Dragonfly & Malcode

CYBERWARS ARE NOT WON BY SOLVING THE HALTING PROBLEM
CYBERWARS ARE WON BY MAKING SOME OTHER POOR SOD SOLVE THE HALTING PROBLEM
imgflip.com

News of The Day...

Dragonfly…

- WPA2-PSK sucks…
  - An eavesdropper gains enough information for an ***offline*** attack on the pre-shared password
- What we really want is “Simultaneous Authentication of Equals”
  - Alice and Bob share the same password ***PW***
  - Alice and Bob can negotiate a shared public-key based secret only if both know ***PW***
  - If Alice or Bob doesn’t know ***PW***, then they don’t learn anything about ***PW*** unless they successfully guessed ***PW*** during the protocol
    - Both Alice and Bob know the password in the clear
- Enter Dragonfly (RFC 7664)
  - Has both EC and conventional DH based variants

Last part is important: Turns offline into online-only attacks

- If Alice or Bob doesn’t know ***PW***, then they don’t learn anything about ***PW*** *unless they successfully guessed* ***PW*** during the protocol
  - Model is we have a set of possible passwords, all equally likely, which one?
- Off-line attacks are death:
  - Attacker can try as many passwords as they want in parallel
- On-line attacks are much more limited:
  - Attacker can only try one at a time…
  - And can rate-limit the attacker
- iOS passcode design is strongly set up to force online-only attacks: Even if you compromise the secure enclave, you have to try each password sequentially

DH-based Dragonfly

- Public parameters:
  - A prime $\boldsymbol{p}$
    - A generator over this $\boldsymbol{G}$
  - A (smaller) prime $\boldsymbol{q}$
    - Size of the group defined by $\boldsymbol{G}$ and $\boldsymbol{q}$ is a large prime divisor of $\boldsymbol{(p-1)/2}$
  - A selected generator $\boldsymbol{g}$ is valid if $\boldsymbol{g} < \boldsymbol{p}$ and $\boldsymbol{g}^q \bmod \boldsymbol{p} = 1$
  - Same idea as with DSA: We can use a smaller specialized group and be sending smaller data elements around
- Identifiers for Alice and Bob
  - EG, MAC addresses, with an ordering function
- Key idea:
  - Select a ***random*** generator ***g,*** called ***P*** (or PE == Password Element) based on $\boldsymbol{H(ID_a \,||\, ID_b \,||\, PW)}$
    - Hmm, I wonder where Nick got that idea for the WhyFi question? 🤔

Actually creating PE

```
found = False
counter = 1
n = len(p) + 64
do {
  base = H(max(Alice,Bob) | min(Alice,Bob) | password | counter)
  temp = KDF-n(base, "Dragonfly Hunting And Pecking")
  seed = (temp mod (p - 1)) + 1
  temp = seed ^ ((p-1)/q) mod p
  if (temp > 1)
  then
    if (not found)
      PE = temp
      found = true
    fi
  fi
  counter = counter + 1
} while ((!found) || (counter <= k))
```

Remarks…

- Called "Hunting and pecking":
  - Select a (pseudo)-Random element, check if its valid
  - If not, repeat
- We need this to resist side-channel attacks
  - So we specify a minimum iteration count ***k***
  - We select the first one, but we keep at it for a suitable ***k*** so the probability of failure is low enough: but still often 40+ times!
- We can't precompute this because we include Alice and Bob's identity in determining ***P***
  - Eliminating this would eliminate the need for online computation of ***P***
  - But we can cache ***P*** still: an important optimization since this calculation is expensive!

Now to prove that everybody knows the same P... And generate a key

- Alice creates two random values:
  - $1 < \boldsymbol{r_a} < q$ (the random value)
  - $1 < \boldsymbol{m_a} < q$ (the mask value)
- Alice now computes
  - $\boldsymbol{s_a} = (\boldsymbol{r_a} + \boldsymbol{m_a}) \bmod \boldsymbol{q}$
  - $\boldsymbol{E_a} = \boldsymbol{P^{-mask}}$
  - Sends those to Bob, Bob sends his counterparts to Alice
- Now the starting secret…
  - $\boldsymbol{ss} = (\boldsymbol{P^{s_b} E_b})^{\boldsymbol{r_a}} = (\boldsymbol{P^{(r_b + m_b - m_b)}})^{\boldsymbol{r_a}} = \boldsymbol{P^{r_a r_b}}$
  - Sends those to Bob, Bob sends his counterparts
  - Verify $\boldsymbol{P^{sb}}$ and $\boldsymbol{Sb}$ are valid
  - Computes $\boldsymbol{H(ss|E_a|s_a|E_b|s_b)}$ and sends that to Bob
  - verifies Bob's counterpart
- Final:
  $\boldsymbol{K = H(ss|E_a * E_b|s_a + s_b)}$

