# CSE 127 Computer Security

Stefan Savage, Spring 2020, Lecture 16

## Malware I

# Today

- We've talked about ways that machines can be compromised
- But what happens afterwards
  - Viruses
  - Worms
  - Malware detection

# Viruses & worms

- **Replicating** malicious programs
  - Viruses *replicate* by attaching to a host program (or document)
    - Copying themselves into new programs/documents they encounter
    - Traditionally driven by human action (e.g., opening document)
  - Worms *replicate* via the network
    - Each compromised host tries to infect *other* hosts; parallelism
    - Self-spreading

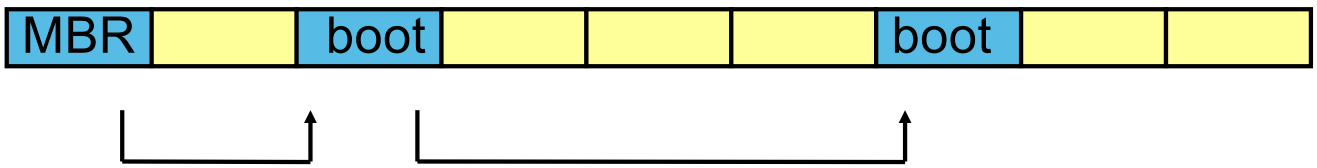
- Goals:
  - Spread
    - This may include evading detection
  - Accomplish their goal (payload)... whatever that is

# Virus design history

- **Bootstrap viruses**
  - Historically important (less common today, except as rootkit)
- Memory resident viruses
  - Standard infected executable
- Encrypted viruses
- Polymorphic/Metamorphic viruses

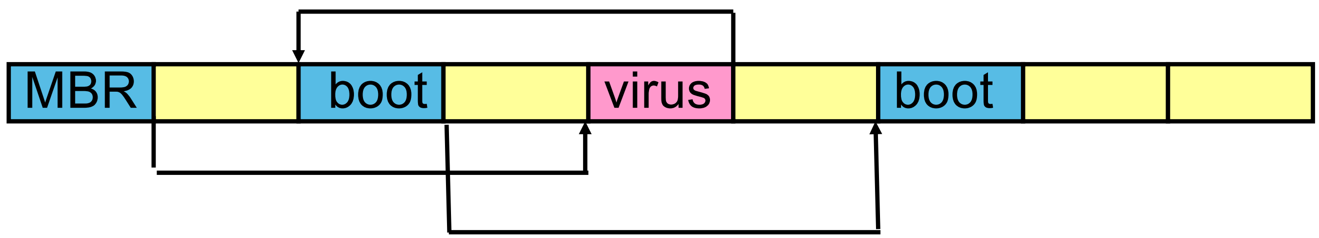
- Each new advancement is the result of co-evolution – Darwinian requirement that malware authors improve to survive

# Boot sector Viruses
# (old school, Elk Cloner 1981)




- Original vector: floppy disks
- Old school Bootstrap Process:
  - Firmware (ROM) copies MBR (master boot record) to memory, jumps to that program
- MBR (or Boot Sector)
  - Fixed position on disk
  - “Chained” boot sectors permit longer Bootstrap Loaders

# Boot sector Viruses




- Virus breaks the chain
- Inserts virus code
- Reconnects chain afterwards
- Same approach applies to hard disk as well...

# Why attack the Bootstrap loader?

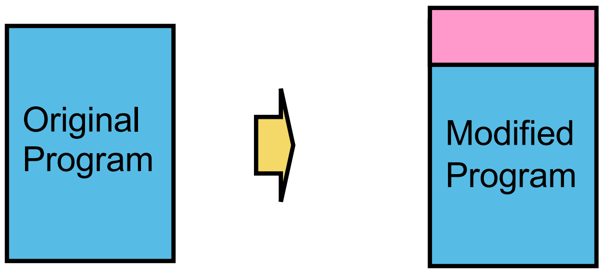
- For floppy disks, this was the mechanism for spreading
  - Infect system on bootup, infect any new floppy disks used; virus spreads as disks moved from computer to computer
- Protection: automatically executed *before* OS is running
  - Any thus, before detection tools are running
  - OS hides boot sector information from users
    - Harder to discover that the virus is there
    - Harder to fix (can just delete a file)
- Modern variants (persistence and stealth)
  - Meebroot – boot sector rootkit
  - IRATEMONK – NSA rootkit that rewrites hard drive firmware
- Solutions
  - Good: Modern malware scanning tools will scan the bootsector
  - Better: Secure bootstrap (firmware validates signature on bootstrap code, which validates signature on OS loader, which validates… etc)

# Virus design history

- Bootstrap viruses
  - Historically important (less common today, except as rootkit)
- **Memory resident viruses**
  - Standard infected executable
- Encrypted viruses
- Polymorphic/Metamorphic viruses

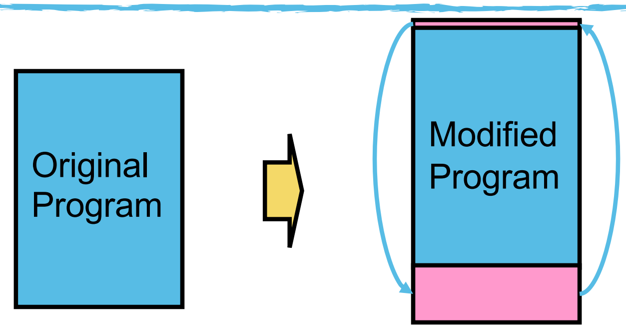
- Each new advancement is the result of co-evolution – Darwinian requirement that malware authors improve to survive

# Virus Attachment to Host Program




- First attempt: insert copy at the beginning of an executable file
  - Runs before other code of the program
    - Select other file to infect; repeat
  - Works fine for position independent code

# Virus Attachment to Host Program




- First attempt: insert copy at the beginning of an executable file
  - Runs before other code of the program
    - Select other file to infect; repeat
  - Works fine for position independent code
- Simple alternative: add virus code to **end** of program, redirect control flow there at program start then jump back to program body

# The Simple Virus

```
0100 EB1C          JMP     011E
0102 BE1B02        MOV     SI,021B
0105 BF1B01        MOV     DI,011B
0108 8BCE          MOV     CX,SI
010A F7D9          NEG     CX
010C FC            CLD
010D B81B01        MOV     AX,011B
0110 06            PUSH    ES
0111 50            PUSH    AX
0112 06            PUSH    ES
0113 B81801        MOV     AX,0118
0116 50            PUSH    AX
0117 CB            RETF
0118 F3            REPZ
0119 A4            MOVSB
011A CB            RETF
011B E93221        JMP     2250
011E 83C24F        ADD     DX,+4F
0121 8BFA          MOV     DI,DX
0123 81FF8000      CMP     DI,0080
0127 725E          JB      0187
0129 7406          JZ      0131
012B C606250273    MOV     BYTE PTR [0225],73
0130 90            NOP
0131 FEC5          INC     CH
0133 7303          JNB     0138
0135 80C140        ADD     CL,40
0138 B8010C        MOV     AX,0C01
013B 8BD6          MOV     DX,SI
013D CD13          INT     13
```

Infected Program

1. User runs an infected program.
2. Program transfers control to the virus.

# The Simple Virus

```
0100 EB1C          JMP     011E
0102 BE1B02        MOV     SI,021B
0105 BF1B01        MOV     DI,011B
0108 8BCE          MOV     CX,SI
010A F7D9          NEG     CX
010C FC            CLD
010D B81B01        MOV     AX,011B
0110 06            PUSH    ES
0111 50            PUSH    AX
0112 06            PUSH    ES
0113 B81801        MOV     AX,0118
0116 50            PUSH    AX
0117 CB            RETF
0118 F3            REPZ
0119 A4            MOVSB
011A CB            RETF
011B E93221        JMP     2250
011E 83C24F        ADD     DX,+4F
0121 8BFA          MOV     DI,DX
0123 81FF8000      CMP     DI,0080
0127 725E          JB      0187
0129 7406          JZ      0131
012B C606250273    MOV     BYTE PTR [0225],73
0130 90            NOP
0131 FEC5          INC     CH
0133 7303          JNB     0138
0135 80C140        ADD     CL,40
0138 B8010C        MOV     AX,0C01
013B 8BD6          MOV     DX,SI
013D CD13          INT     13
```

Infected Program

```
0100 B435          MOV     AH,35
0102 B021          MOV     AL,21
0104 CD21          INT     21
0106 8C06A002      MOV     [02A0],ES
010A 891E9E02      MOV     [029E],BX
010E B425          MOV     AH,25
0110 B021          MOV     AL,21
0112 BA2001        MOV     DX,0120
0115 CD21          INT     21
0117 83C24F        ADD     DX,+4F
011A 8BFA          MOV     DI,DX
011C 81FF8000      CMP     DI,0080
0120 725E          JB      0187
0122 7406          JZ      0131
0124 C606250273    MOV     BYTE PTR [0225],73
0129 90            NOP
012A FEC5          INC     CH
012C 7303          JNB     0138
012E 80C140        ADD     CL,40
0132 B8010C        MOV     AX,0C01
0135 8BD6          MOV     DX,SI
0137 CD13          INT     13
```

3. Virus locates a new program.
4. Virus appends its logic to the end of the new file.

# The Simple Virus

```
0100 EB1C          JMP     011E
0102 BE1B02        MOV     SI,021B
0105 BF1B01        MOV     DI,011B
0108 8BCE          MOV     CX,SI
010A F7D9          NEG     CX
010C FC            CLD
010D B81B01        MOV     AX,011B
0110 06            PUSH    ES
0111 50            PUSH    AX
0112 06            PUSH    ES
0113 B81801        MOV     AX,0118
0116 50            PUSH    AX
0117 CB            RETF
0118 F3            REPZ
0119 A4            MOVSB
011A CB            RETF
011B E93221        JMP     2250
011E 83C24F        ADD     DX,+4F
0121 8BFA          MOV     DI,DX
0123 81FF8000      CMP     DI,0080
0127 725E          JB      0187
0129 7406          JZ      0131
012B C606250273    MOV     BYTE PTR [0225],73
0130 90            NOP
0131 FEC5          INC     CH
0133 7303          JNB     0138
0135 80C140        ADD     CL,40
0138 B8010C        MOV     AX,0C01
013B 8BD6          MOV     DX,SI
013D CD13          INT     13
```

Infected Program

```
0100 EB1C          JMP     0117
0102 B021          MOV     AL,21
0104 CD21          INT     21
0106 8C06A002      MOV     [02A0],ES
010A 891E9E02      MOV     [029E],BX
010E B425          MOV     AH,25
0110 B021          MOV     AL,21
0112 BA2001        MOV     DX,0120
0115 CD21          INT     21
0117 83C24F        ADD     DX,+4F
011A 8BFA          MOV     DI,DX
011C 81FF8000      CMP     DI,0080
0120 725E          JB      0187
0122 7406          JZ      0131
0124 C606250273    MOV     BYTE PTR [0225],73
0129 90            NOP
012A FEC5          INC     CH
012C 7303          JNB     0138
012E 80C140        ADD     CL,40
0132 B8010C        MOV     AX,0C01
0135 8BD6          MOV     DX,SI
0137 CD13          INT     13
```

5. Virus updates the new program so the virus gets control when the program is launched.

# Detecting Viruses

- **Scanning (signatures)**

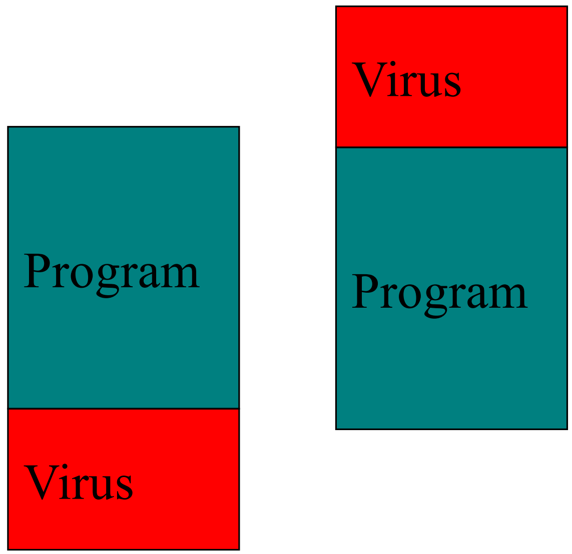
- Integrity checking (check if file has changed)
  - Keep "known good" hash of existing executables (whitelist); validate programs on computer against whitelist
- Behavior (heuristic) detection
  - E.g. does software use system features atypical of an application program; make anomalous network access; try to read sensitive files, etc…

# Virus Signatures

- Viruses can’t be completely invisible:
  - Code must be stored somewhere
  - Virus must do something when it runs
  - Identify **existing viruses** and extract “**signature**” byte sequences unique to them
  - Idea: look in files these signatures
- Issues
  - Where to scan (beginning of file, whole file, registry settings, etc)
  - How to scan (just look for string, or actually execute program)
  - How long to scan (tradeoffs in **performance**/coverage)
  - Are we sure there is a common signature?

