Malware: Viruses

CS 161 - Computer Security

Profs. Vern Paxson & David Wagner

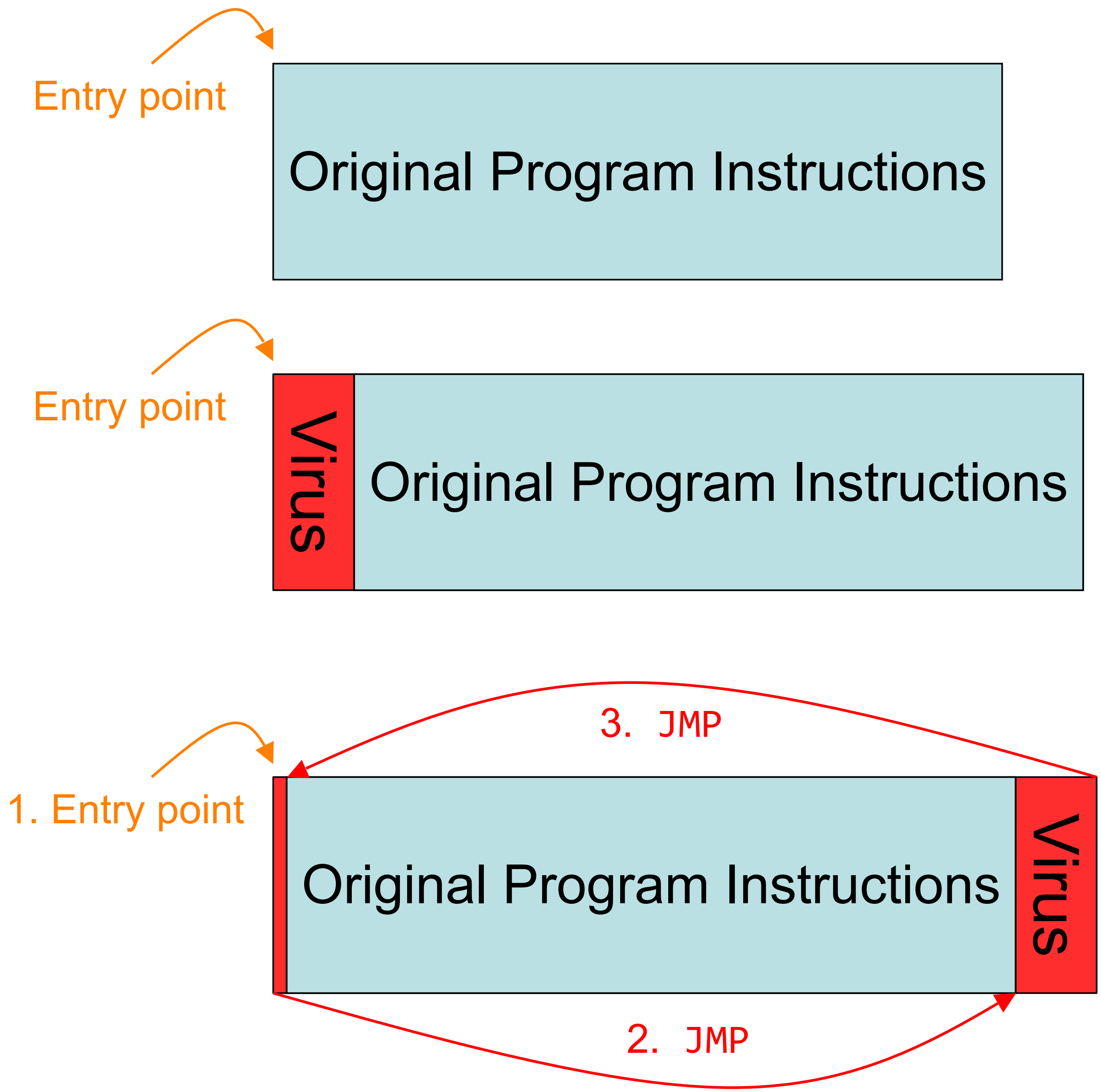
TAs: John Bethencourt, Erika Chin, Matthew Finifter, Cynthia Sturton, Joel Weinberger

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

April 12, 2010

The Problem of Viruses

- Virus = code that replicates
  - Instances opportunistically create new addl. instances
  - Goal of replication: install code on additional systems
- Opportunistic = code will eventually execute
  - Generally due to user action
    - Running an app, booting their system, opening an attachment
- Separate notions for a virus: 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



