Malware: Viruses

CS 161: Computer Security

Prof. Vern Paxson

TAs: Paul Bramsen, Apoorva Dornadula,
David Fifield, Mia Gil Epner, David Hahn, Warren He,
Grant Ho, Frank Li, Nathan Malkin, Mitar Milutinovic,
Rishabh Poddar, Rebecca Portnoff, Nate Wang

https://inst.eecs.berkeley.edu/~cs161/

April 20, 2017

Inside a Modern HIDS ("AV")

- URL/Web access blocking:
  - Prevent users from going to known bad locations
- Protocol scanning of network traffic (esp. HTTP)
  - Detect & block known attacks
  - Detect & block known malware communication
- Payload scanning
  - Detect & block known malware
- (Auto-update of signatures for these)
- Cloud queries regarding reputation
  - Who else has run this executable and with what results?
  - What's known about the remote host / domain / URL?

Inside a Modern HIDS, con't

- *Sandbox execution*
  - Run selected executables in constrained/monitored environment
  - Analyze:
    - System calls
    - Changes to files / registry
    - Self-modifying code (*polymorphism/metamorphism*)
- File scanning
  - Look for known malware that installs itself on disk
- Memory scanning
  - Look for known malware that never appears on disk
- Runtime analysis
  - Apply heuristics/signatures to execution behavior

Inside a Modern NIDS

- Deployment inside network as well as at border
  - Greater visibility, including tracking of user identity
- Full protocol analysis
  - Including extraction of complex embedded objects
  - In some systems, 100s of known protocols
- Signature analysis (also behavioral)
  - Known attacks/vulnerabilities, malware communication, blacklisted hosts/domains
  - Known malicious payloads
  - Sequences/patterns of activity
- *Shadow execution* (e.g., Flash, PDF programs)
- Extensive logging (in support of forensics)
- Auto-update of signatures, blacklists; cloud queries

Malware

The Problem of Malware

- Malware = malicious code that runs on a victim's system
- How does it manage to run?
  - Attacks a network-accessible vulnerable service
  - Vulnerable client connects to remote system that sends over an attack (a *driveby*)
  - *Social engineering:* trick user into running/installing
  - "Autorun" functionality (esp. from plugging in USB device)
  - Slipped into a system component (at manufacture; compromise of software provider; substituted via MITM)
  - Attacker with local access downloads/runs it directly
    - Might include using a local "privilege escalation" exploit

What Can Malware Do?

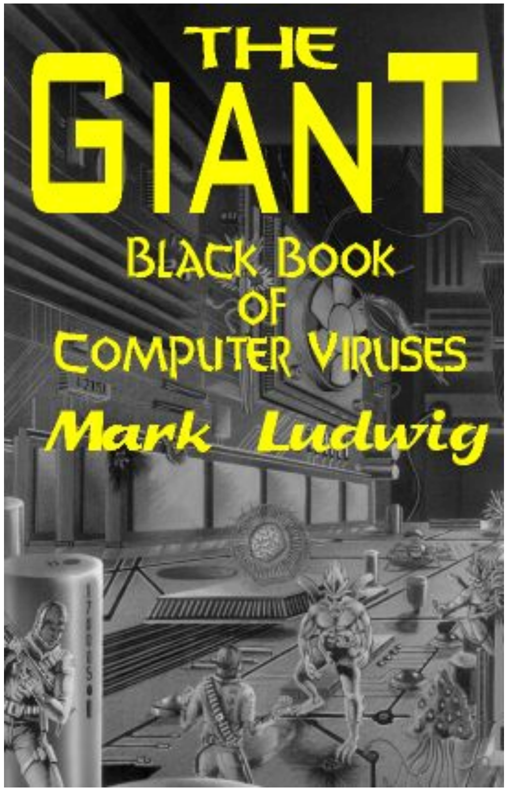
- Pretty much *anything*
  - Payload generally decoupled from how manages to run
  - Only subject to permissions under which it runs
- Examples:
  - Brag or exhort or extort (pop up a message/display)
  - Trash files (just to be nasty)
  - Damage hardware (!)
  - Launch external activity (spam, *click 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)