# Head/Tail Scanners

Early application-infecting viruses attached themselves to either the top or bottom of the host file:




So anti-virus engineers built head/tail scanners.

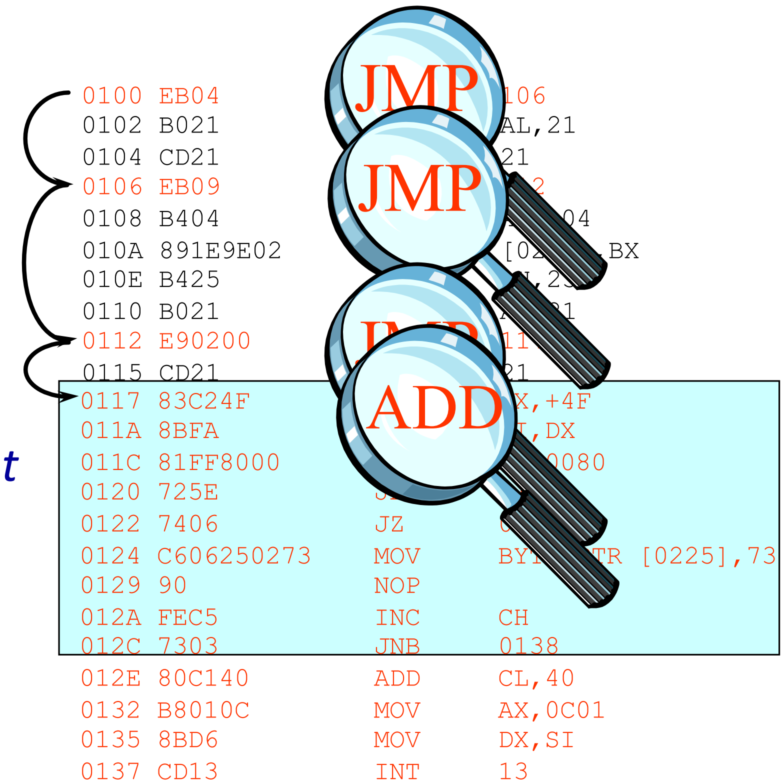
The scanner loads the head and tail regions of the file into a buffer and then scans with a multi-string search algorithm.

# So what do the bad guys do?

- Move the virus to the middle of the file
- Becomes prohibitively expensive to scan
  - Must scan whole file
- Solution: scalpel scanning
  - Idea: limit scanning to likely **entry-points** for viruses
  - If you have more time you can also scan for more than just strings (regular expressions)

# Scalpel scanning

1. Locate the main program entry-point.
2. While the current instruction is a JUMP or a CALL instruction, trace it.
3. If the current instruction is *not* a JUMP or CALL instruction, search for all fingerprints in this region of the file.



# Virus design history

- Bootstrap viruses
  - Historically important (less common today, except as rootkit)
- Memory resident viruses
  - Standard infected executable
- **Encrypted viruses**
- **Polymorphic/Metamorphic viruses**

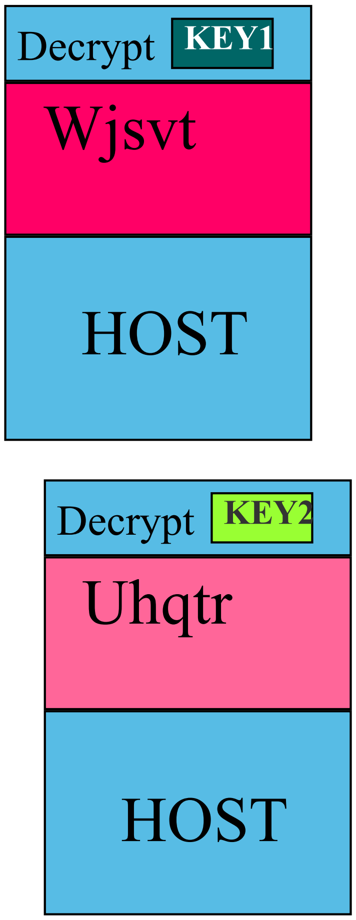
- Each new advancement is the result of co-evolution – Darwinian requirement that malware authors improve to survive

# Encrypted viruses

Soon after the first generation of executable viruses, virus authors began writing self-encrypting strains.

These viruses carry a small decryption loop that runs first, decrypts the virus body and then launches the virus.

Each time the virus infects a new file, it changes the encryption key so the virus body looks different.



# Encrypted viruses

```
1. MOV DI, 120h
2. MOV AX, [DI]
3. XOR AX, 5132h
4. MOV [DI], AX
5. ADD DI, 2h
6. CMP DI, 2500h
7. JNE 3
8.  WJSVTPBMZPL
9.  NAADJGNANW
...
```

The decryption routine stays the same. Only the key(s) change.

The encrypted body changes.

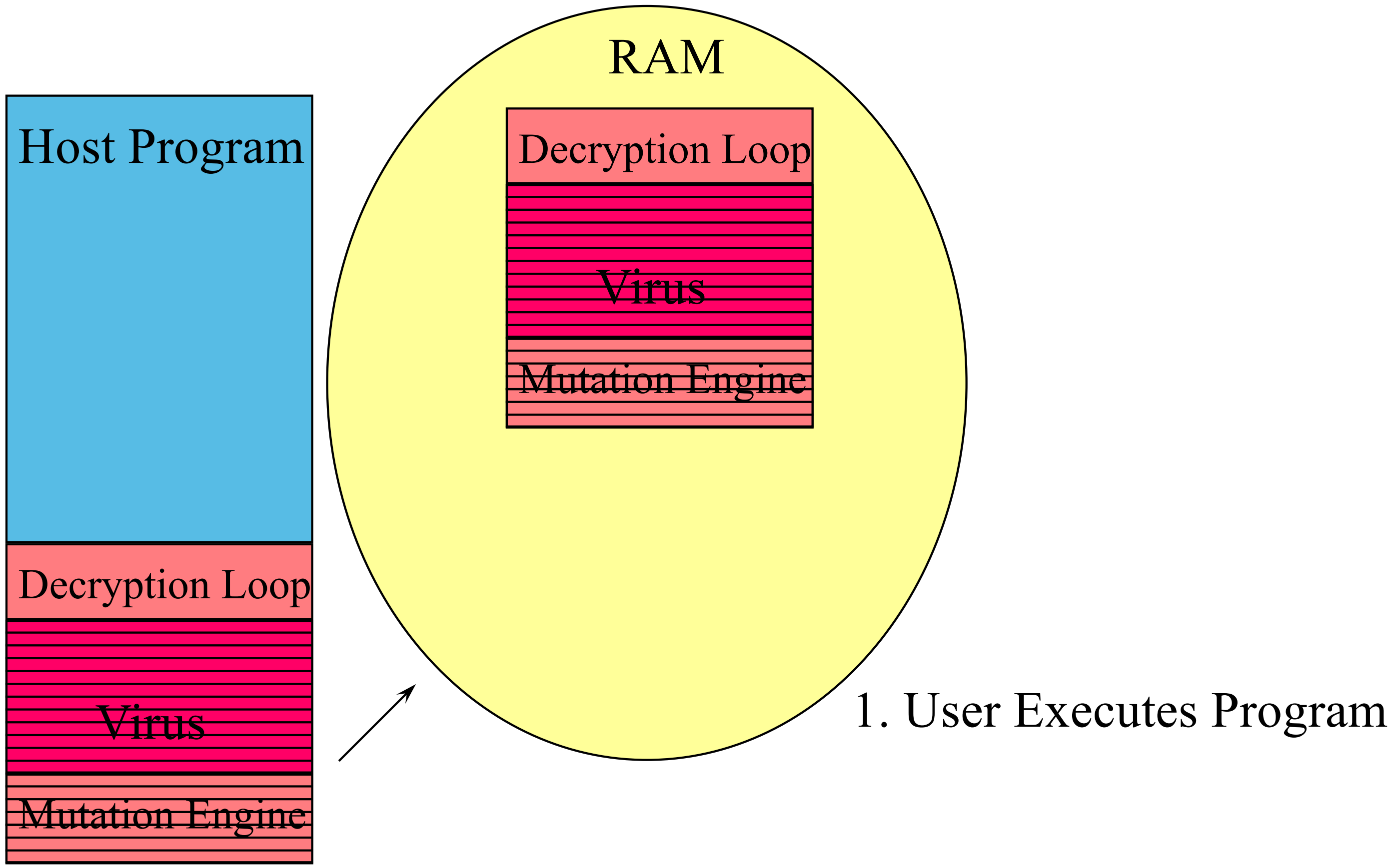
```
1. MOV DI, 120h
2. MOV AX, [DI]
3. XOR AX, 0030h
4. MOV [DI], AX
5. ADD DI, 2h
6. CMP DI, 2500h
7. JNE 3
8.  PKEPAJHENZAW
9.  MNANTPOOTIZN
...
```

Still easy to detect because the **decryption loop stays the same.**
***Virus signature = decryption code***

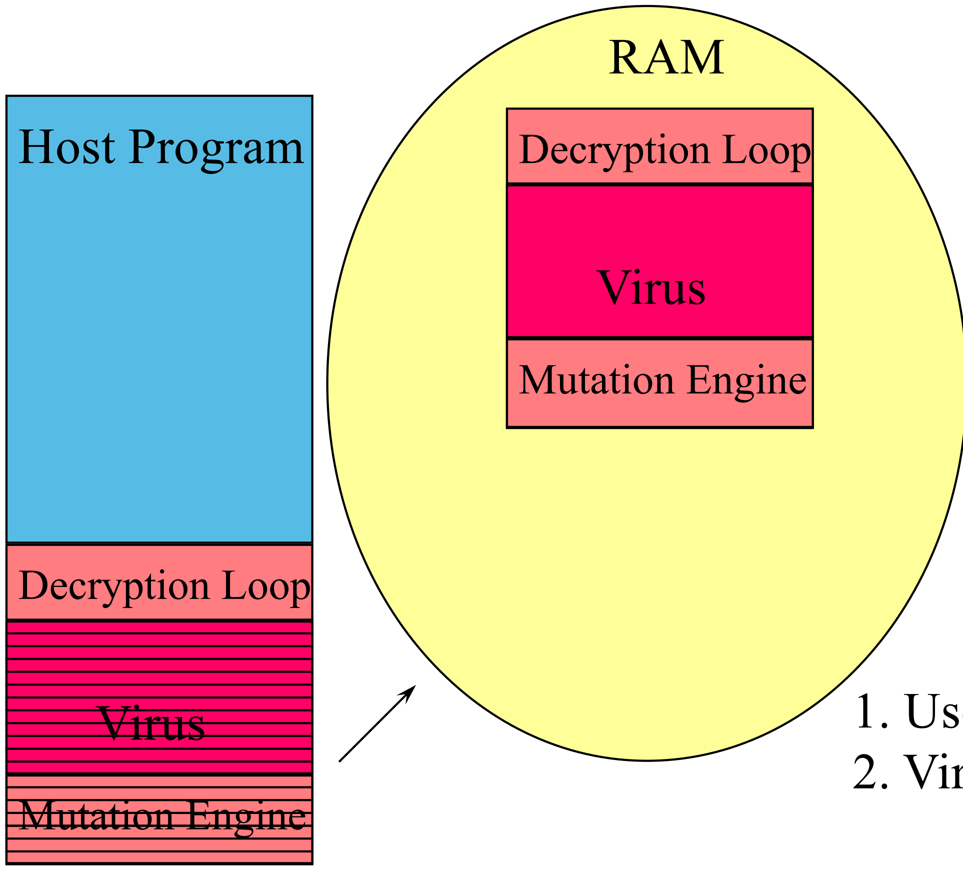
# The Polymorphic Virus

- Take this idea to the next step…
- Polymorphic viruses are self-encrypting viruses with a changing decryption ***algorithm***
- When infecting a new file, such a virus:
  - Generates **brand-new decryption code** from scratch
  - Encrypts a copy of itself using a complementary encryption algorithm
  - Inserts both the new decryption code and the encrypted body of the virus into target file

