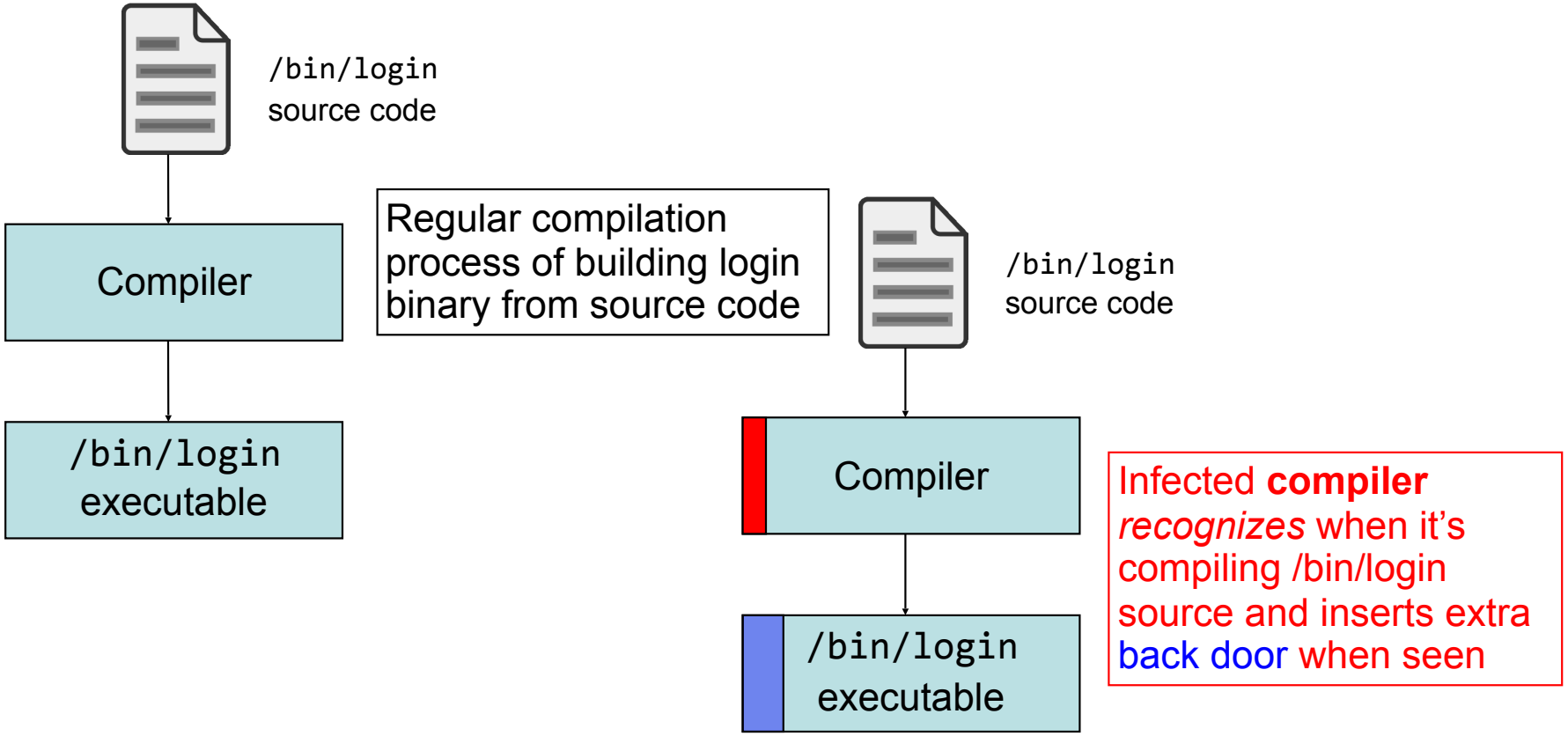
- Once malware detected on a system, how do we get rid of it?
- May require restoring/repairing many files
  - This is part of what AV companies sell: per-specimen disinfection procedures
- What about if malware executed with adminstrator privileges?
  - "Game over man, Game Over!"
  - “Dust off and nuke the entire site from orbit. It’s the only way to be sure”- ALIENS
  - i.e., rebuild system from original media + data backups
- Malware may include a rootkit: kernel patches to hide its presence (its existence on disk, processes)