Graphically

Alice

Calculate: P

Randoms: $1 < \boldsymbol{r_a} < q$, $1 < \boldsymbol{m_a} < q$

$\boldsymbol{s_a} = (\boldsymbol{r_a} + \boldsymbol{m_a}) \bmod \boldsymbol{q}$

$\boldsymbol{E_a} = \boldsymbol{P}^{\boldsymbol{-m_a}}$

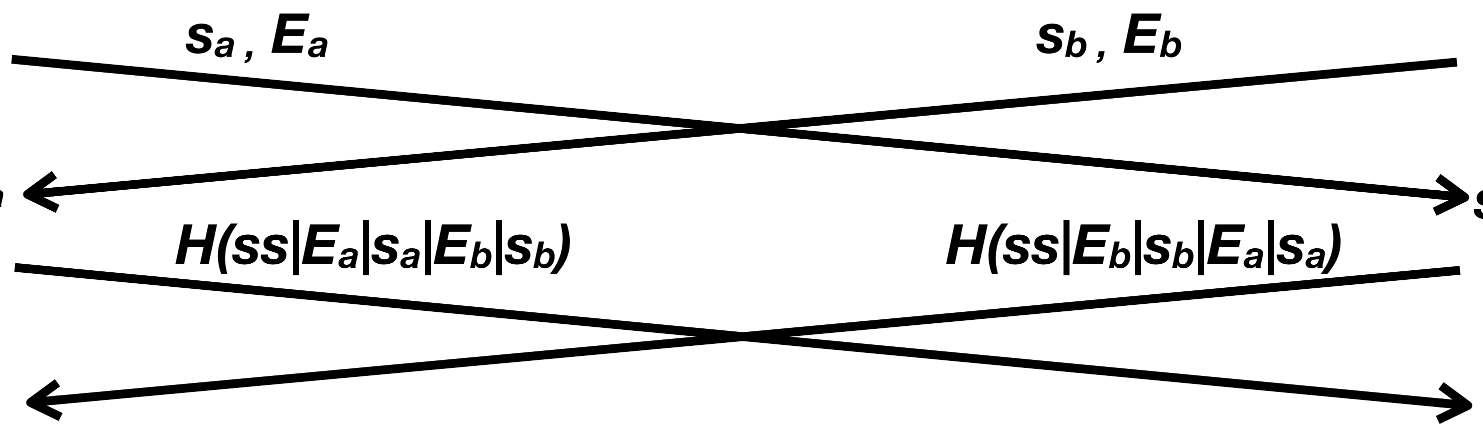
Bob

Calculate: P

Randoms: $1 < \boldsymbol{r_b} < q$, $1 < \boldsymbol{m_b} < q$

$\boldsymbol{s_b} = (\boldsymbol{r_b} + \boldsymbol{m_b}) \bmod \boldsymbol{q}$

$\boldsymbol{E_b} = \boldsymbol{P}^{\boldsymbol{-m_b}}$



$ss = (P^{s_b}E_b)^{ra}$

$ss = (P^{s_a}E_a)^{rb}$

Verify

Verify

$$K = H(ss|E_a * E_b|s_a + s_b)$$

Use in WPA3

- WPA3 does this
  - Well, over an elliptic curve instead, but same idea:
    Generate a random generator and use that
- But it is not during the 4-way handshake…
  - Instead, it is 2 additional handshakes ***before*** the 4-way handshake
  - Result is higher latency but, eh, 🤷‍♂️
- Exists correctness and security proofs
- Result is WPA3:
  - ***Eliminates*** the off-line brute force attacks
  - ***Eliminates*** the "adversary with the password" L2 attacks

Malware: Catch-All Term for "Malicious Code"

- Attacker code running on victim computer(s)
- Two parts:
  - How it gets there (propagation)
  - What it does (payload)

What Can Malware Payload Do?

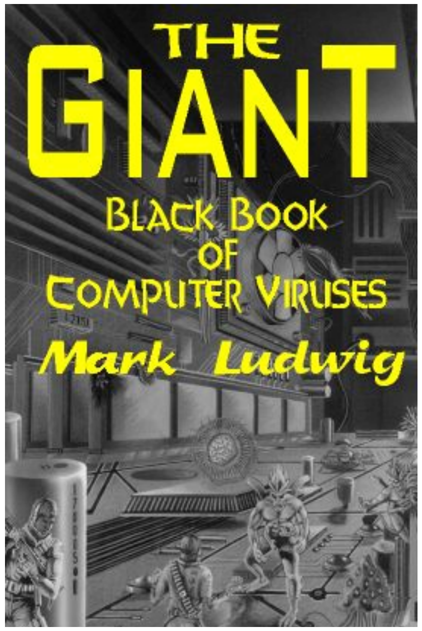
- Pretty much anything
  - Payload generally decoupled from how manages to run
  - Only subject to permissions under which it runs
- Examples:
  - Brag or exhort or extort (pop up a message/display)
  - Trash files (just to be nasty)
  - Launch external activity (spam, click fraud, DoS; banking)
  - Steal information (exfiltrate)
  - Keylogging; screen / audio / camera capture
  - Encrypt files (ransomware)
  - Cause ***physical*** damage
- 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)
  - ***Trojan*** = code that does ***NOT*** self propagate, but instead requires a user action
- ***NO EXPERIMENTATION WITH SELF REPLICATING CODE!***