# The Polymorphic Virus

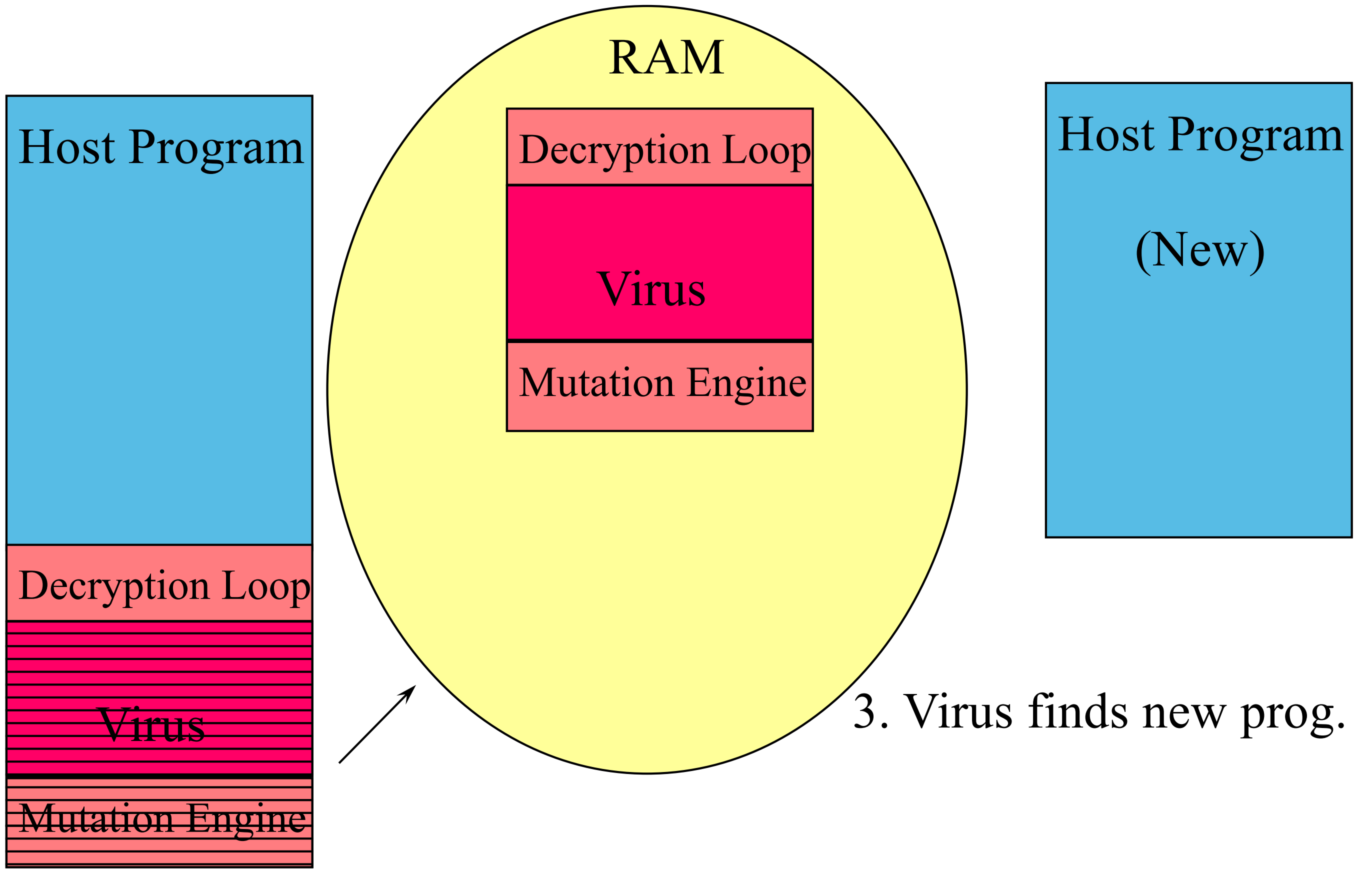


# The Polymorphic Virus

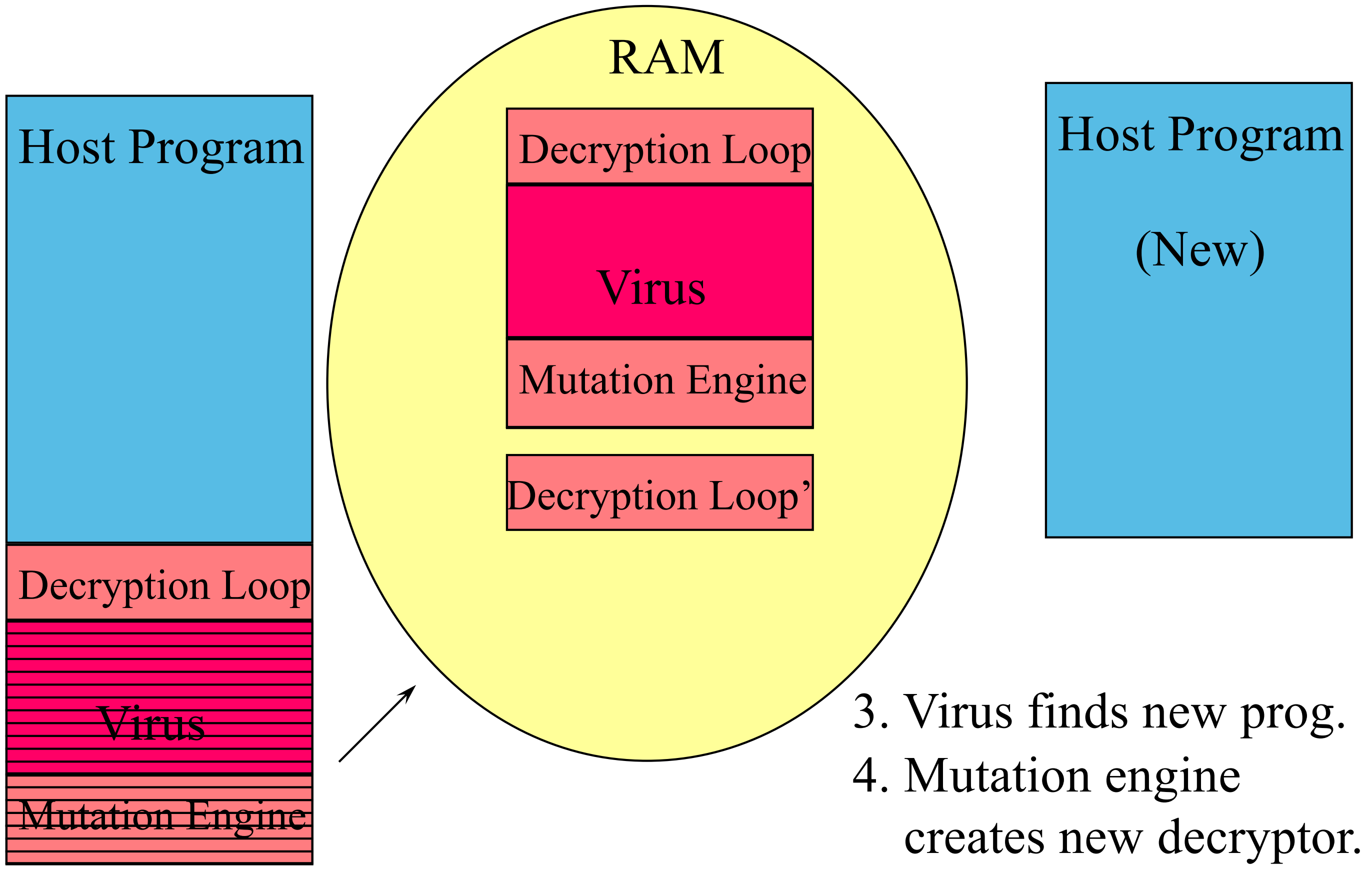



1. User Executes Program
2. Virus Decrypts Itself

# The Polymorphic Virus

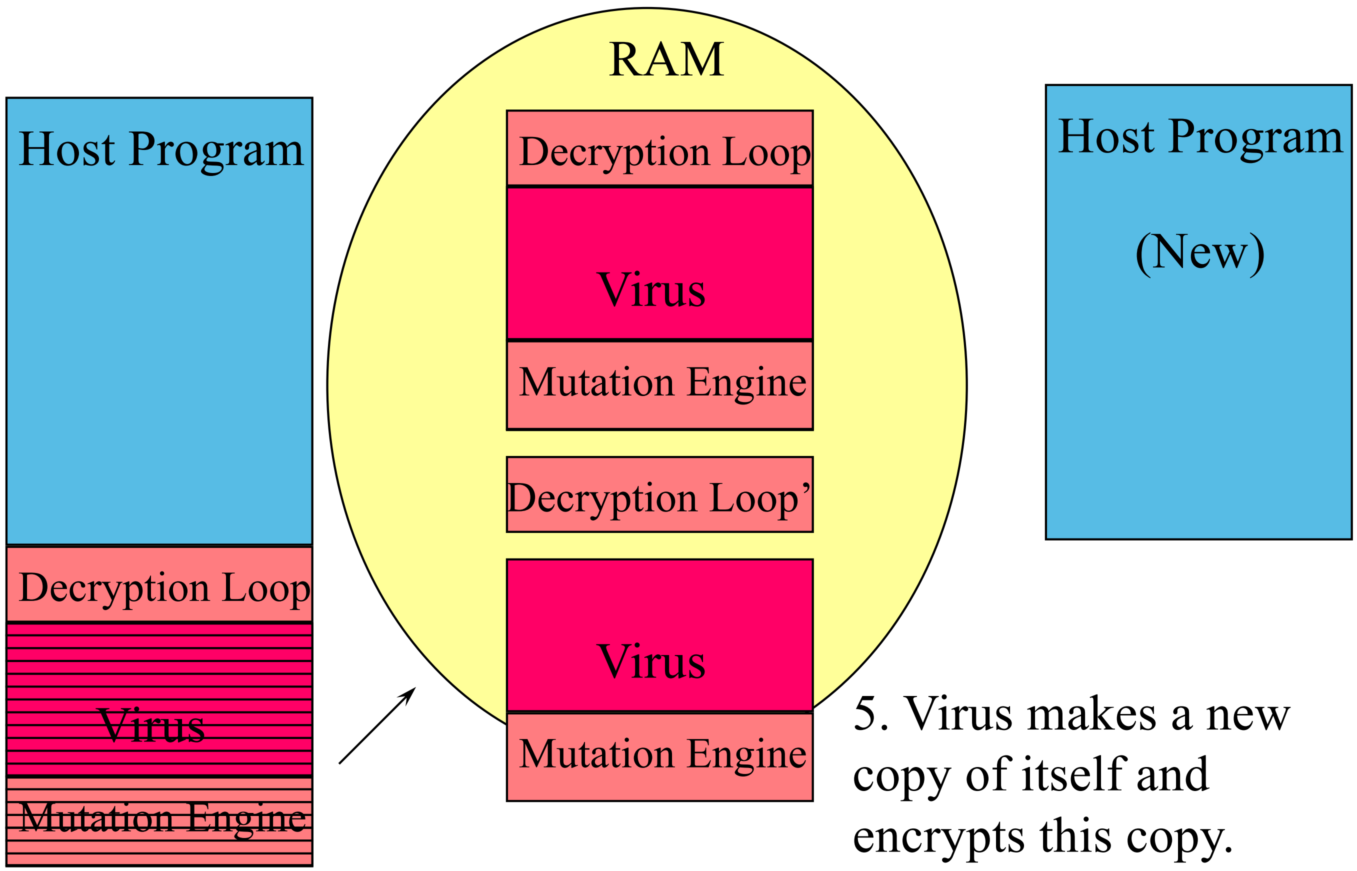


# The Polymorphic Virus



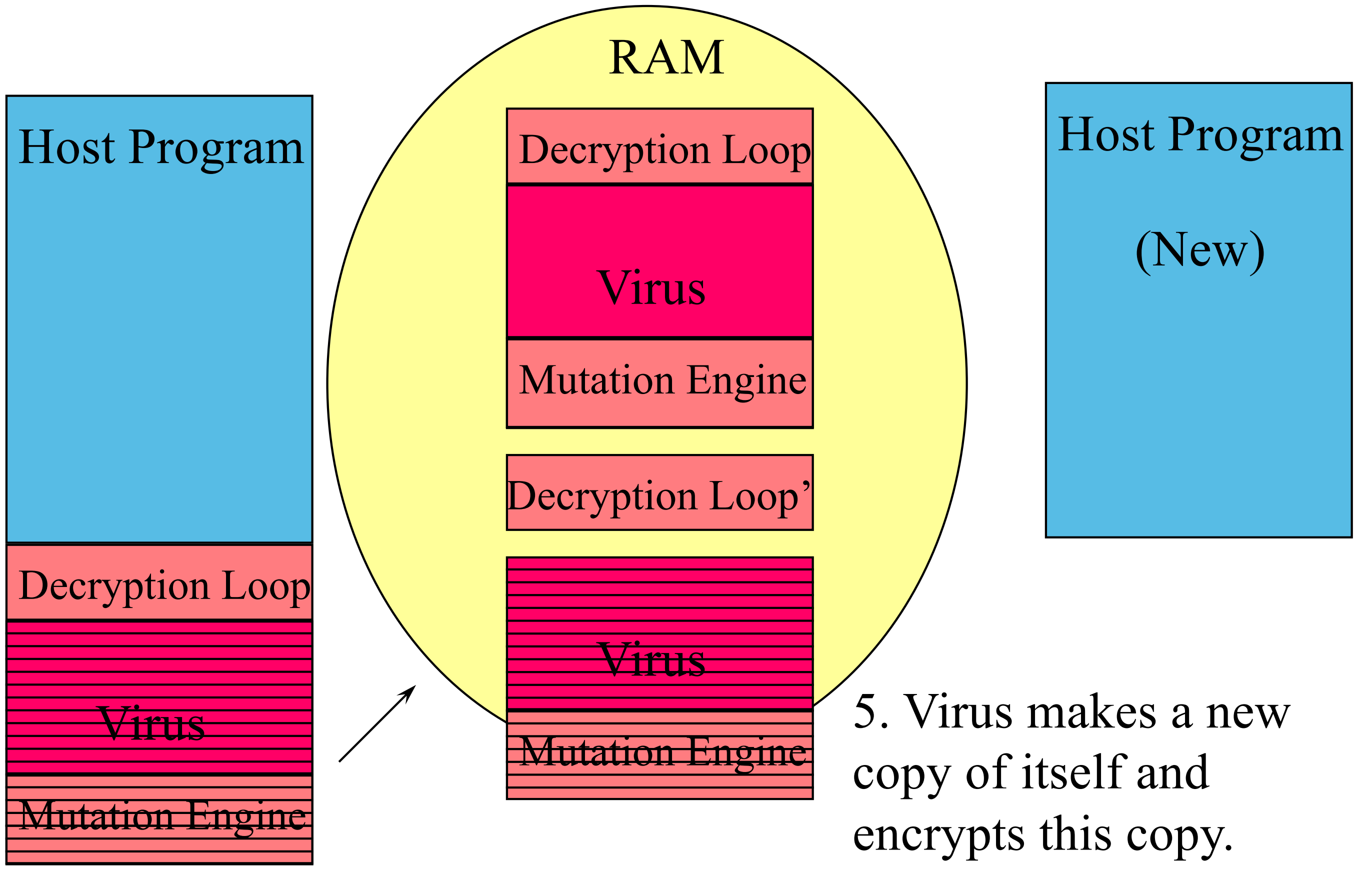

3. Virus finds new prog.
4. Mutation engine
   creates new decryptor.

# The Polymorphic Virus



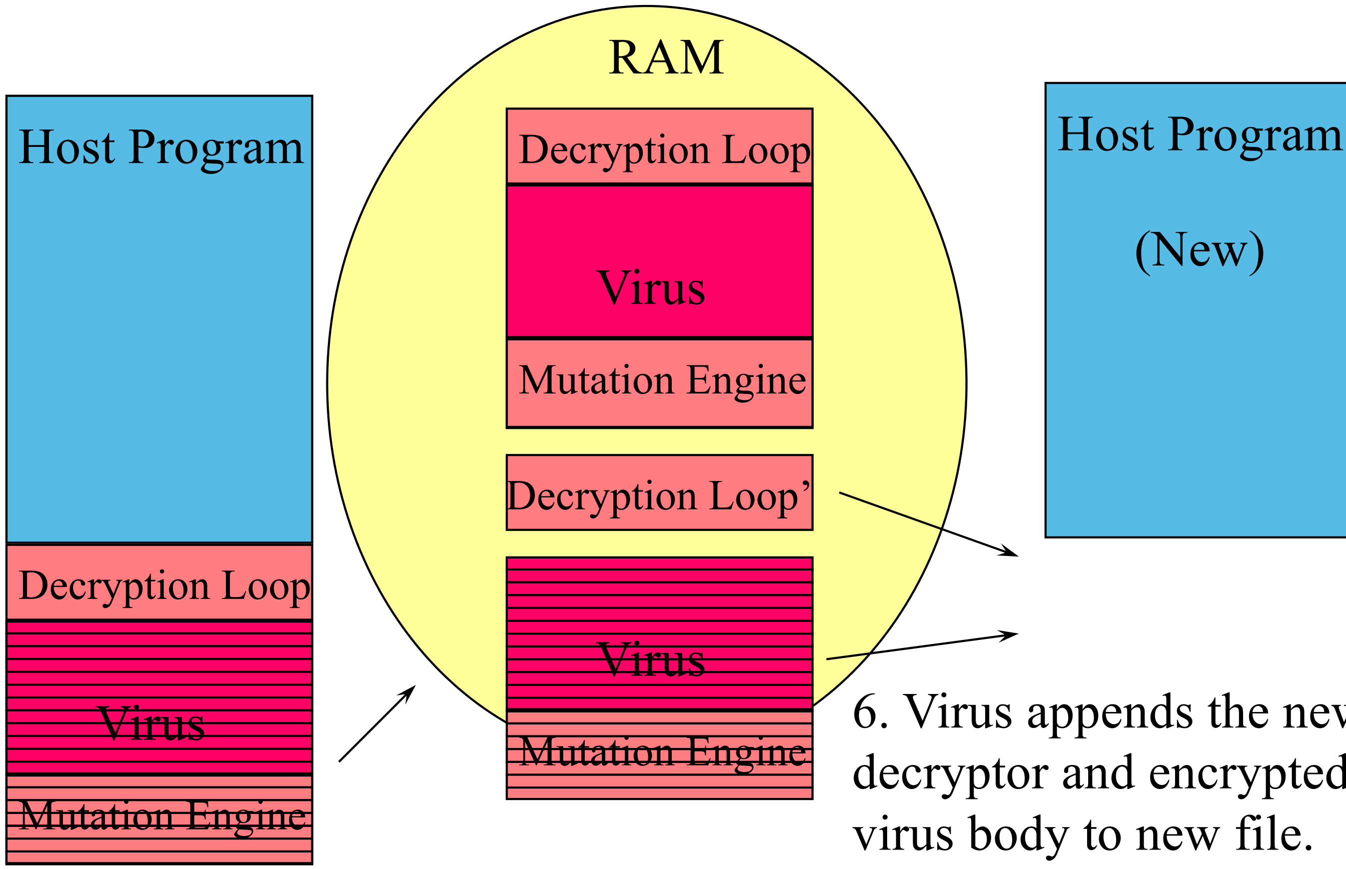