Infection Cleanup

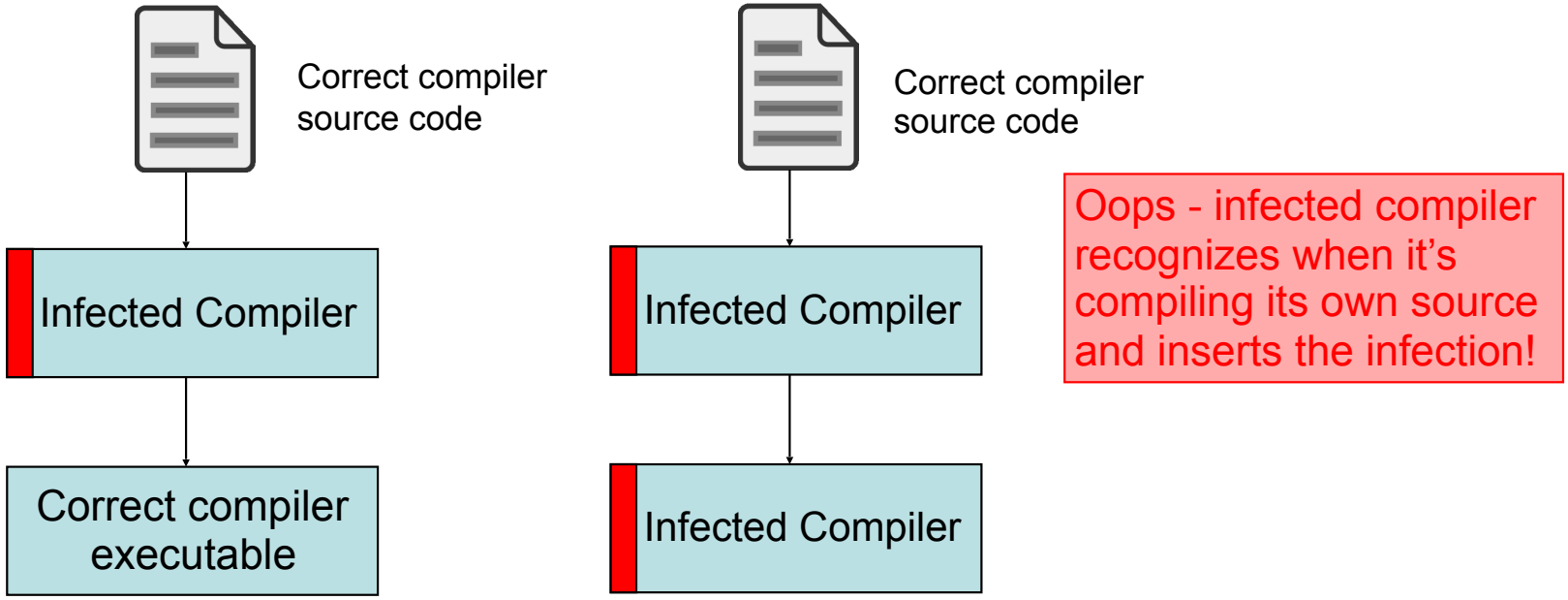
- Once malware detected on a system, how do we get rid of it?
- May require restoring/repairing many files
  - This is part of what AV companies sell: per-specimen disinfection procedures
- What about if malware executed with adminstrator privileges?
  - "Game over man, Game Over!"
  - “Dust off and nuke the entire site from orbit. It’s the only way to be sure”- ALIENS
  - i.e., rebuild system from original media + data backups
- Malware may include a rootkit: kernel patches to hide its presence (its existence on disk, processes)

Infection Cleanup, con't

- If we have complete source code for system, we could rebuild from that instead, couldn't we?
- No!
- Suppose forensic analysis shows that virus introduced a backdoor in /bin/login executable
  - (Note: this threat isn't specific to viruses; applies to any malware)
- Cleanup procedure: rebuild /bin/login from source …


/bin/login
source code
Compiler
/bin/login
executable
Regular compilation process of building login binary from source code
/bin/login
source code
Compiler
/bin/login
executable
Infected **compiler** *recognizes* when it's compiling /bin/login source and inserts extra back door when seen

No problem, first step, rebuild the compiler so it's uninfected



Oops - infected compiler recognizes when it's compiling its own source and inserts the infection!

No amount of careful source-code scrutiny can prevent this problem.
And if the *hardware* has a back door …

Reflections on Trusting Trust
Turing-Award Lecture, Ken Thompson, 1983

More On "Rootkits"

- If you control the operating system...
  - You can hide extremely well
- EG, your malcode is on disk...
  - So it will persist across reboots
- But if you try to ***read the disk***...
  - The operating system just says "Uhh, this doesn't exist!"