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



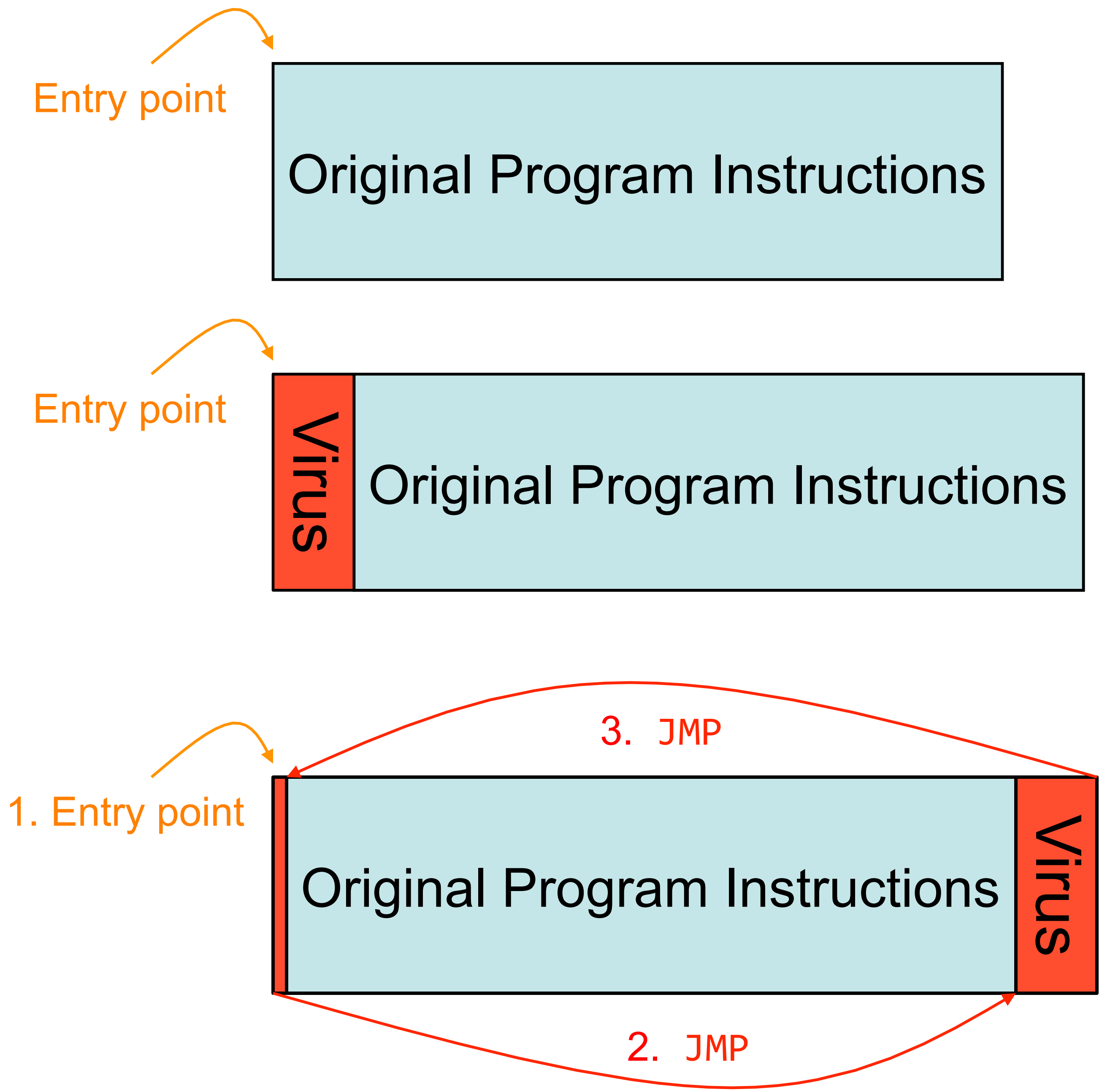
- Have been around for decades …
  - … resulting **arms race** has heavily influenced evolution of modern malware