5. Virus makes a new copy of itself and encrypts this copy.

# The Polymorphic Virus



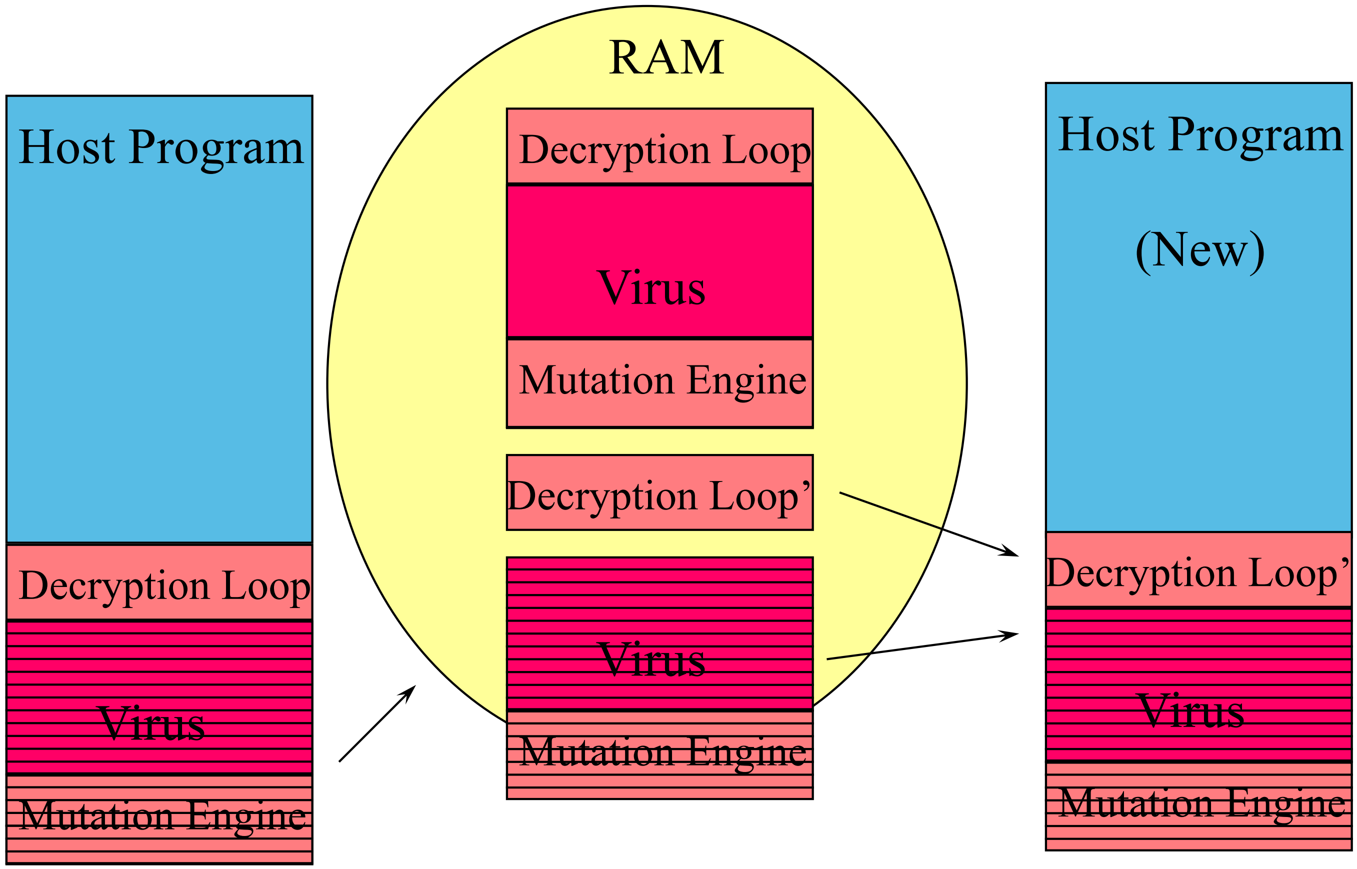

5. Virus makes a new copy of itself and encrypts this copy.

# The Polymorphic Virus



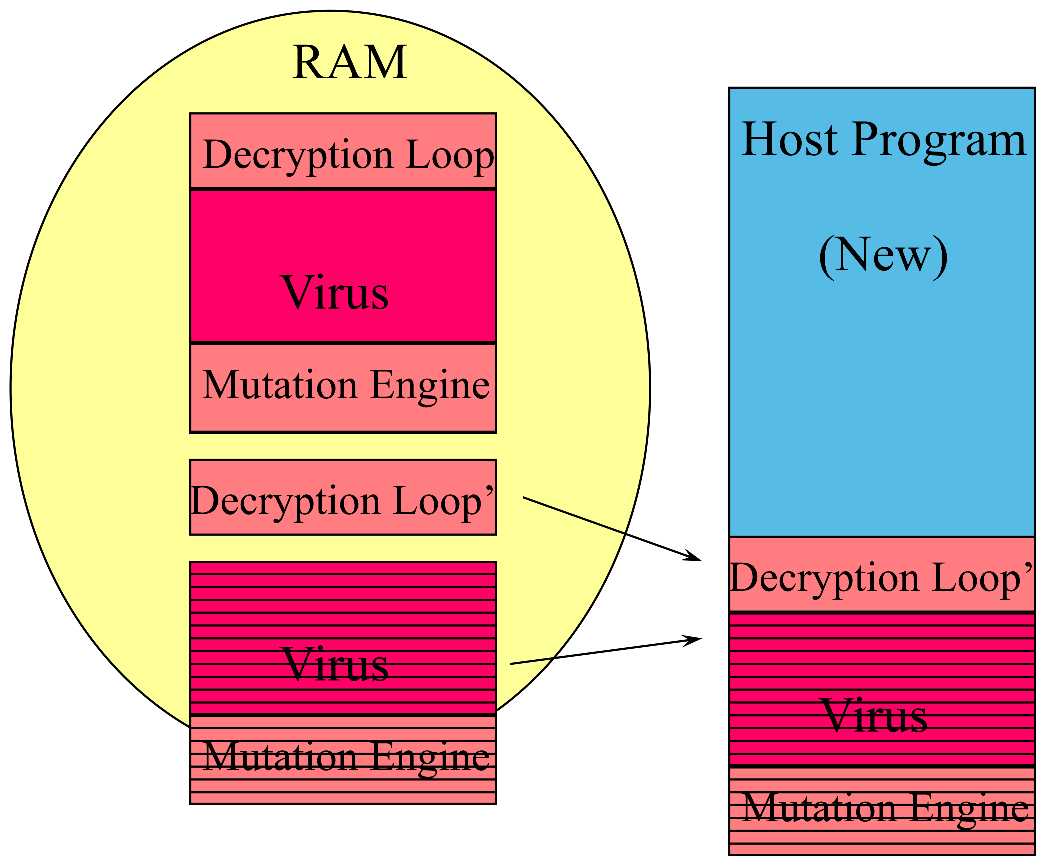

6. Virus appends the new decryptor and encrypted virus body to new file.

# The Polymorphic Virus



# The Polymorphic Virus

And we have a new infection!