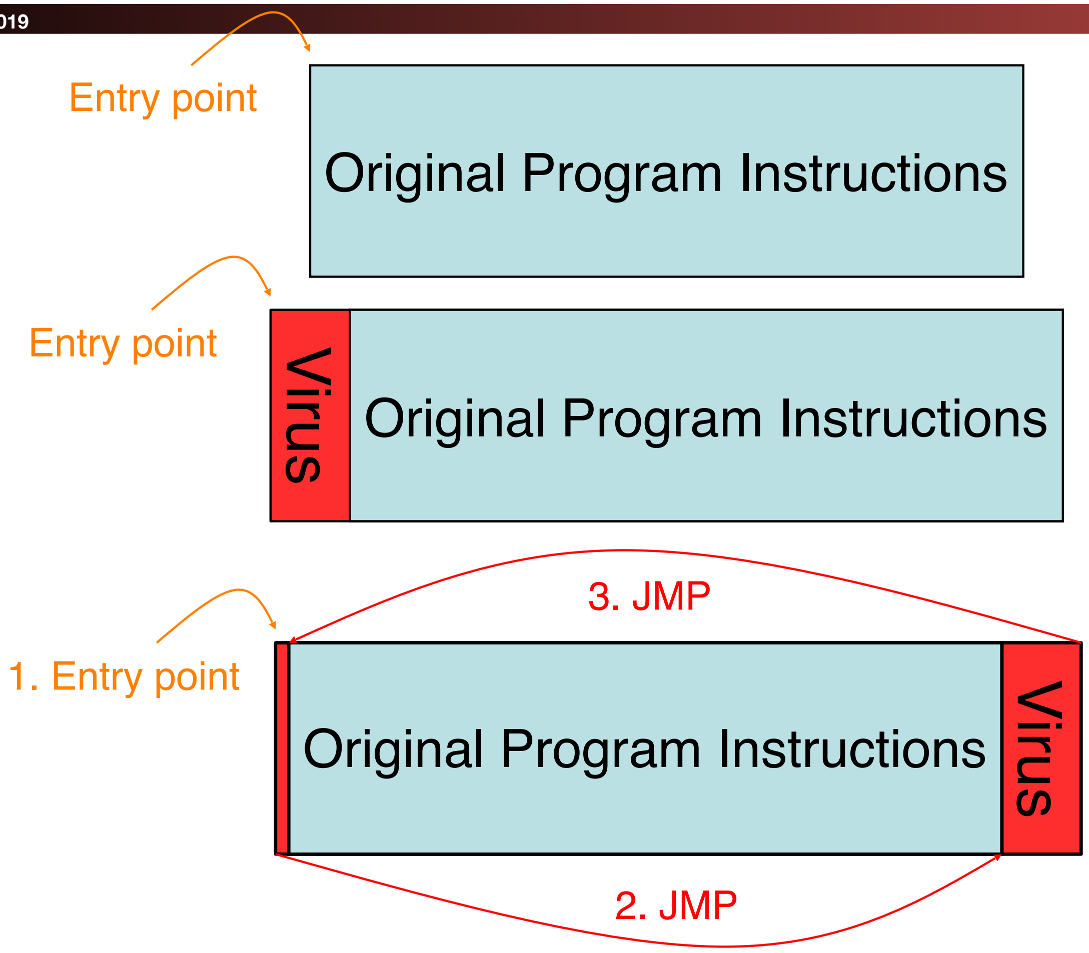
The Problem of Viruses

- 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")


Entry point
Original Program Instructions
Entry point
Virus
Original Program Instructions
3. JMP
1. Entry point
Original Program Instructions
Virus
2. JMP