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

__autorun__ is handy here!

- 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

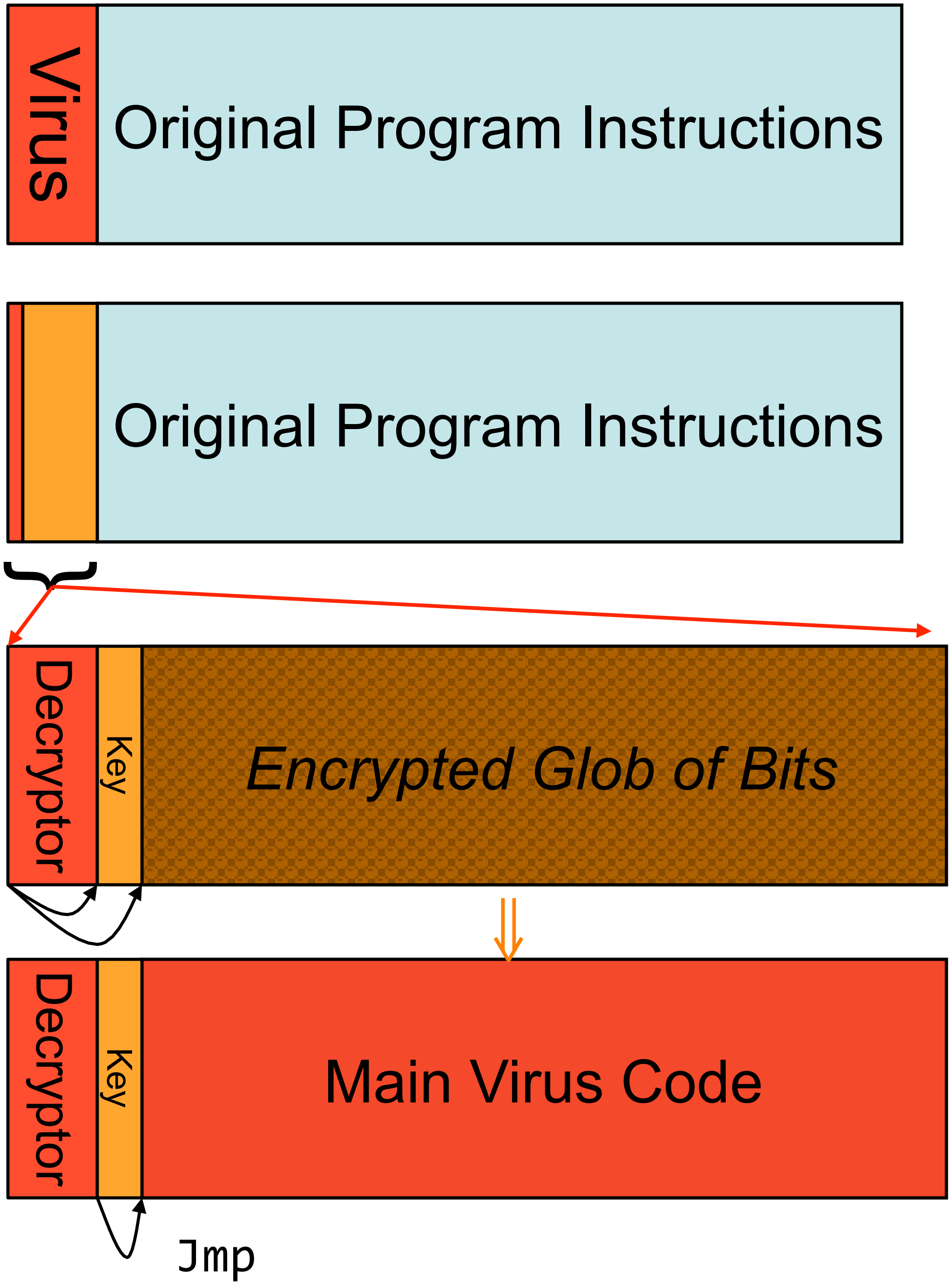
Antivirus	Result	Update
Avira (no cloud)	TR/Crypt.ZPACK.atbin	20170418
CrowdStrike Falcon (ML)	malicious_confidence_100% (W)	20170130
DrWeb	Trojan.PWS.Panda.11620	20170418
Endgame	malicious (moderate confidence)	20170413
ESET-NOD32	a variant of Win32/GenKryptik.ACKE	20170418
Invincea	virus.win32.ramnit.ah	20170413
Kaspersky	Trojan.Win32.Yakes.tavs	20170418
Palo Alto Networks (Known Signatures)	generic.ml	20170418
TrendMicro-HouseCall	Suspicious_GEN.F47V0418	20170418
Webroot	W32.Malware.Gen	20170418
ZoneAlarm by Check Point	Trojan.Win32.Yakes.tavs	20170418
Ad-Aware	✔	20170418
AegisLab	✔	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