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
- …

Many variants are possible, and of course can combine techniques

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: if 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), PDFs (Javascript)
  - Virus can also send out such email proactively, using user's address book + enticing subject ("I Love You")

Payload

- Besides propagating, what else can the virus do when executing?
  - Pretty much *anything*
    - Payload is decoupled from propagation
    - Only subject to permissions under which it runs
- Examples:
  - Brag or exhort (pop up a message)
  - Trash files (just to be nasty)
  - Damage hardware (!)
  - Keylogging
  - Encrypt files
    - "Ransomware"
- Possibly delayed until condition occurs
  - "time bomb" / "logic bomb"

Detecting Viruses

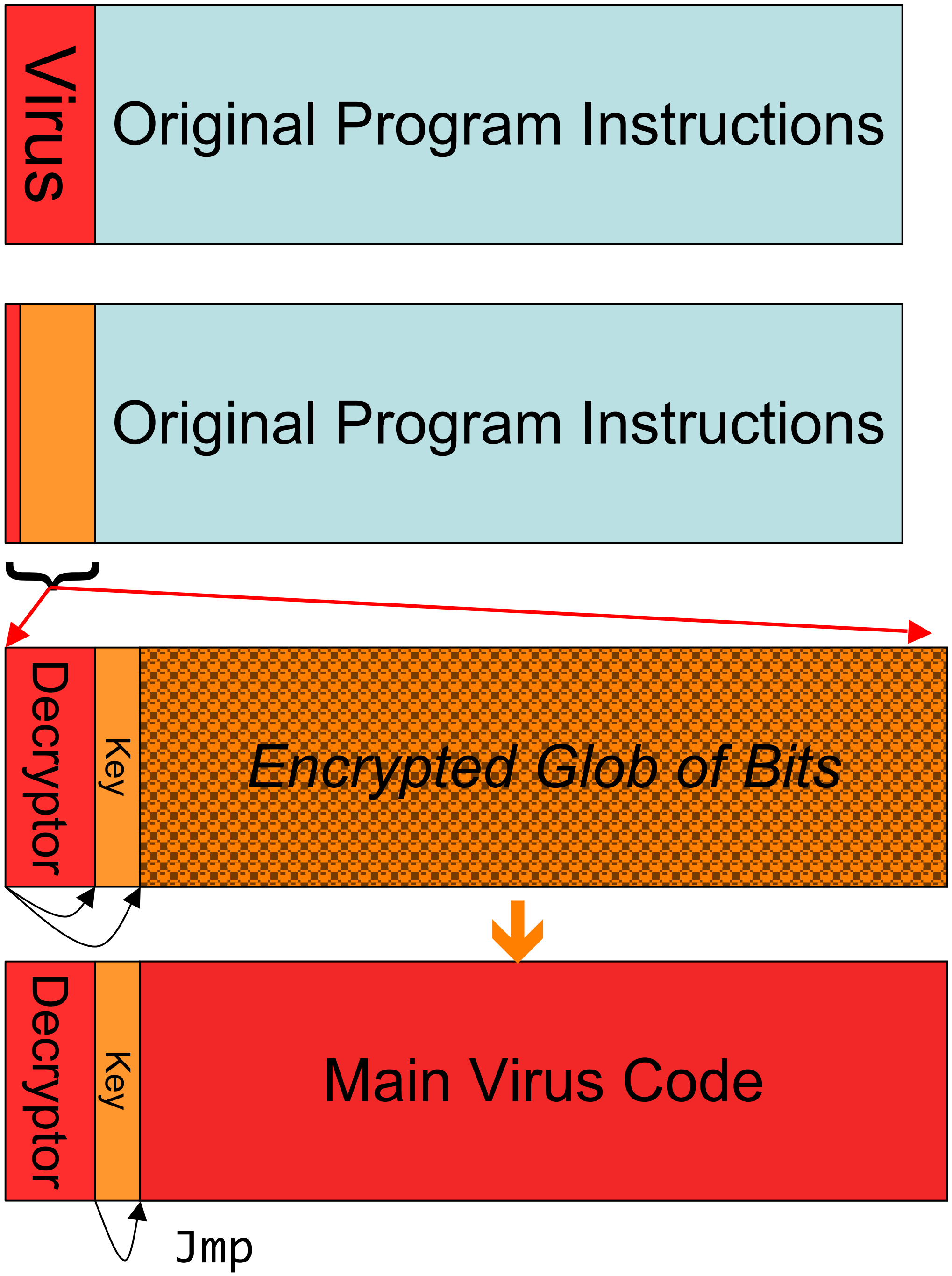
- Signature-based detection
  - Look for bytes corresponding to injected virus code
  - High utility due to replicating nature
    - If you capture a virus V on one system, by its nature the virus will be trying to infect *many other systems*
    - Can protect those other systems by installing recognizer 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)