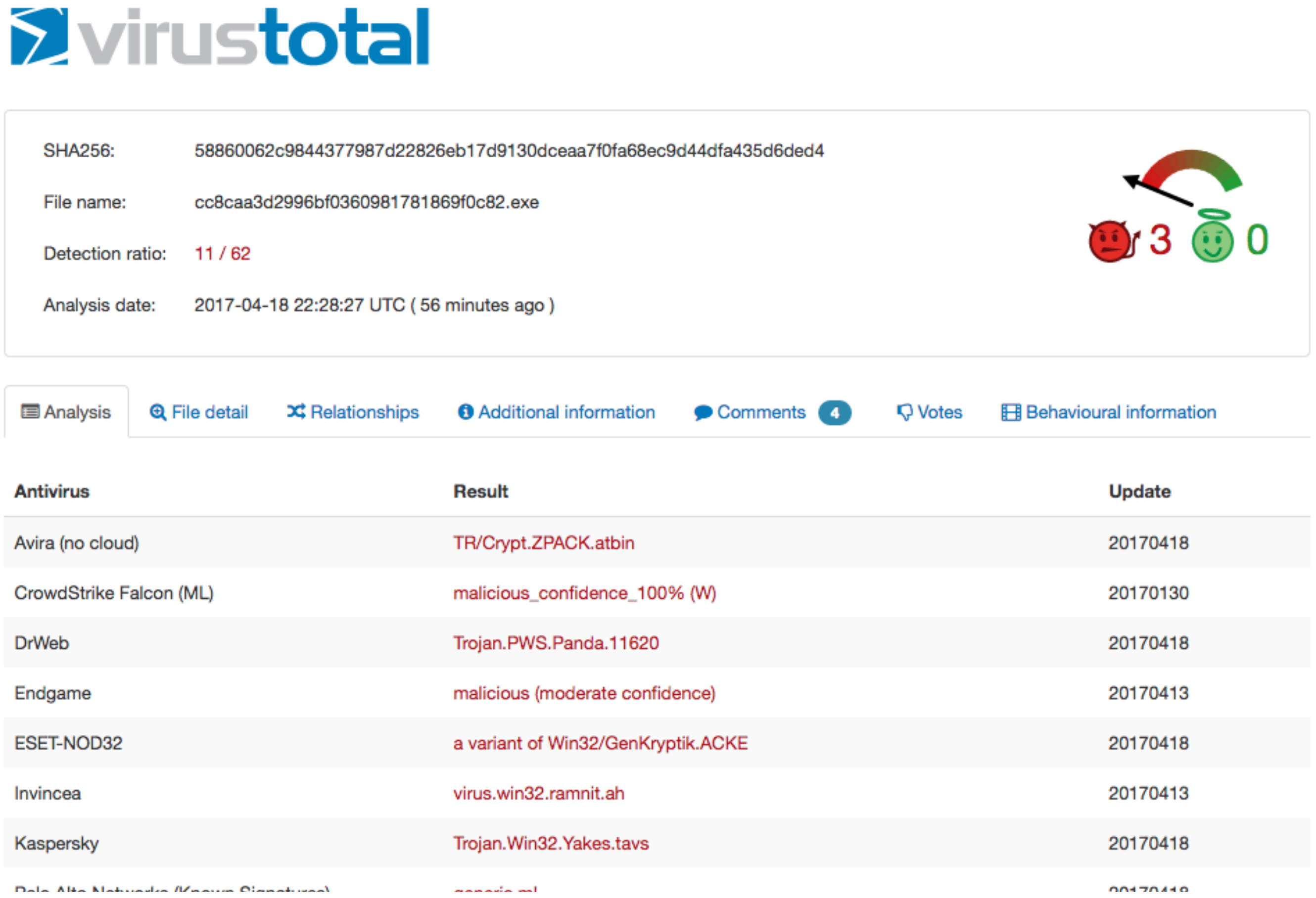
Original program instructions can be:

- 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)

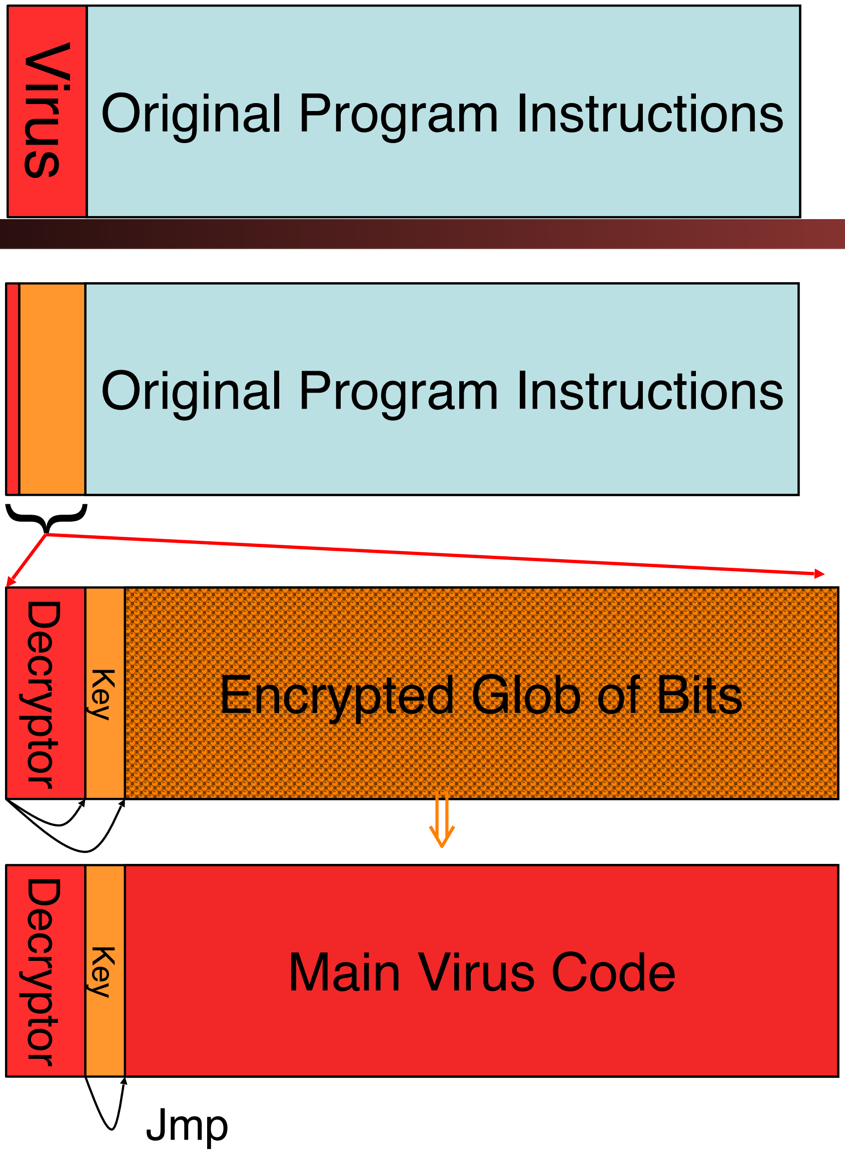
Antivirus	Result	Update
Avira (no cloud)	TR/Crypt.ZPACK.atbin	20170418
CrowdStrike Falcon (ML)	malicious_confidence_100% (W)	20170130
DrWeb	Trojan.PWS.Panda.11620	20170418
Endgame	malicious (moderate confidence)	20170413
ESET-NOD32	a variant of Win32/GenKryptik.ACKE	20170418
Invincea	virus.win32.ramnit.ah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