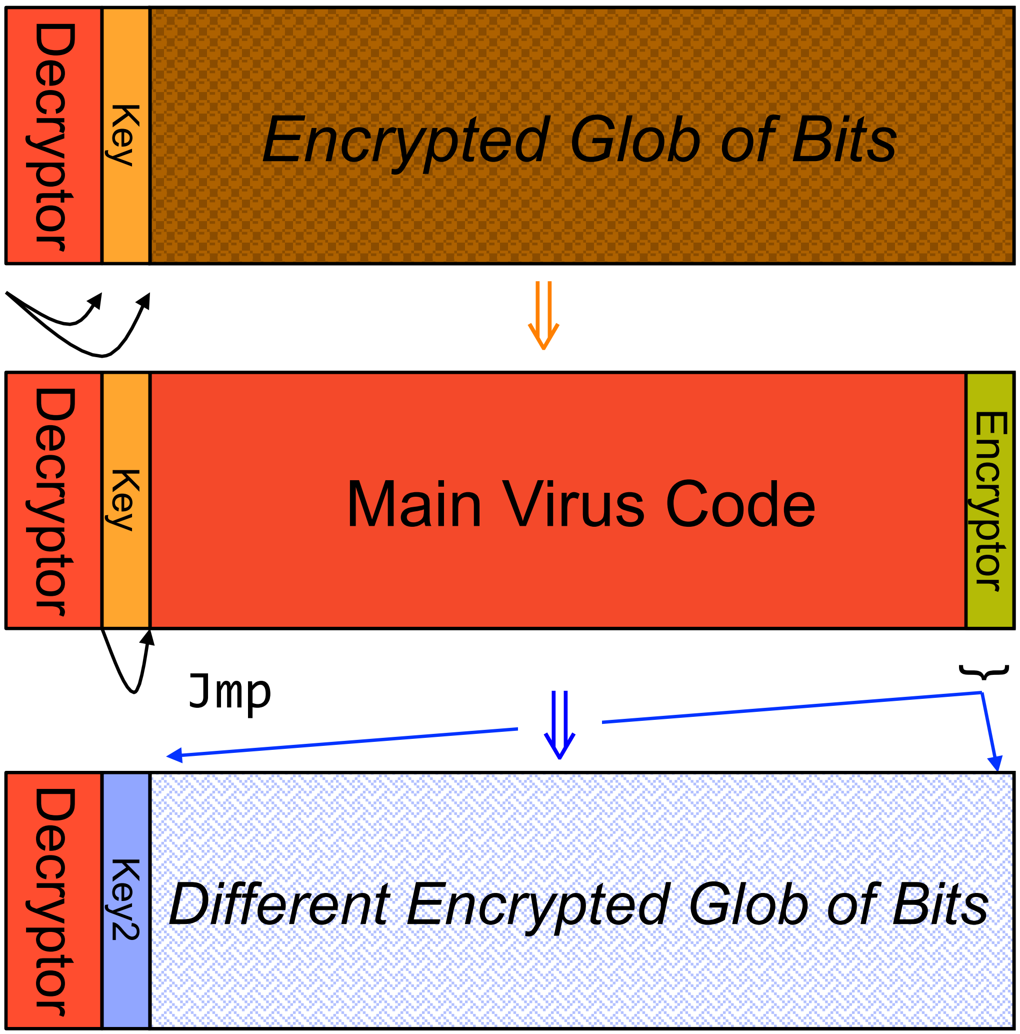
Instead of this …

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