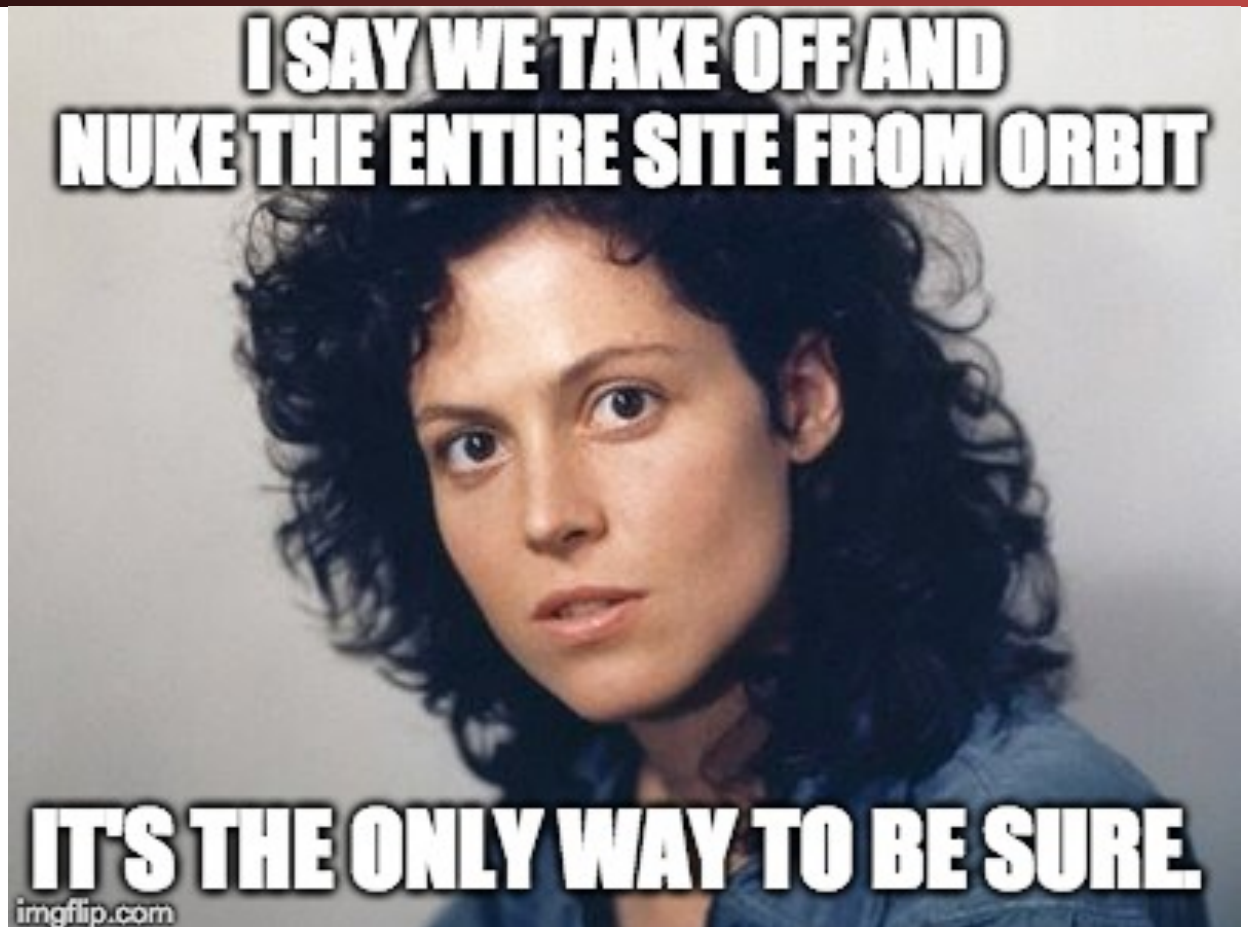
Even More Places To Hide!

- In the BIOS/EFI Firmware!
  - So you corrupt the BIOS which corrupts the OS...
  - Really hard to find:
    Defense, ***only*** run cryptographically signed BIOS code as part of the Trusted Base
- In the disk controller firmware!
  - So the master boot record, when read on boot up corrupts the OS...
  - But when you try to read the MBR later... It is just "normal"
  - Again, defense is ***signed code:*** The Firmware will only load a signed operating system
    - Make sure the disk itself is ***not trusted!***

Robust Rootkit Detection: Detect the act of hiding...

- Do an "in-system" scan of the disk...
  - Record it to a USB drive
- Reboot the system with trusted media
  - So a known good operating system
- Do the same scan!
  - If the scans are different, you found the rootkit!
- For windows, you can also do a "high/low scan" on the Registry:
  - Forces the bad guy to understand the registry as well as Mark Russinovich (the guy behind Sysinternals who's company Microsoft bought because he understood the Registry better than Microsoft's own employees!)
- Forces a bind on the attacker:
  - Hide and persist? You can be detected
  - Hide but don't persist? You can't survive reboots!

Which Means ***Proper*** Malcode Cleanup...



Forensics

- Vital complement to detecting attacks: figuring out what happened in wake of successful attack
- Doing so requires access to rich/extensive logs
  - Plus tools for analyzing/understanding them
- It also entails looking for patterns and understanding the implications of structure seen in activity
  - An iterative process ("peeling the onion")
- Consider these actual emails from operational security …

Emails omitted from on-line slides