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!
- Writing new viruses by hand takes a lot of effort
- How can you mechanize the creation of new instances of your viruses …
  - … such 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 some data that appears completely unrelated to the original data: 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



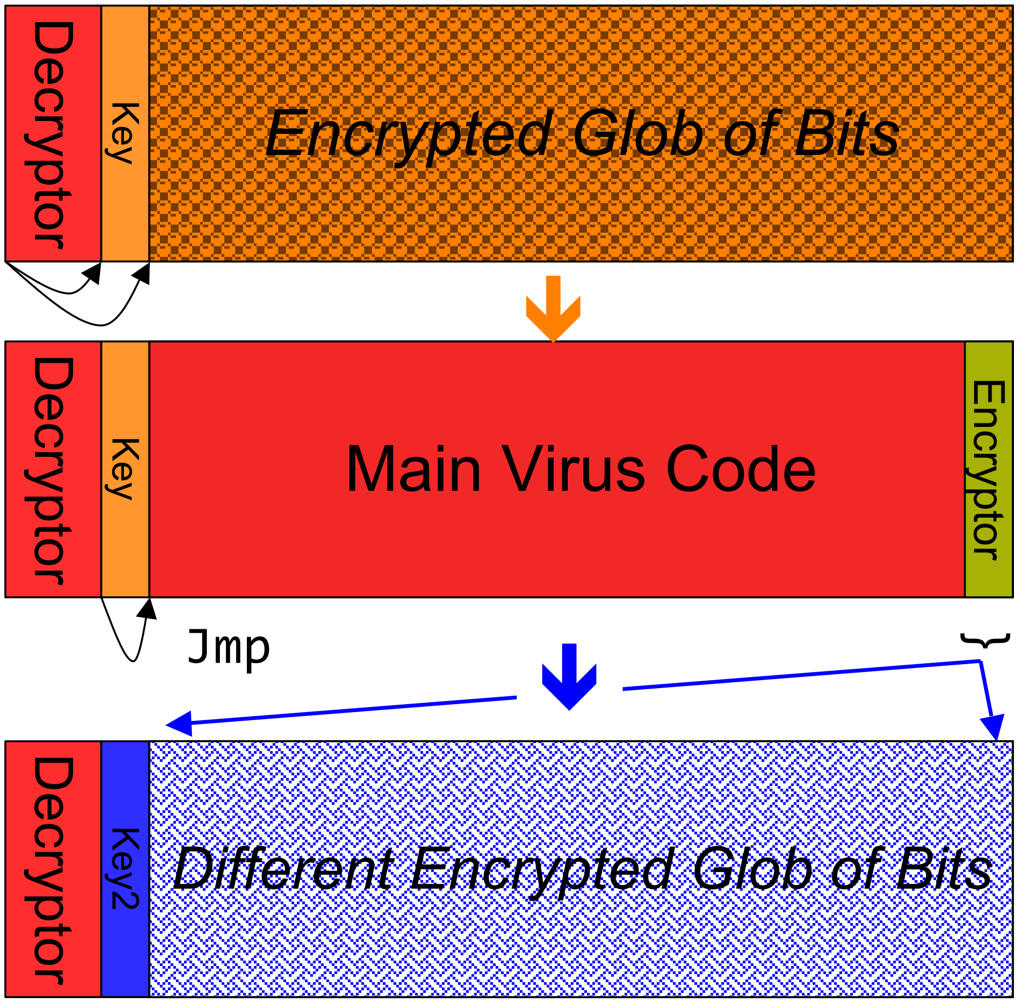
Instead of this …

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