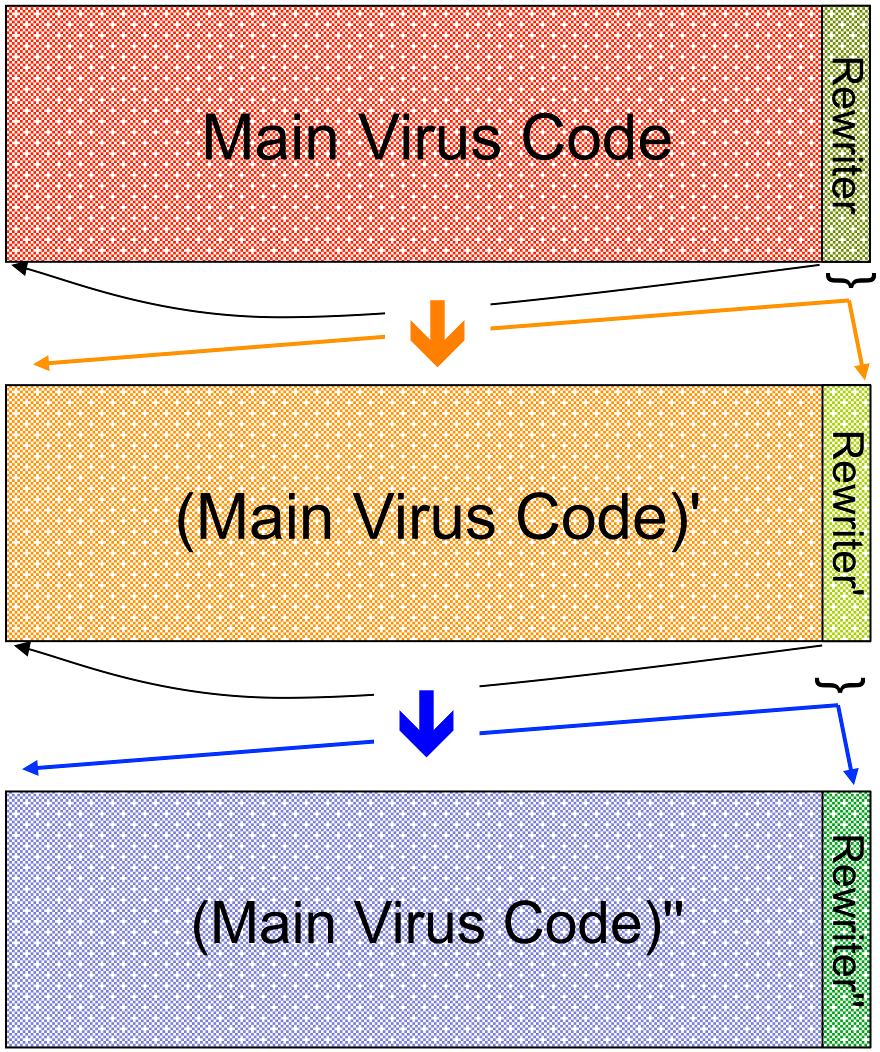
Arms Race: Polymorphic Code

- 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