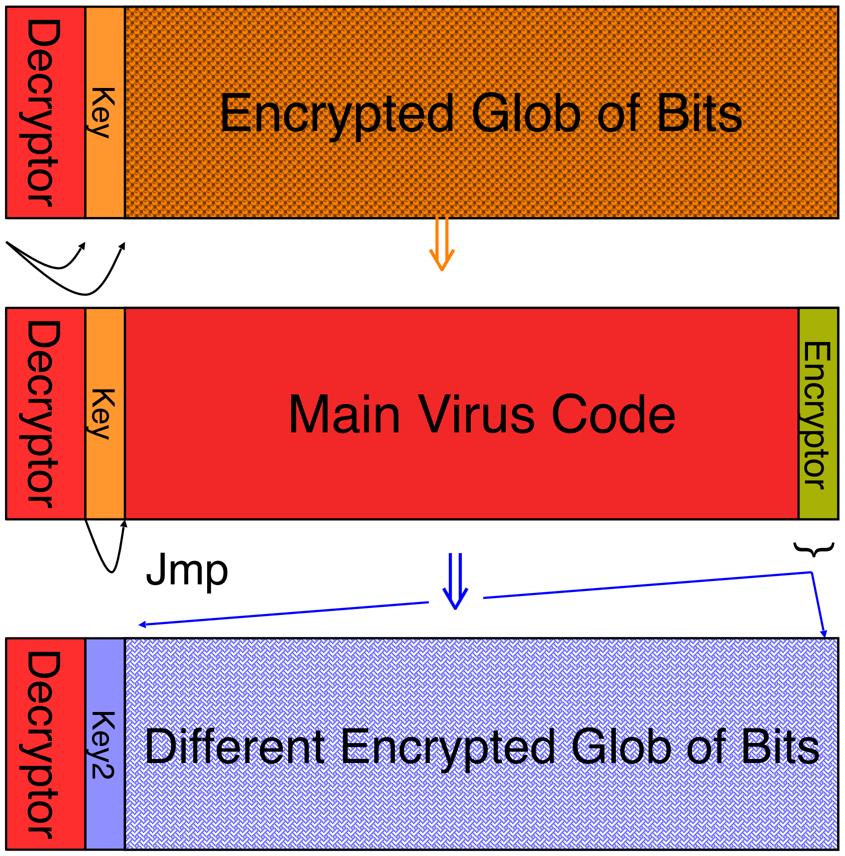
Instead of this …

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

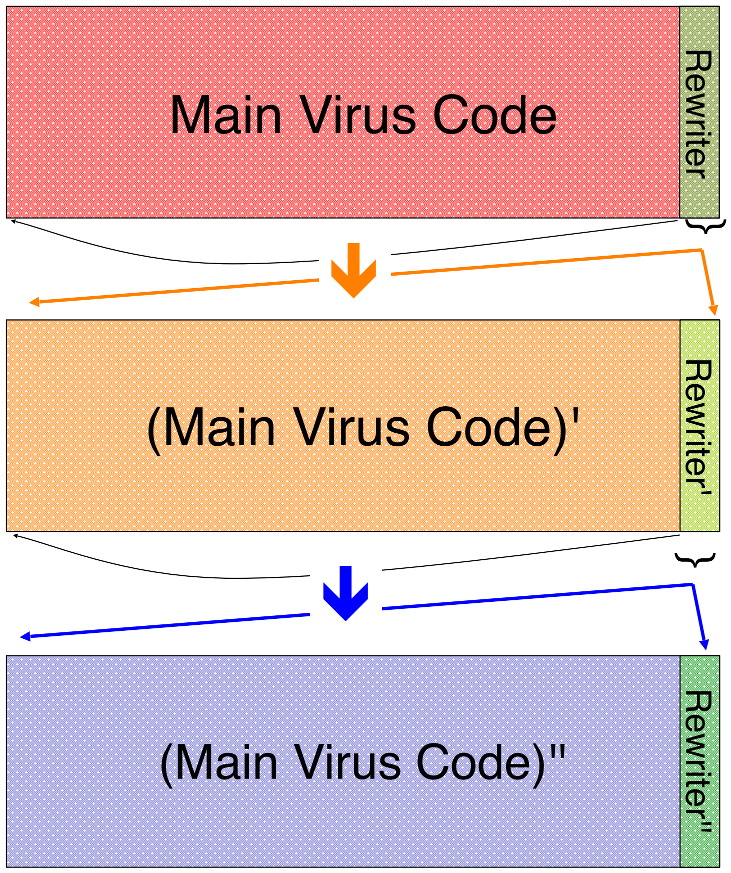
Arms Race: Polymorphic Code

- 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 (<u>including the rewriter</u>)