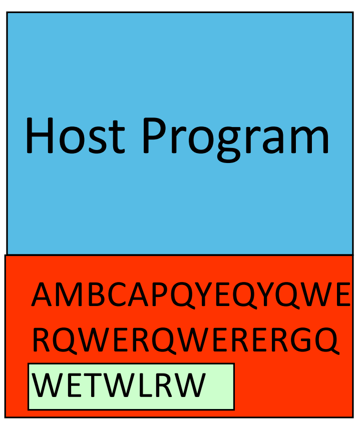
# Polymorphic malware: Extremely difficult to detect…

- May be no shared unencrypted code between two malware samples of the same virus
- Heuristics
  - Xray scanning: guess the key assuming you know plaintext of virus body (works only for simple schemes like XOR)
  - Try to symbolically analyze semantics of what code does (blows up quickly)
- **Generic decryption**
  - Key idea: let virus do the hard decryption work for you
    - ***Emulate*** code execution until the virus decrypts itself
      - Typically use some sort of virtual machine (VM) environment
    - Search for signatures in memory
  - **Assumptions**
    - Virus gains control of the host immediately
    - Virus decrypts itself deterministically
    - Virus has some static body that can be detected with traditional signatures

# Xray scanning (useful hack)

Assume the file is infected and perform a plain-text attack of the encrypted virus code. Slide plain-text code across file and test…
*This only works for simple schemes like byte-wise XOR*

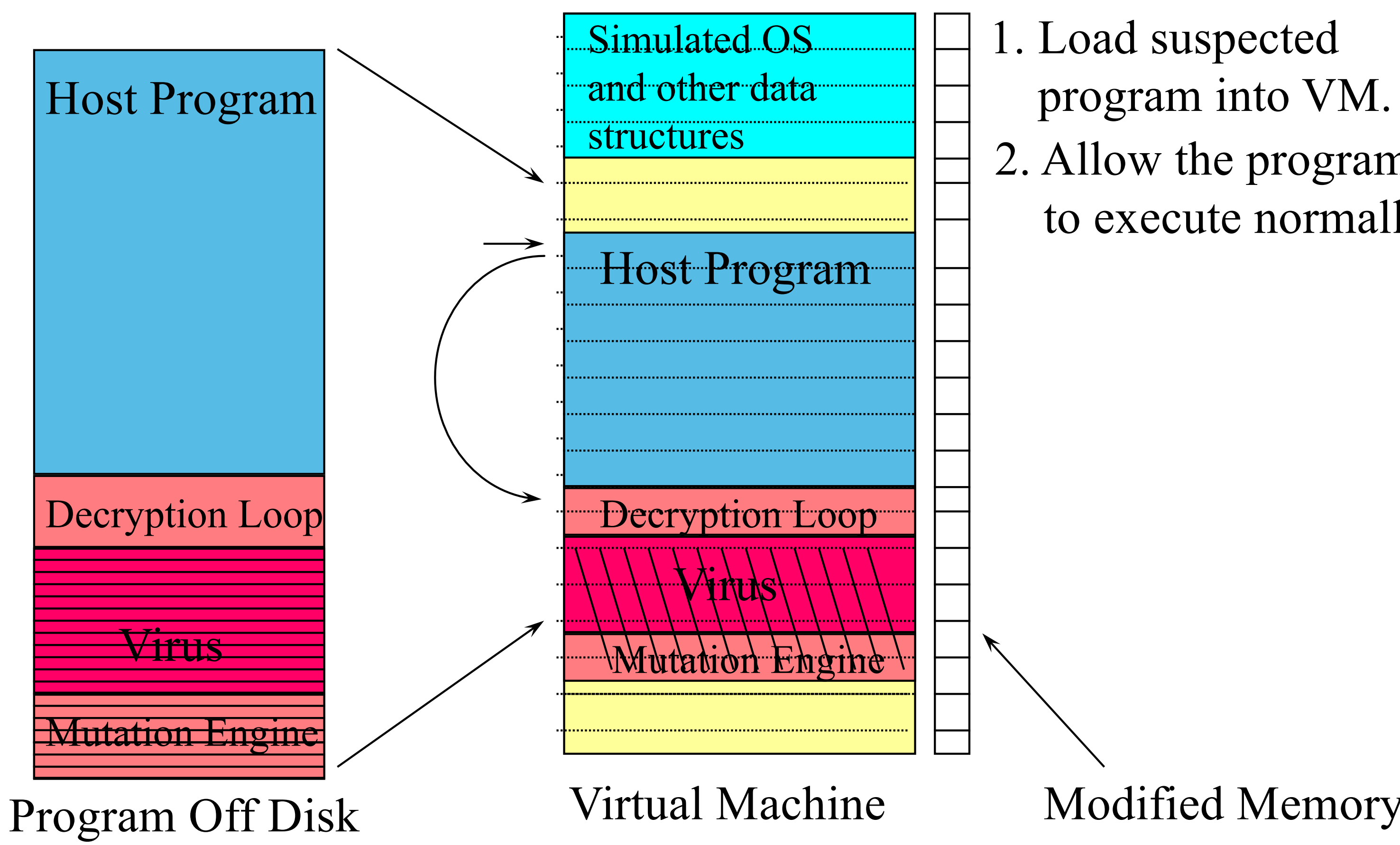
| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Scanned file: | 60 | 5C | 5D | 47 | 14 | 5D | 47 | 14-55 | 14 | 40 | 51 | 47 | 40 |
| | ⊕ | ⊕ | ⊕ | ⊕ | | | | | | | | | |
| Virus plain-text: | 54 | 68 | 69 | 73 | 20 | 69 | 73 | 20-61 | 20 | 74 | 65 | 73 | 74 |
| | 34 | 34 | 34 | 34 | | | | | | | | | |




The key must be *34*!

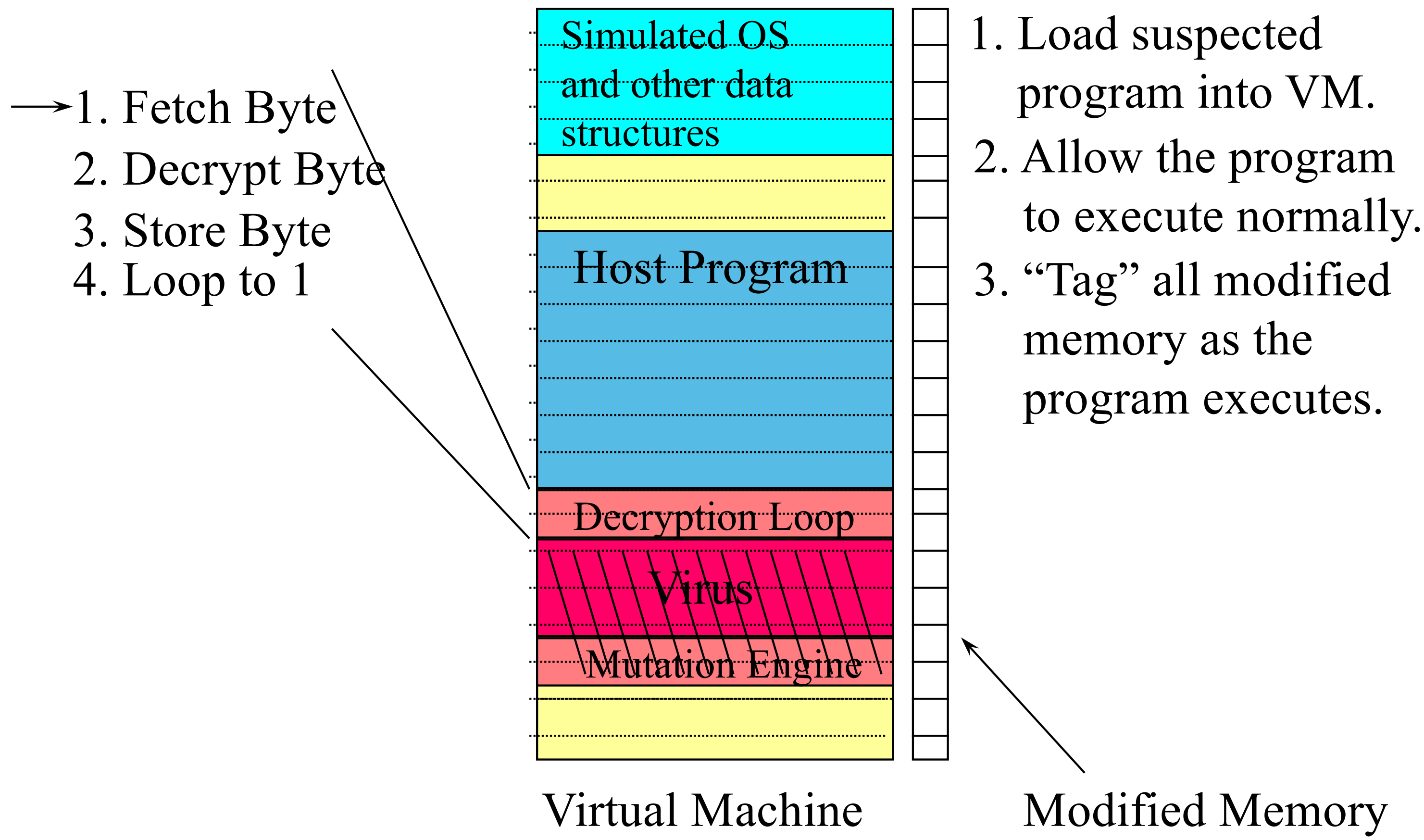
7 bytes from EOF = “VIRUS”?