***

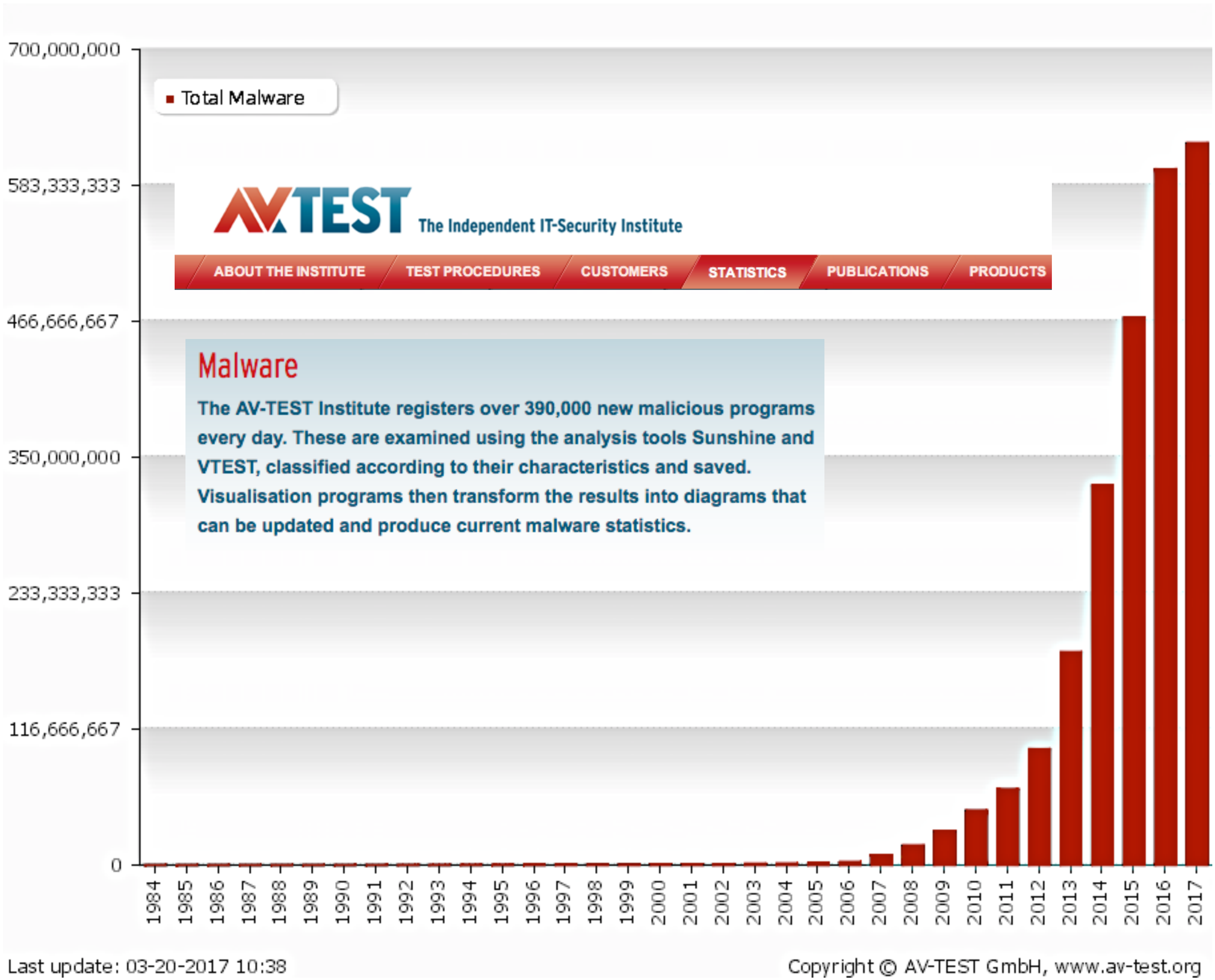
5 Minute Break

Questions Before We Proceed?