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

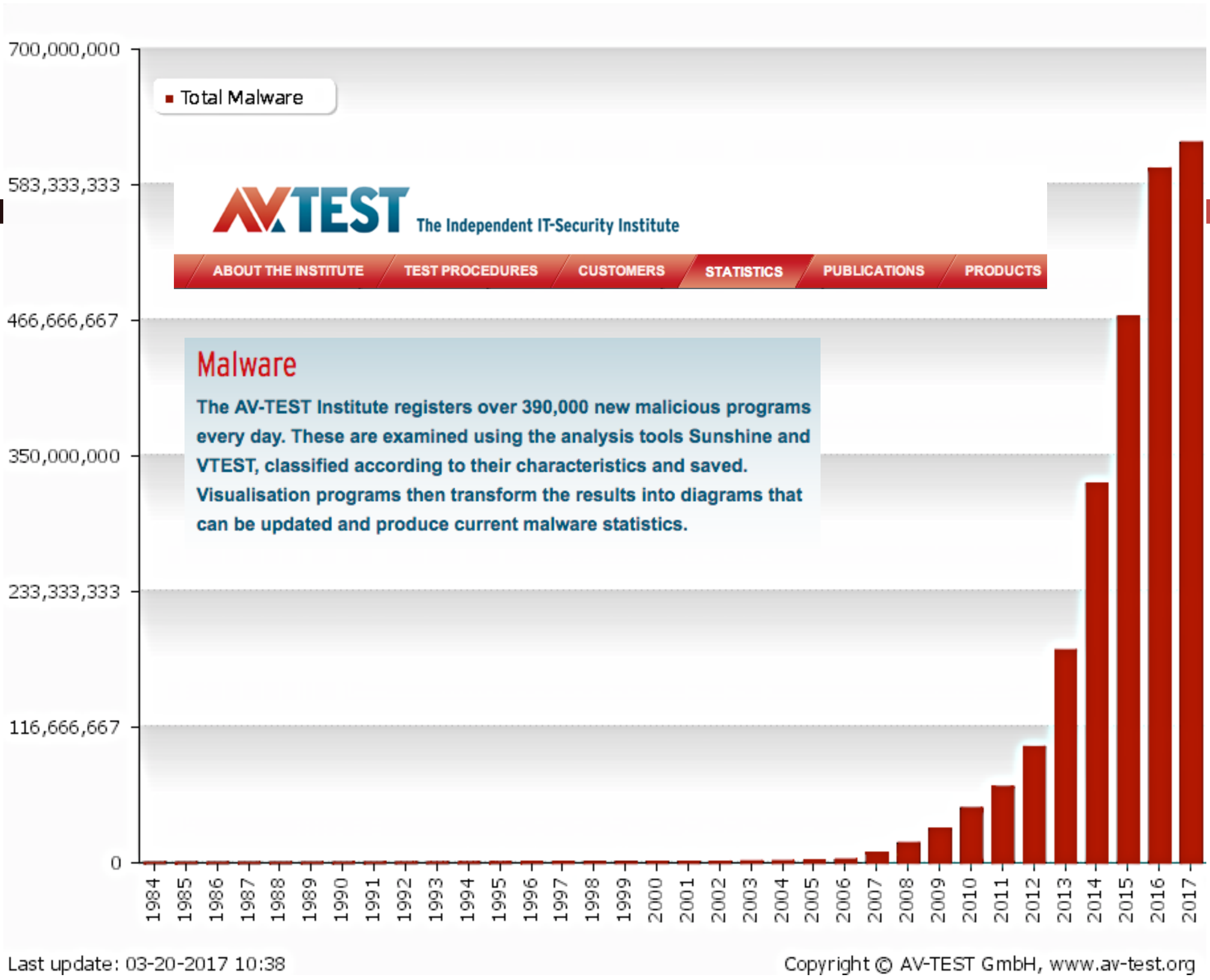
CYBERWARS ARE NOT WON BY SOLVING THE HALTING PROBLEM
CYBERWARS ARE WON BY MAKING SOME OTHER POOR SOD SOLVE THE HALTING PROBLEM
imgflip.com

Creating binds is very powerful...