# Generic decryption

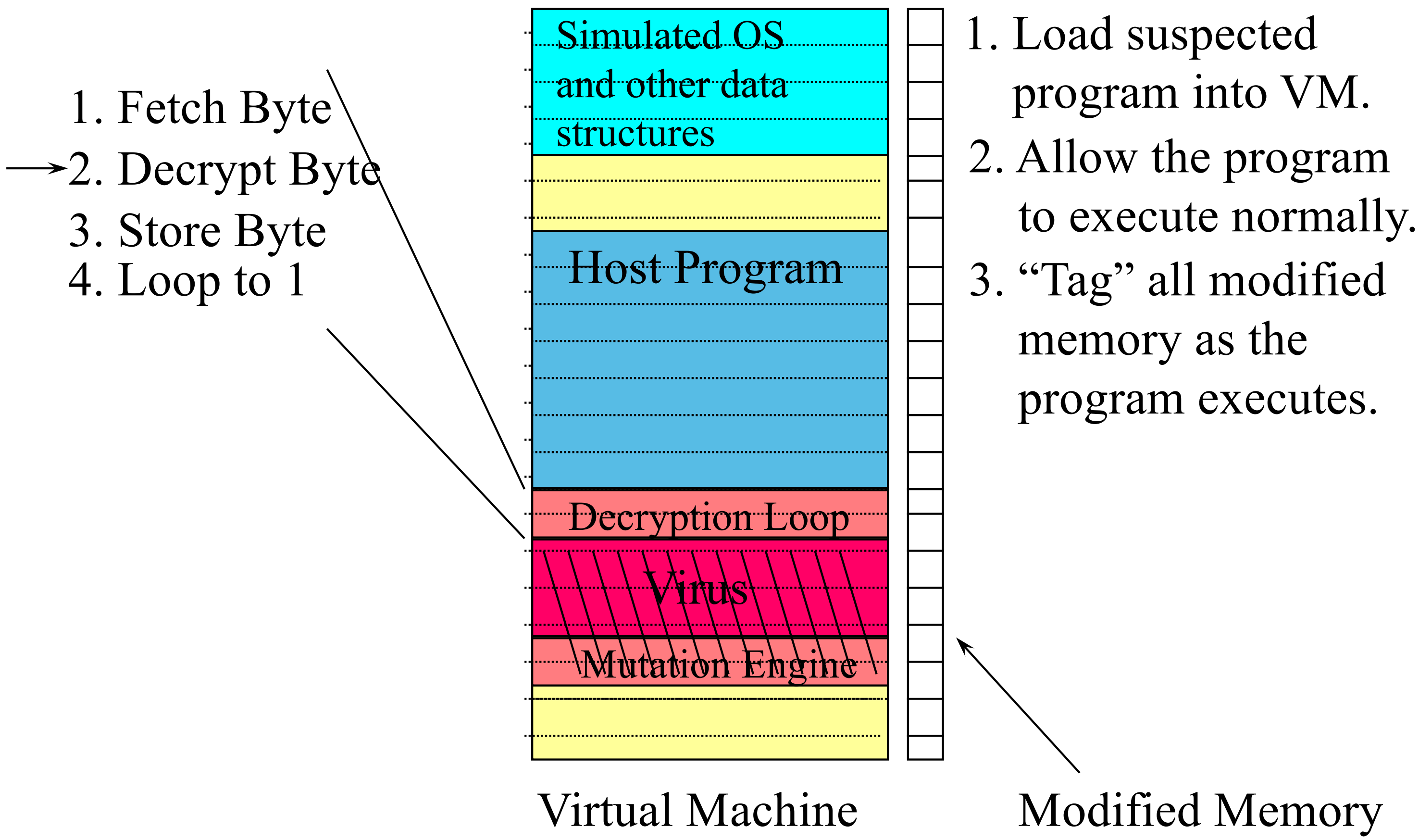



1. Load suspected program into VM.
2. Allow the program to execute normally.

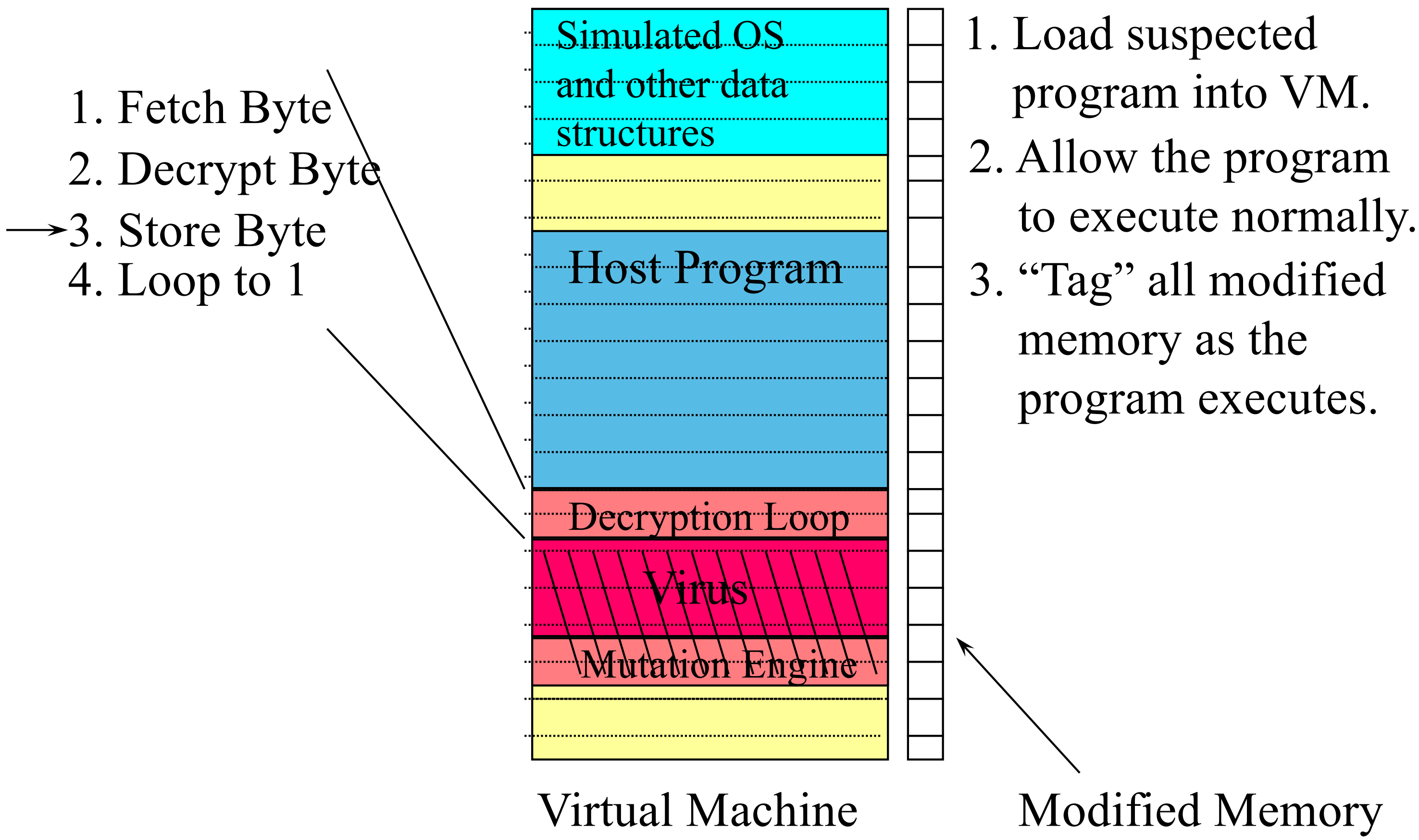
# Generic decryption



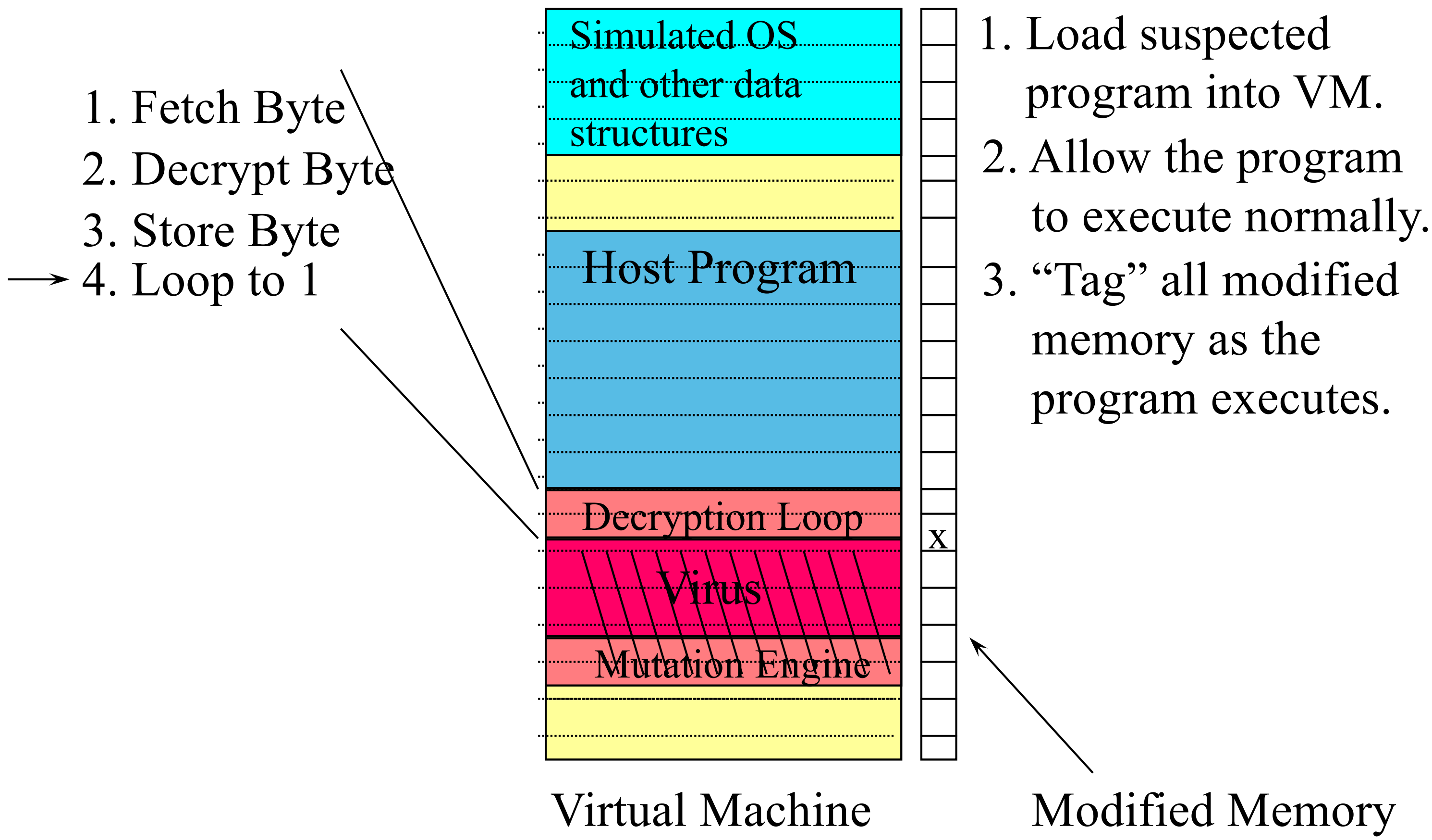
# Generic decryption



# Generic decryption



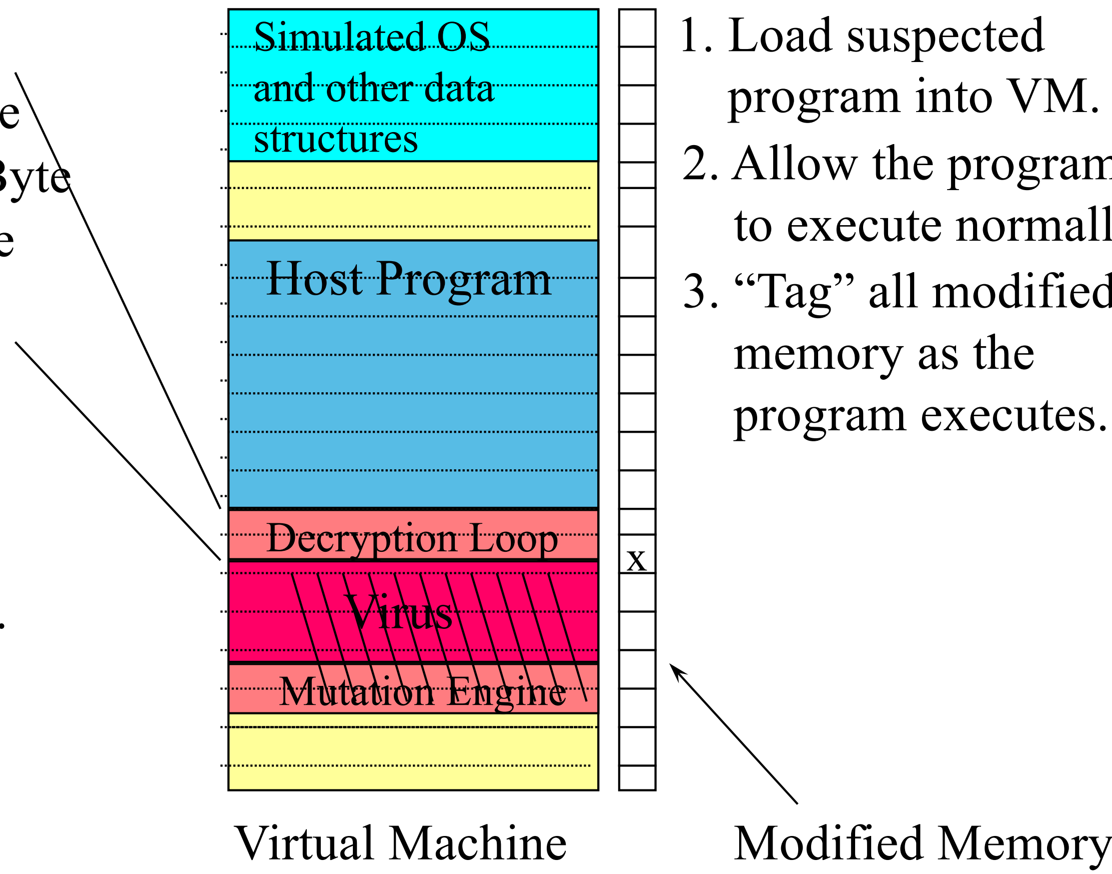
# Generic decryption



# Generic decryption

1. Fetch Byte
2. Decrypt Byte
3. Store Byte
4. Loop to 1

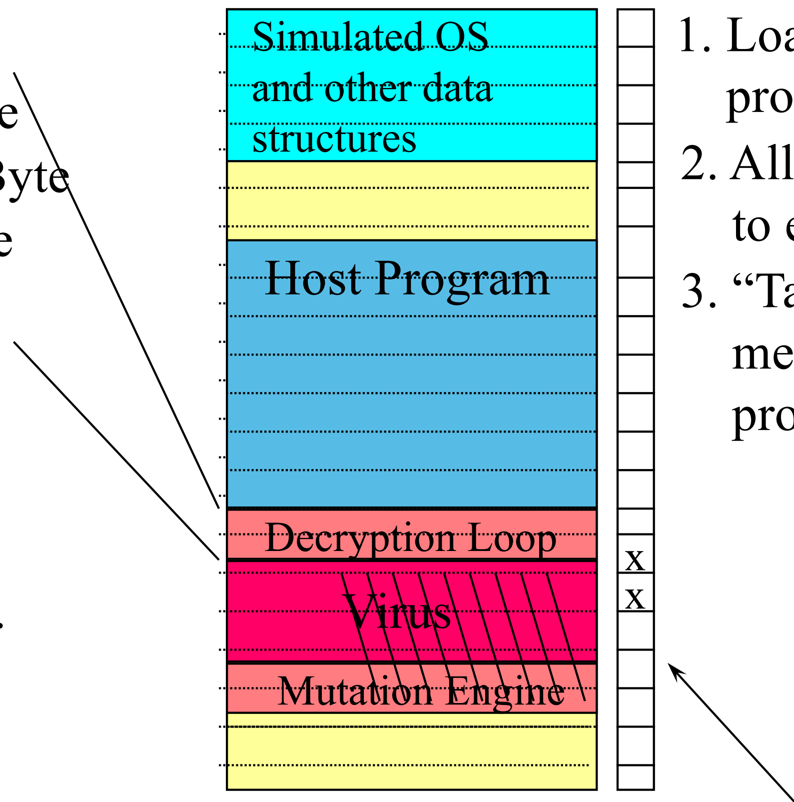
And on it goes...