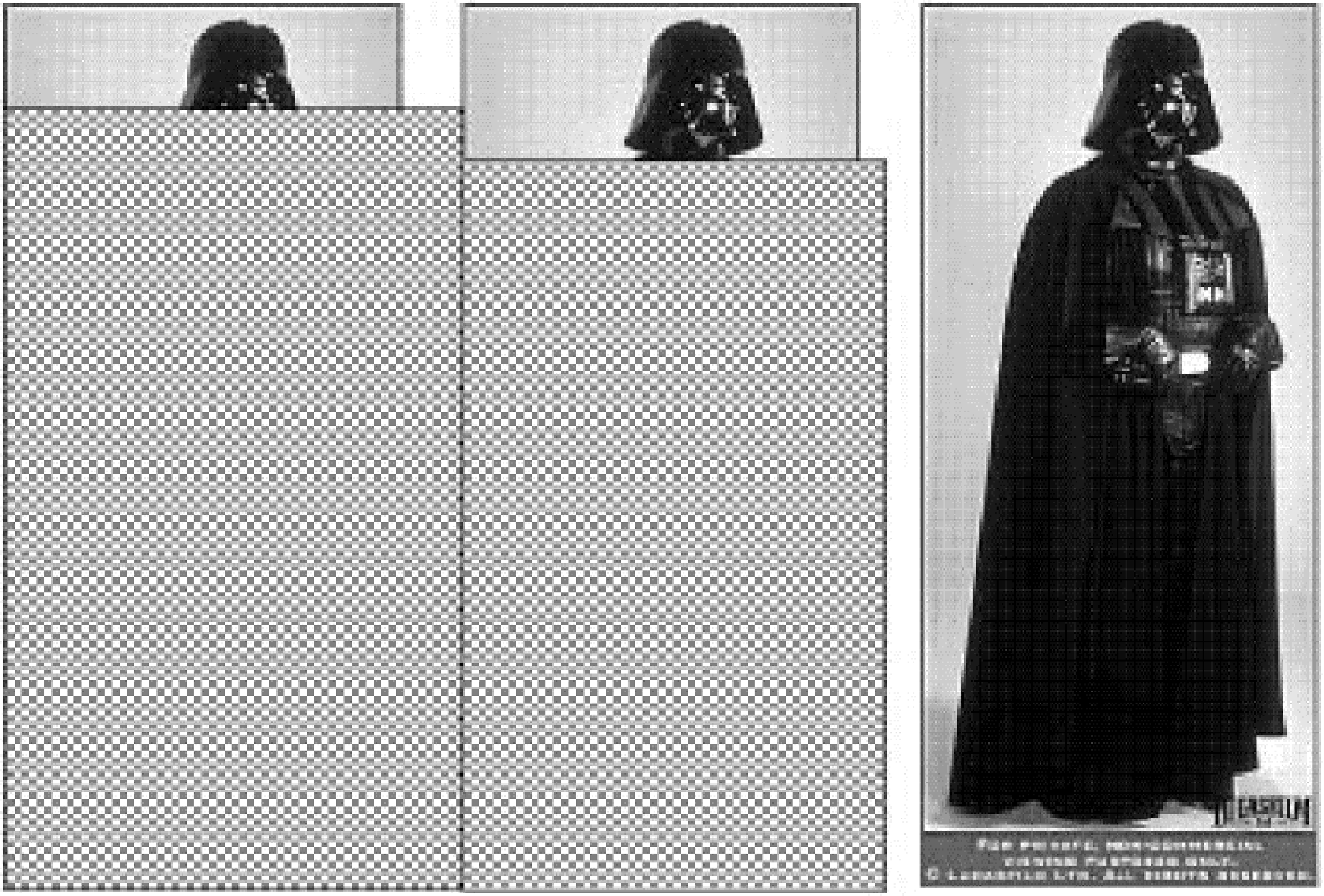
Arms Race: Polymorphic Code

- 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
      - Can be very complex, legit code … if it's never called!

Polymorphic Code In Action



Hunting for Metamorphic, Szor & Ferrie, Symantec Corp., Virus Bulletin Conference, 2001

Metamorphic Code In Action

Hunting for Metamorphic, Szor & Ferrie, Symantec Corp., Virus Bulletin Conference, 2001

Detecting Metamorphic Viruses?