- You have a detector D for some bad behavior...
    - So bad-guys come up with a way of avoiding the 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)



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.

Last update: 03-20-2017 10:38

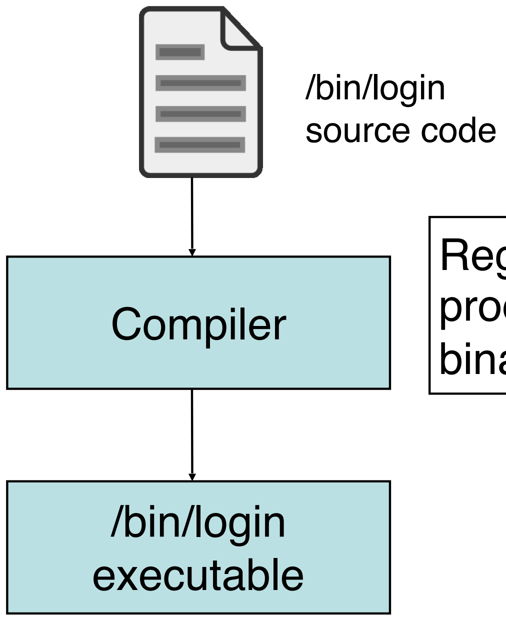
Copyright © AV-TEST GmbH, www.av-test.org

Infection Cleanup

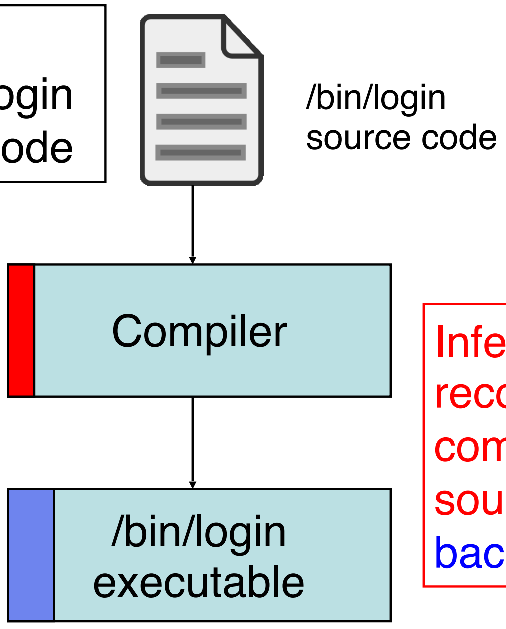
- Once malware detected on a system, how do we get rid of it?
- May require restoring/repairing many files
  - This is part of what AV companies sell: per-specimen disinfection procedures
- What about if malware executed with adminstrator privileges?
  - "Game over man, Game Over!"
  - “Dust off and 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, 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 …