1. Load suspected program into VM.
2. Allow the program to execute normally.
3. “Tag” all modified memory as the program executes.

# Generic decryption

1. Fetch Byte
2. Decrypt Byte
3. Store Byte
4. Loop to 1




Virtual Machine

Modified Memory

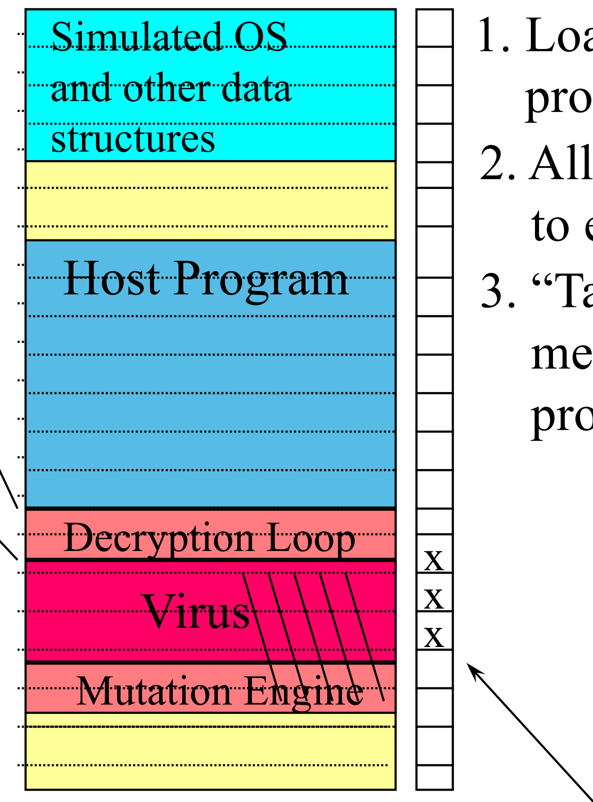
1. Load suspected program into VM.
2. Allow the program to execute normally.
3. “Tag” all modified memory as the program executes.

And on it goes...

# Generic decryption

1. Fetch Byte
2. Decrypt Byte
3. Store Byte
4. Loop to 1

And on it goes...




Virtual Machine

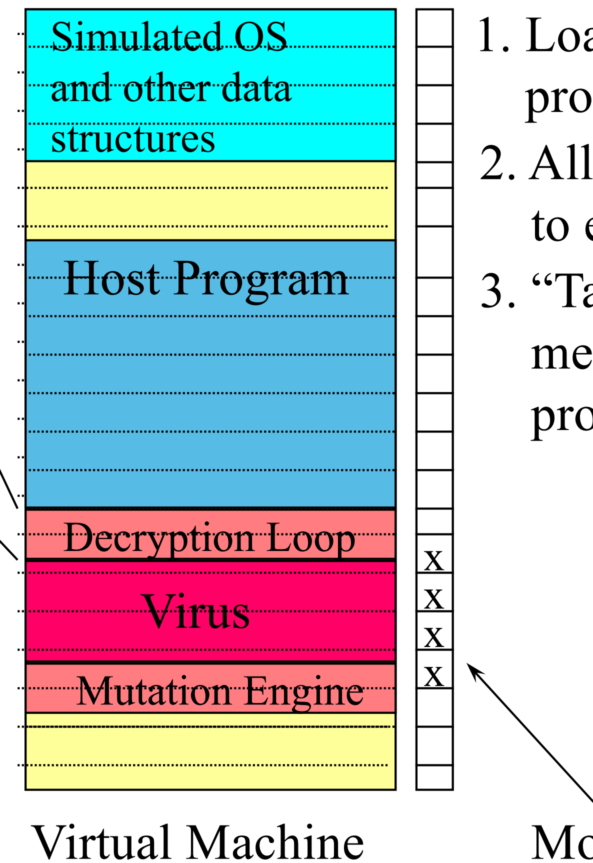
Modified Memory

1. Load suspected program into VM.
2. Allow the program to execute normally.
3. "Tag" all modified memory as the program executes.

# Generic decryption

1. Fetch Byte
2. Decrypt Byte
3. Store Byte
4. Loop to 1

And on it goes...

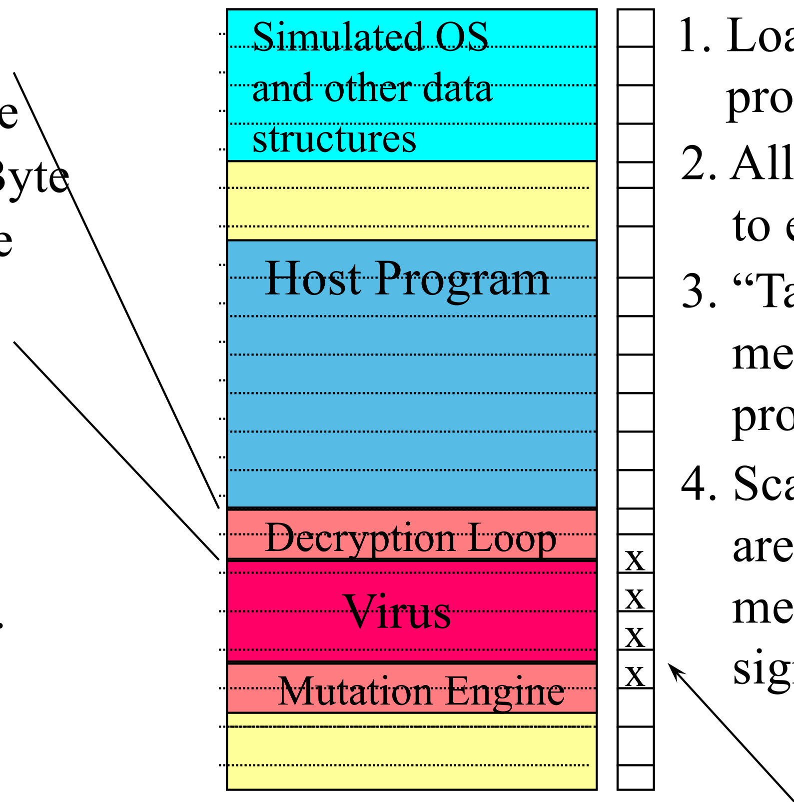



1. Load suspected program into VM.
2. Allow the program to execute normally.
3. “Tag” all modified memory as the program executes.

# Generic decryption

1. Fetch Byte
2. Decrypt Byte
3. Store Byte
4. Loop to 1

And on it goes...



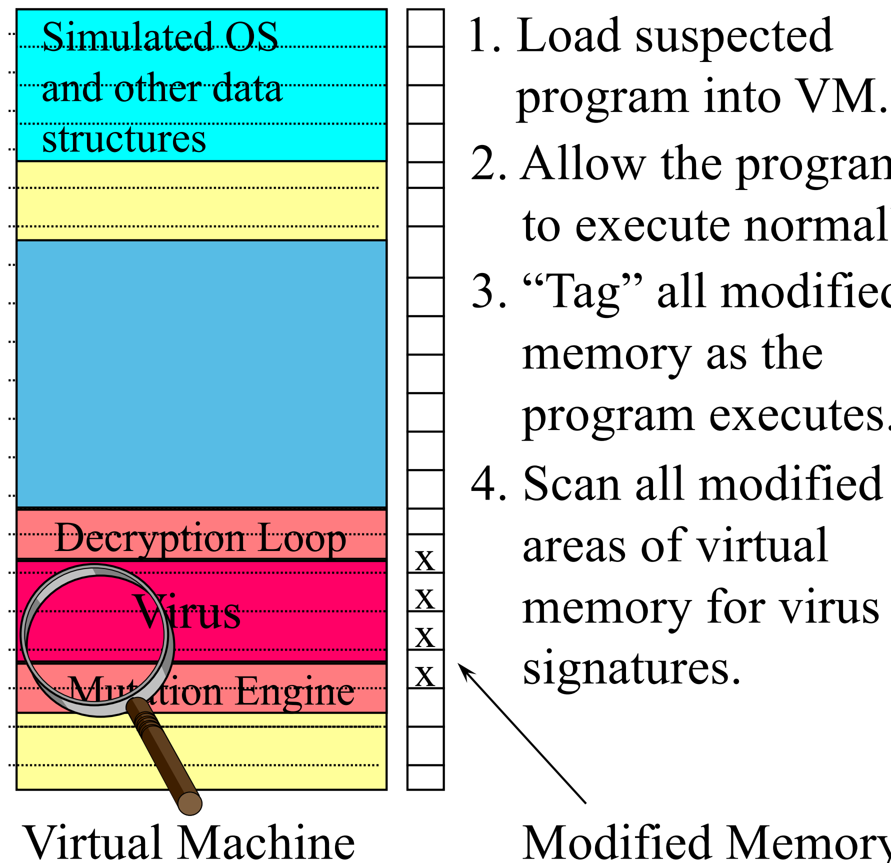

Virtual Machine

1. Load suspected program into VM.
2. Allow the program to execute normally.
3. "Tag" all modified memory as the program executes.
4. Scan all modified areas of virtual memory for virus signatures.

Modified Memory

# Generic decryption

KILL KILL KILL




1. Load suspected program into VM.
2. Allow the program to execute normally.
3. “Tag” all modified memory as the program executes.
4. Scan all modified areas of virtual memory for virus signatures.

# But many problems left…

- How long to emulate program?
  - Emulate too long and the system slows to a crawl
  - Don’t emulate enough and you might miss the virus
- What if malware can tell its running inside a VM?
  - E.g., don’t decrypt if you are
- What about malware that only activates with some specific input? Specific time?
- What if it doesn’t have a signature…

# The Metamorphic Virus

These viruses rewrite their logic in each new infection!
They have no byte-level fingerprint *anywhere*!

Metamorphic viruses use the current infection's code as a *template* and then *expand and contract sets of instructions* within the body to create a child infection.

# But many problems left…

- How long to emulate program?
  - Emulate too long and the system slows to a crawl
  - Don’t emulate enough and you might miss the virus
- What if malware can tell its running inside a VM?
  - E.g., don’t decrypt if you are
- What about malware that only activates with some specific input? Specific time?
- What if it doesn’t have a signature…
- Bottom line: detection is complex and malware authors constantly work to make it harder to do host-based malware detection
- Key assumptions of signature-based anti-malware software:
  - Malware is known a priori (i.e., there are good signatures that can be extracted)
  - Malware is used again (i.e., that discovering new malware instance is useful)
  - Malware signatures are widely distributed (cost/benefit)

# Detecting Viruses

- Scanning (signatures)

- **Integrity checking (check if file has changed)**
  - Keep “known good” hash of existing executables (whitelist); validate programs on computer against whitelist
- **Behavior (heuristic) detection**
  - E.g. does software use system features atypical of an application program; make anomalous network access; try to read sensitive files, etc…

# Integrity checking

- Change detection (e.g. Tripwire)
  - Assume programs are good when they are first installed
  - Take one-way hash of program in installed state
  - Periodically recheck hash to ensure it hasn't changed
- Whitelisting
  - Import list of "known good" software (one-way hashes)
  - Validate that all programs on disk hash to something on the "known good" list
- General issues
  - Hash list must be well-protected
  - Hash list must be comprehensive and kept up to date (white listing)
  - Doesn't deal well with editable documents (e.g., Word, Excel)
  - Note: most modern anti-virus systems will send the vendor hashes and filenames of every program you run on your machine

# Behavioral detection

- Identify suspicious behaviors in software
  - Can be decrypting code in memory
  - Unusual instruction sequences
  - Unusual use of file system or network interfaces (e.g., sending copy of code)
- Software reputation
  - Where did program get downloaded from? Have other people run it too? Do they tend do get infected a lot?
  - Do filename, libraries, compile, symbols, etc… correlate with past malware?
- Can run in real-time, amenable to machine-learning approaches
- Issues
  - Suspicious doesn't mean malicious; false positives
  - Forced to tune for low FP

# Disinfection

- Ok, you found a virus in a file… now what?
- **Standard disinfection**
  - Virus saves the beginning of the file it overwrites (for control transfer) so it can correctly execute it later
  - To clean: find virus, find original host file beginning, find size of virus. Now move original code to beginning, and truncate file to eliminate virus code
  - Specialized to each virus – only worth effort for really popular ones
- **Generic disinfection**
  - Run program and emulate until it restores the file to its normal state (so it can execute normally); let the virus itself do the tough work
  - Rewrite cleaned program back to disk
  - Works with majority of viruses
  - Problems: viruses that overwrite code, viruses with unknown entry points, viruses not well modeled by heuristics – when is image clean?)