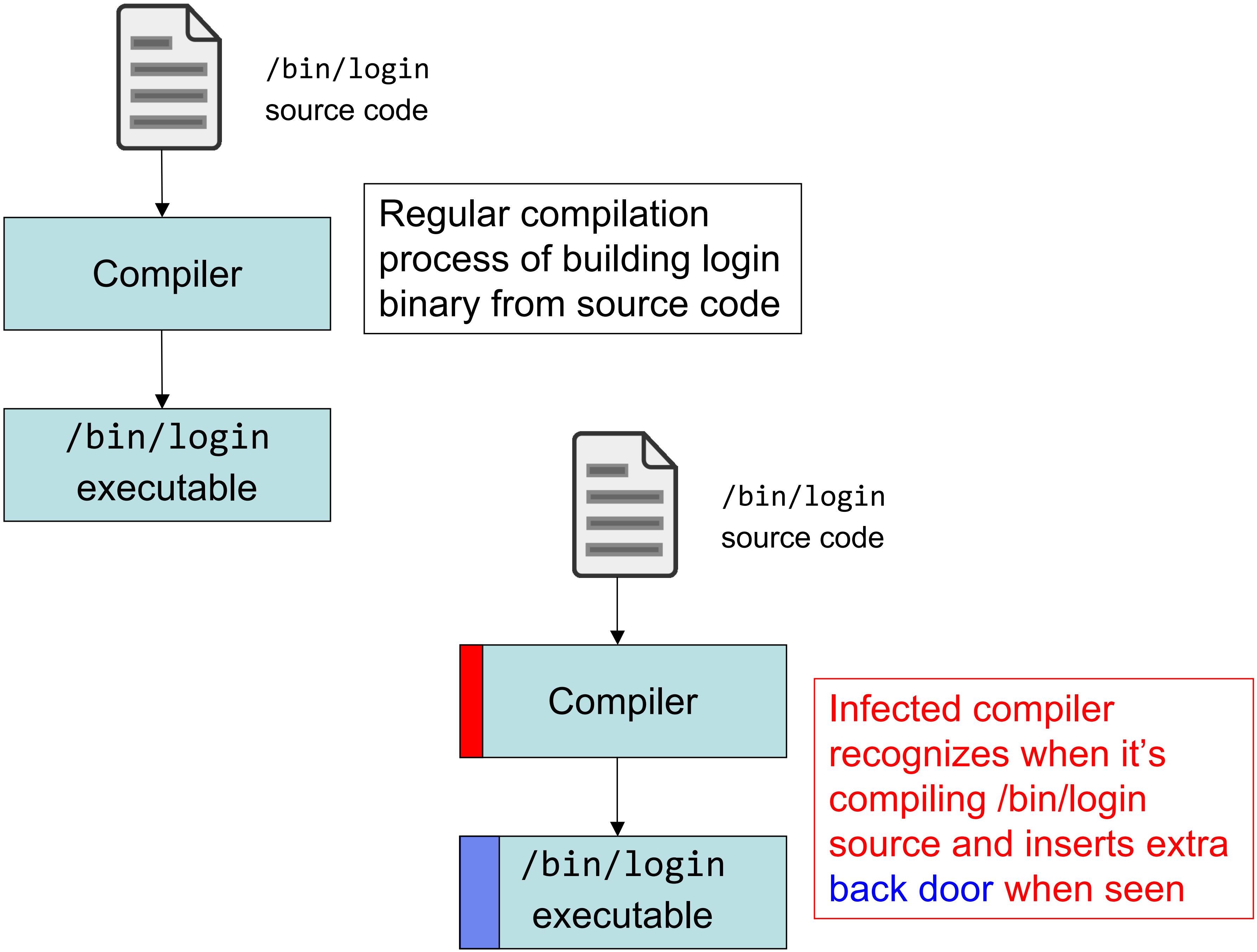
- Need to analyze execution behavior
  - Shift from syntax (*appearance* of instructions) to semantics (*effect* of instructions)
- Two stages: (1) AV company analyzes new virus to find execution signature, (2) AV software on end system analyzes suspect code to test for match to signature
- What countermeasures 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*