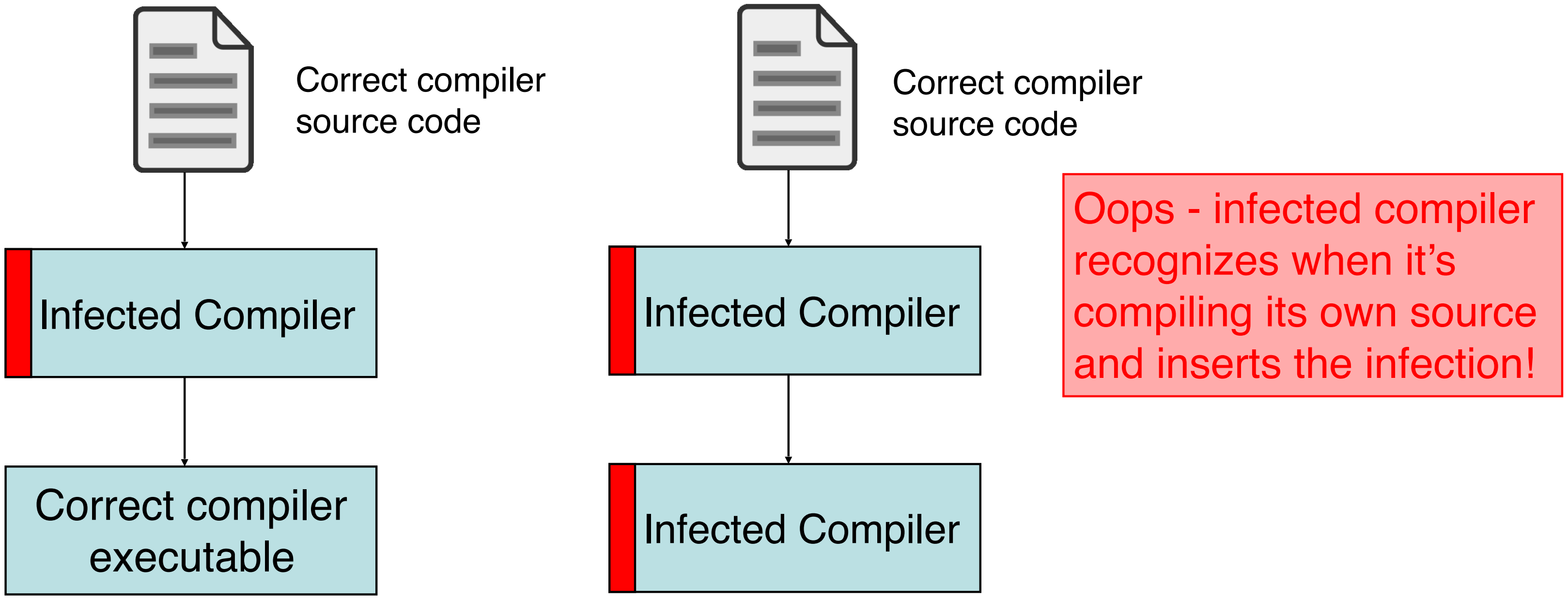


Regular compilation process of building login binary from source code



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~~


Correct compiler source code
Infected Compiler
Correct compiler executable
Correct compiler source code
Infected Compiler
Infected Compiler

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!"

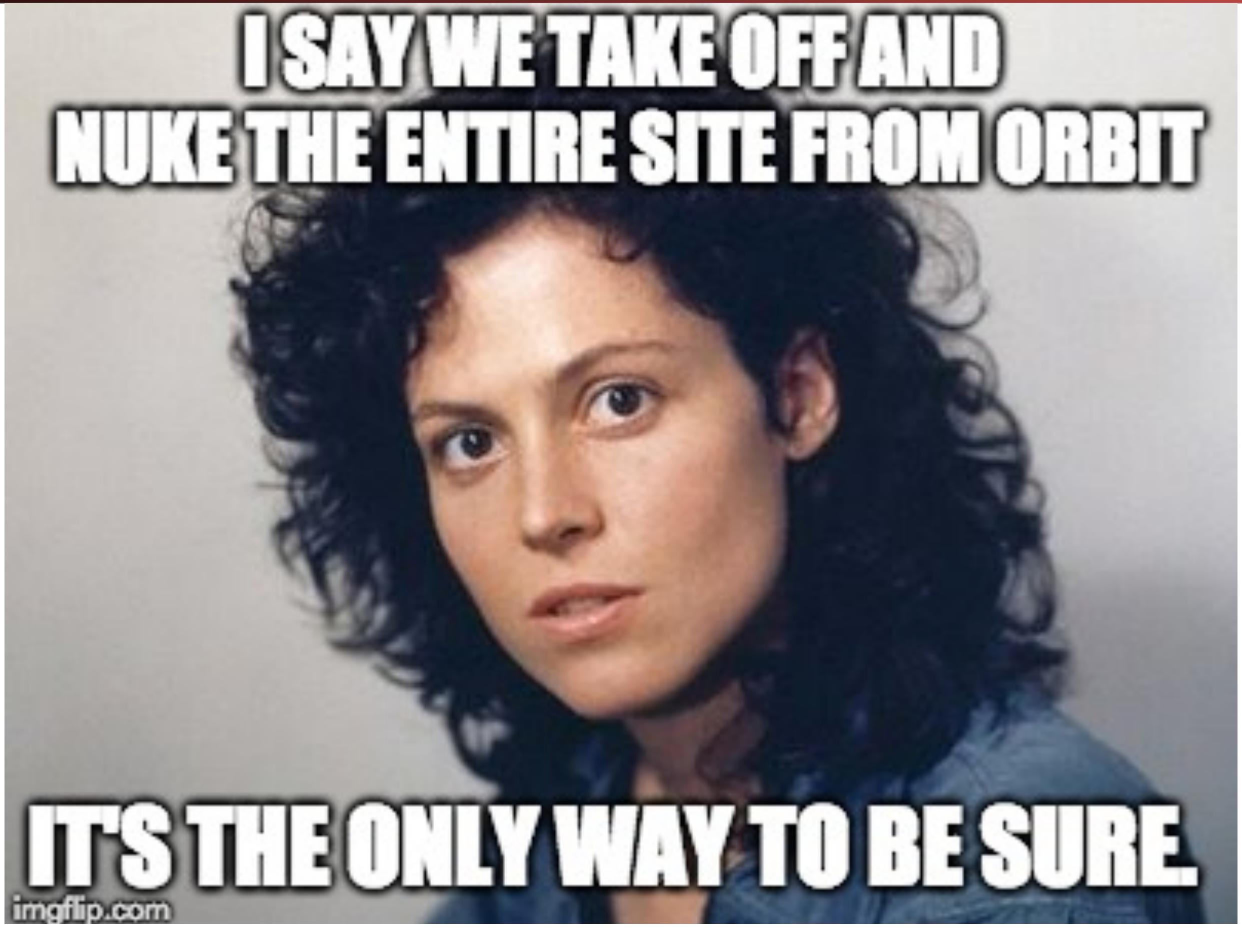
Even More Places To Hide!

- 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...


I SAY WE TAKE OFF AND
NUKE THE ENTIRE SITE FROM ORBIT
IT'S THE ONLY WAY TO BE SURE.
imgflip.com