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?

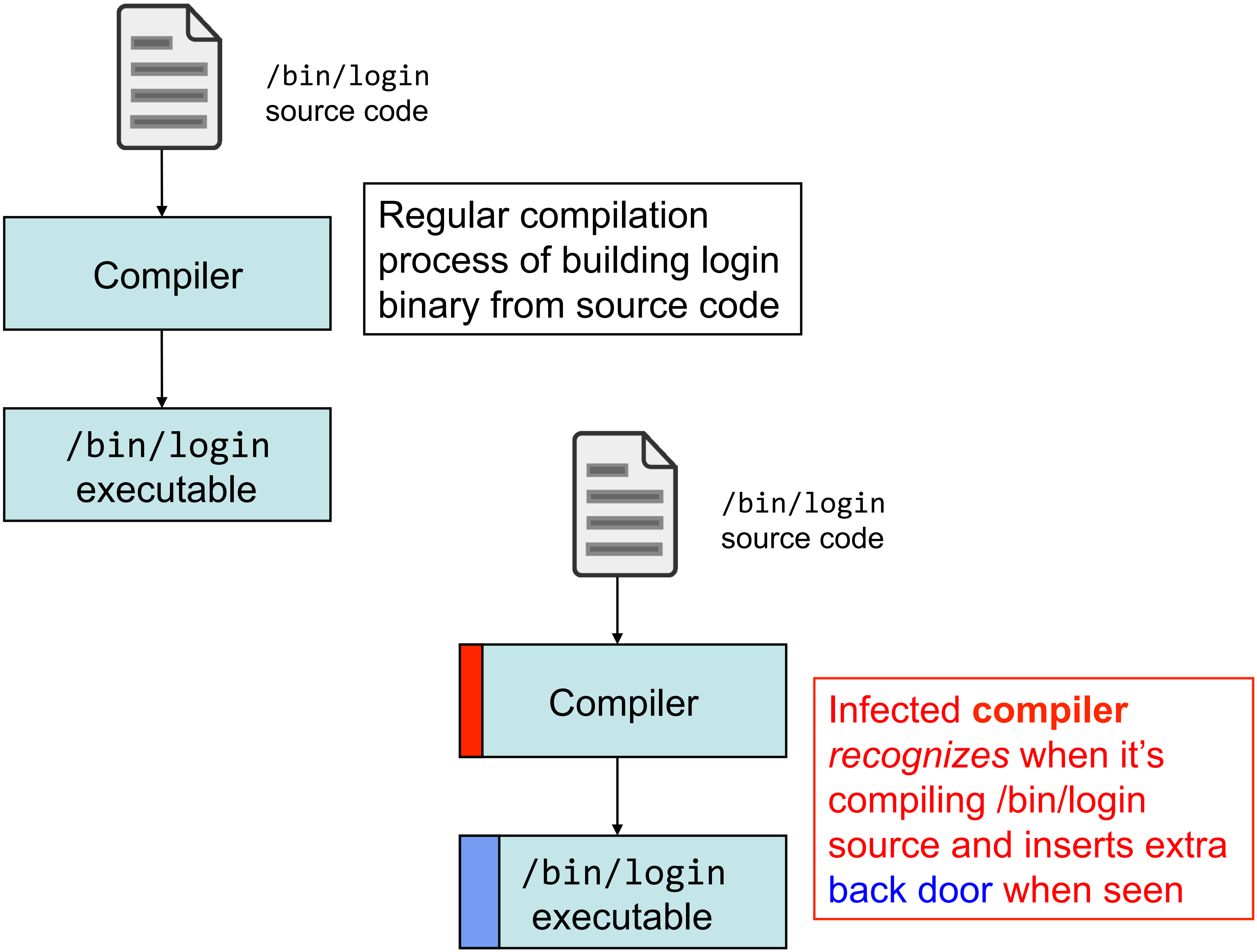
  "*nuke the entire site from orbit. It's the only way to be sure*"