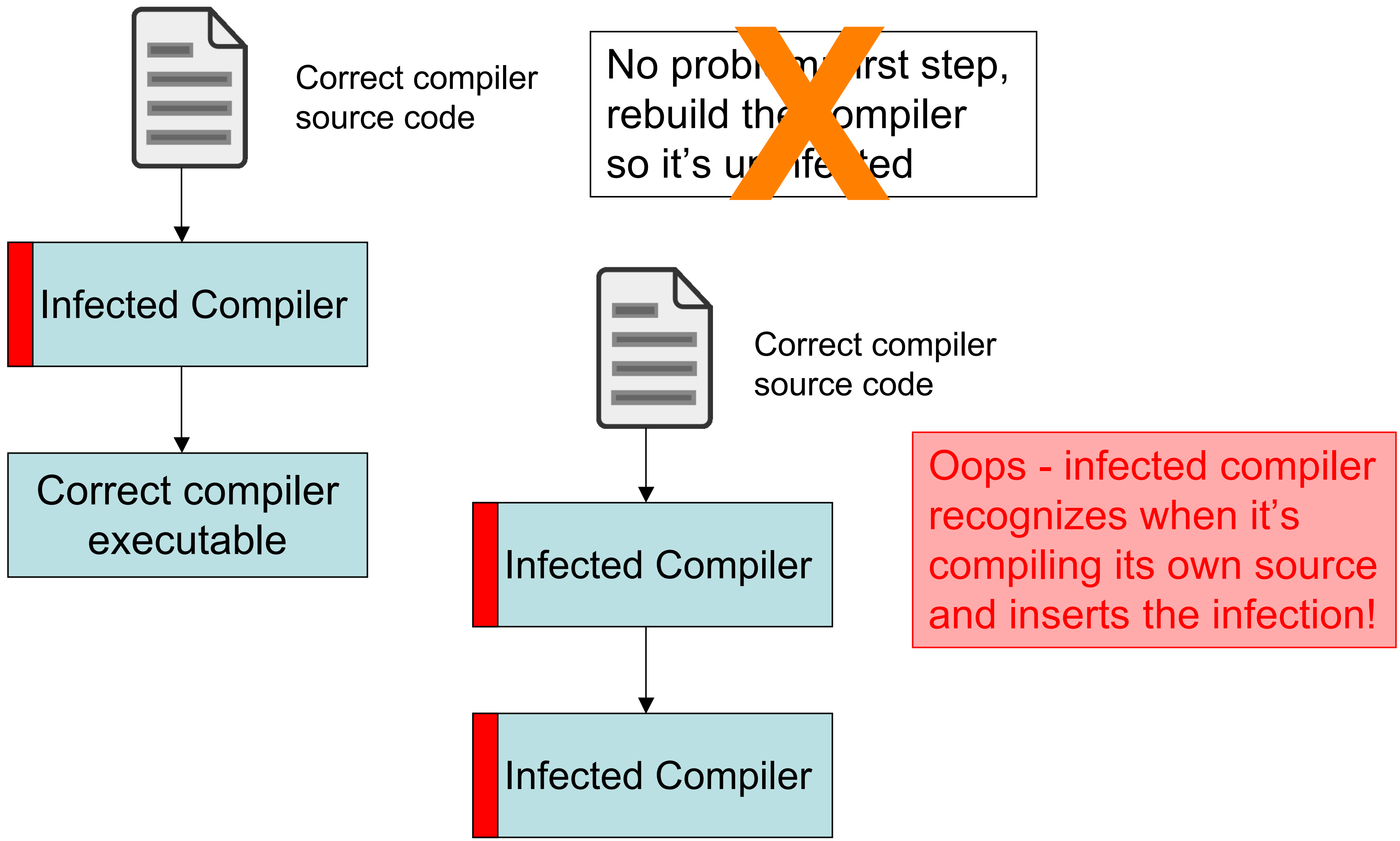
Detecting Metamorphism, con't

- Such AV analysis very expensive computationally
- Possible anomaly-based approach to reduce load by leveraging *The Cloud* ("crowdsourcing")
  - Whenever local system is about to execute a new binary, query whether anyone else across the whole Internet has already run it
    - Anyone else = other customers of AV vendor
  - If so, then it's already been analyzed as safe
  - If not, subject it to rigorous based analysis
- Note: uses notion of "anomaly" as a trigger for further action, rather than for a detection decision
- Final consideration re metamorphism: its presence can lead to mis-counting a single virus outbreak as instead reflecting 1000s of *seemingly different* viruses
  - Thus take care in interpreting vendor statistics on malcode varieties
    - (also note: public perception that many varieties exist is in their interest)

Infection Cleanup

- Once malware detected on a system, how do we get rid of it?
- May require restoring/repairing many files
- 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
- If we have complete source code for system, we could rebuild from that instead, right?


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



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