# Today, not so many “viruses”

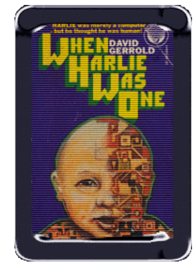
- Why? File sharing isn’t the best vector for replication
- What is? The Internet

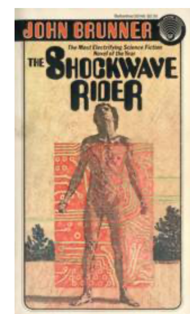
# Quick aside: why is self-replication interesting?

- Because it allows massive compromise for low investment in resources
- Some worms have taken over hundreds of thousands of hosts in a day; others have covered the **entire Internet in 10 minutes**

# A brief history of worms…

- As always Sci-Fi authors get it right first
  - Gerold's "*When H.A.R.L.I.E. was One*" (1972) – "Virus"
  - Brunner's "Shockwave Rider" (1975) – "tapeworm program"
- Shoch&Hupp co-opt idea; coin term "worm" (1982)
  - Key idea: programs that self-propagate through network to accomplish some task; benign
- Fred Cohen demonstrates power and threat of self-replicating viruses (1984)
- First significant worm in the wild: Morris worm (1988)



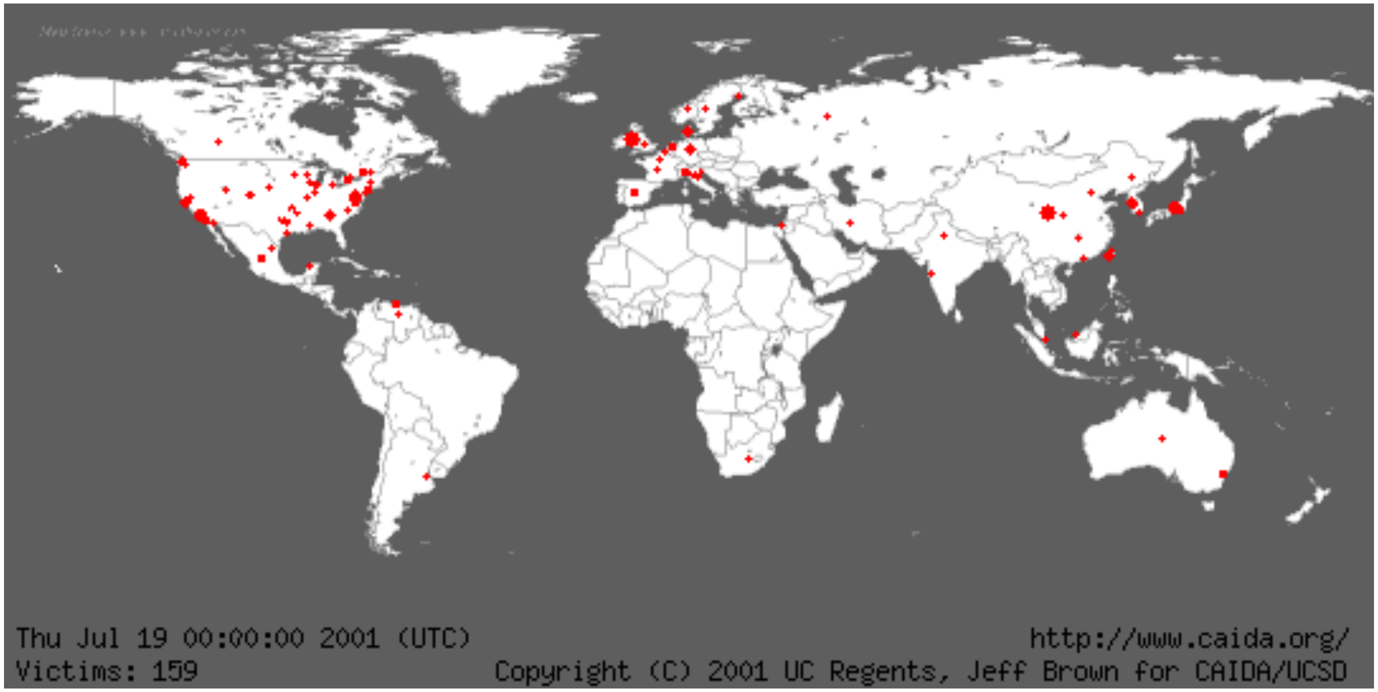
# History: Morris Internet Worm

- November 2, 1988
- Operation
  - Buffer overflow in fingerd
  - DEBUG mode left enabled in sendmail (enabled shelling out)
  - Dictionary attacks on /etc/password
  - Infected around 6,000 major Unix machines
- Shutdown big chunks of the Internet and e-mail
- Cost of the damage estimated at $10m - $100m
- Robert T. Morris Jr. unleashed Internet worm
  - Graduate student at Cornell University
  - Convicted in 1990 of violating Computer Fraud and Abuse Act (CFAA)
  - $10,000 fine, 3 yr. Suspended jail sentence, 400 hours of community service
  - Today he's a professor at MIT (and a great guy I might add)

# The Modern Worm era

- Email based worms in late 90's (Melissa & ILoveYou)
  - Infect >1M hosts, but requires user participation
- **CodeRed** worm released in Summer 2001
  - Exploited buffer overflow in IIS; no user interaction
  - Uniform random target selection (after fixed bug in CRv1)
  - Infects 360,000 hosts in 10 hours (CRv2)
  - Like the energizer bunny… still going years later

# Code Red worm



# The Modern Worm era

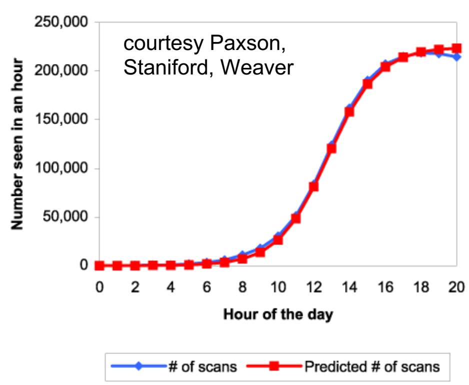
- Email based worms in late 90's (Melissa & ILoveYou)
  - Infect >1M hosts, but requires user participation
- **CodeRed** worm released in Summer 2001
  - Exploited buffer overflow in IIS; no user interaction
  - Uniform random target selection (after fixed bug in CRv1)
  - Infects 360,000 hosts in 10 hours (CRv2)
  - Like the energizer bunny… still going years later
- **Slammer** (2003)
  - Hits peak BW in 3mins (55M targets/sec)
  - Scans 90% of Internet in < 10mins
- Energizes **renaissance** in worm construction (1000's)
  - Exploit-based: CRII, Nimda, **Slammer**, Blaster, Witty, Conficker, etc…
  - Human-assisted: SoBig, NetSky, MyDoom, etc…

# How to think about network malware outbreaks

- Well described as infectious epidemics
  - Simplest model: Homogeneous random contacts
  - Aside: this is also the basics of how we model Covid-19 spreading
- Classic SI model
  - N: population size
  - S(t): susceptible hosts at time t
  - I(t): infected hosts at time t
  - ß: contact rate
  - i(t): I(t)/N, s(t): S(t)/N

$$\frac{dI}{dt} = \beta \frac{IS}{N}$$
$$\frac{dS}{dt} = -\beta \frac{IS}{N}$$
$$\Rightarrow \quad \frac{di}{dt} = \beta i(1-i)$$

$$i(t) = \frac{e^{\beta(t-T)}}{1+e^{\beta(t-T)}}$$



# Takeaway

- Two things matter when considering the scope of an outbreak

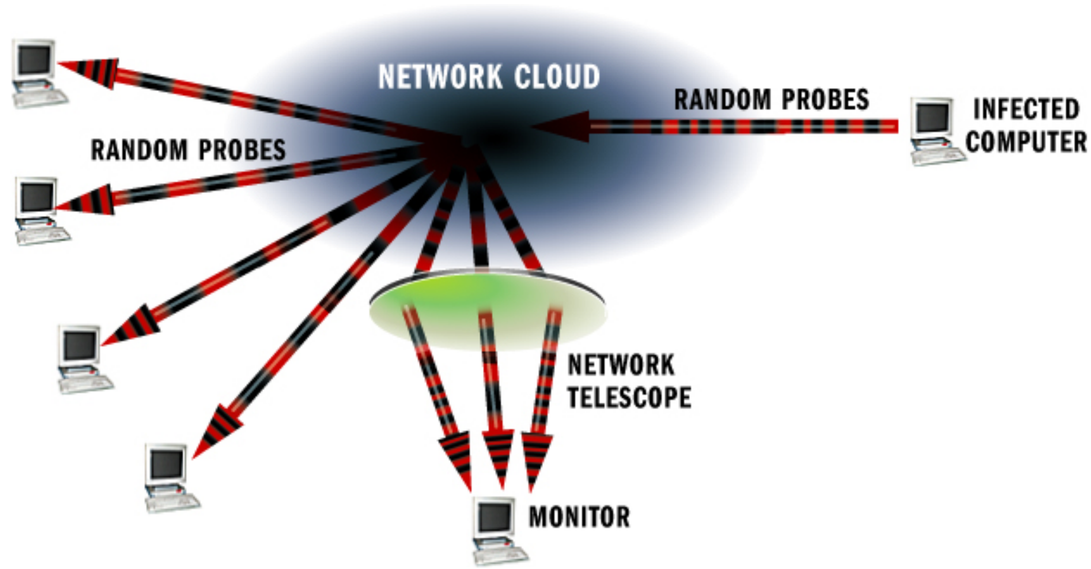
  - How **likely** is it that a given infection attempt is successful?
    - Vulnerability distribution (e.g. density – S(o)/N)
    - Target selection (can you be better than random?)

  - How **frequently** are infections attempted?
    - ß: Contact rate

# What can be done?

- Reduce the number of susceptible hosts
  - **Prevention**, reduce S(t) while I(t) is still small (ideally reduce S(o))
  - Basic software security (don't have bugs, patch the ones you have, etc)
  - In practice:
    - Turn on firewall, turn off unneeded network services, keep patches up to date
- Reduce the number of infected hosts
  - **Treatment**, reduce I(t) after the fact
  - Tends to be easy to detect infected hosts (spewing traffic to random destinations) but treatment is slow
  - Aside: white worms – illegal and problematic, but have been deployed

# Network Telescopes




- Network Telescope: monitor large range of **unused** IP addresses – If worm scans randomly, will hit telescope repeatedly
- Very scalable. UCSD monitored ~1% of all routable addresses

# Why do telescopes work?

- Assume worm spreads randomly
  - Picks 32bit IPv4 address at random and probes it
- Monitor block of *n* IP addresses
- If worm sends *m* probes/sec, we expect to see one within:

$$\frac{nm}{2^{32}} \sec$$

- If monitor receives R' probes per second, can estimate infected host is sending at:

$$R \geq R' \frac{2^{32}}{n}$$

# What can be done?

- Reduce the number of susceptible hosts
  - **Prevention**, reduce S(t) while I(t) is still small (ideally reduce S(0))
  - Basic software security (don't have bugs, patch the ones you have, etc)
  - In practice:
    - Turn on firewall, turn off unneeded network services, keep patches up to date
- Reduce the number of infected hosts
  - **Treatment**, reduce I(t) after the fact
  - Tends to be easy to detect infected hosts (spewing traffic to random destinations) but treatment is slow
  - Aside: white worms – illegal and problematic, but have been deployed
- Reduce the contact rate
  - **Containment**, reduce ß while I(t) is still small
  - Some network switches will rate limit sources that are sending to too many different destinations in a set time period

# Lots of other mechanisms for spreading malware

- Drive-by Downloads: vulnerability in Web browser
  - Drive traffic to Web site – spam, twitter bots, search engine abuse, ad fraud, etc
- Social engineering
  - E-mail/IM/Chat file attachments – "You'll never believe the photos from the office party!"
  - Add-ons – "To watch this video click here to install the latest codec"
  - Malicious apps, browser extensions, etc…
- File Sharing networks
  - Seed popular software (typically pirated or game cheats) and add malware to it

# So you’ve taken over 100,000 machines…

- Then what?
- Use machines *together* for some purpose
- Next-time: botnets