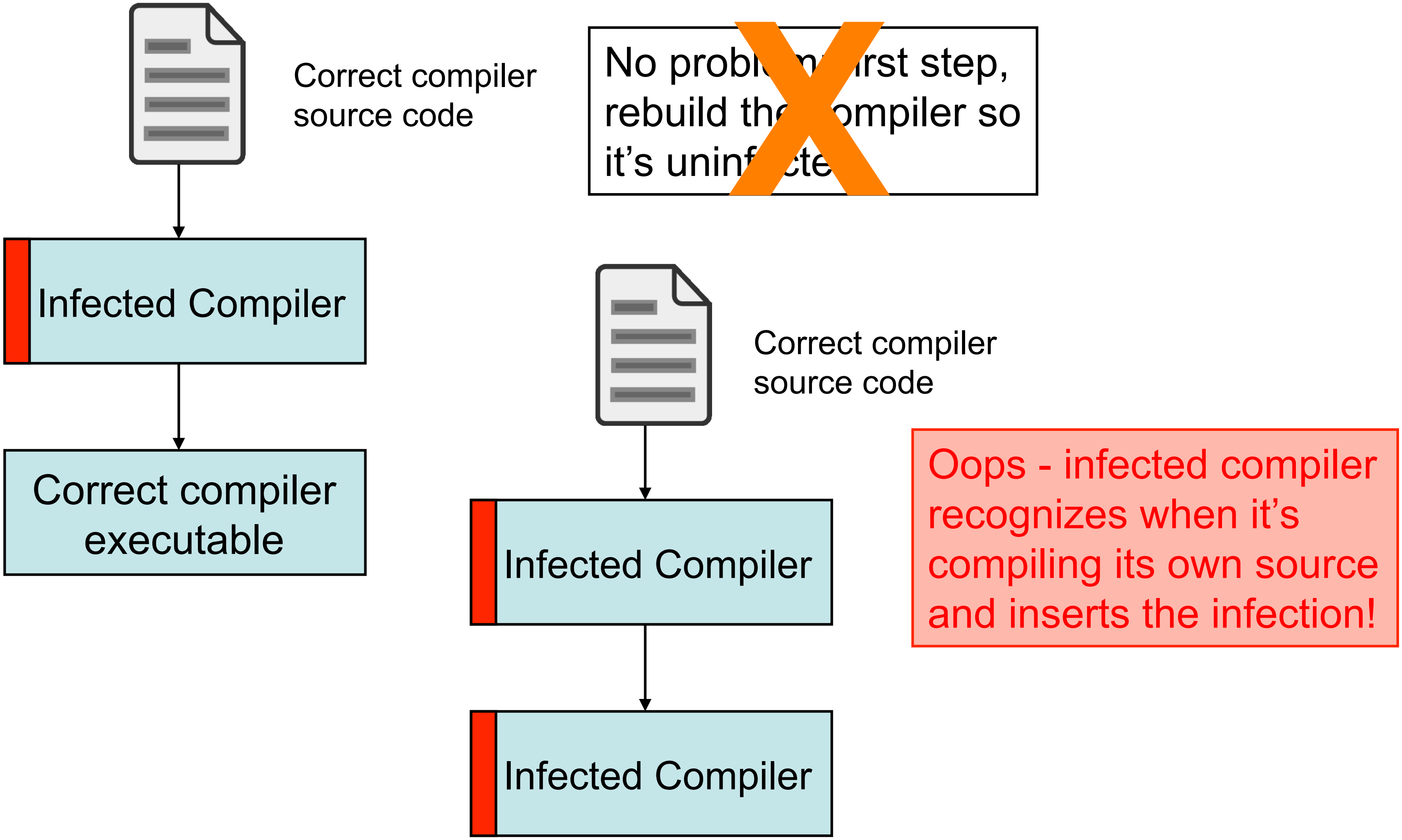
  \- Aliens

  - 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 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