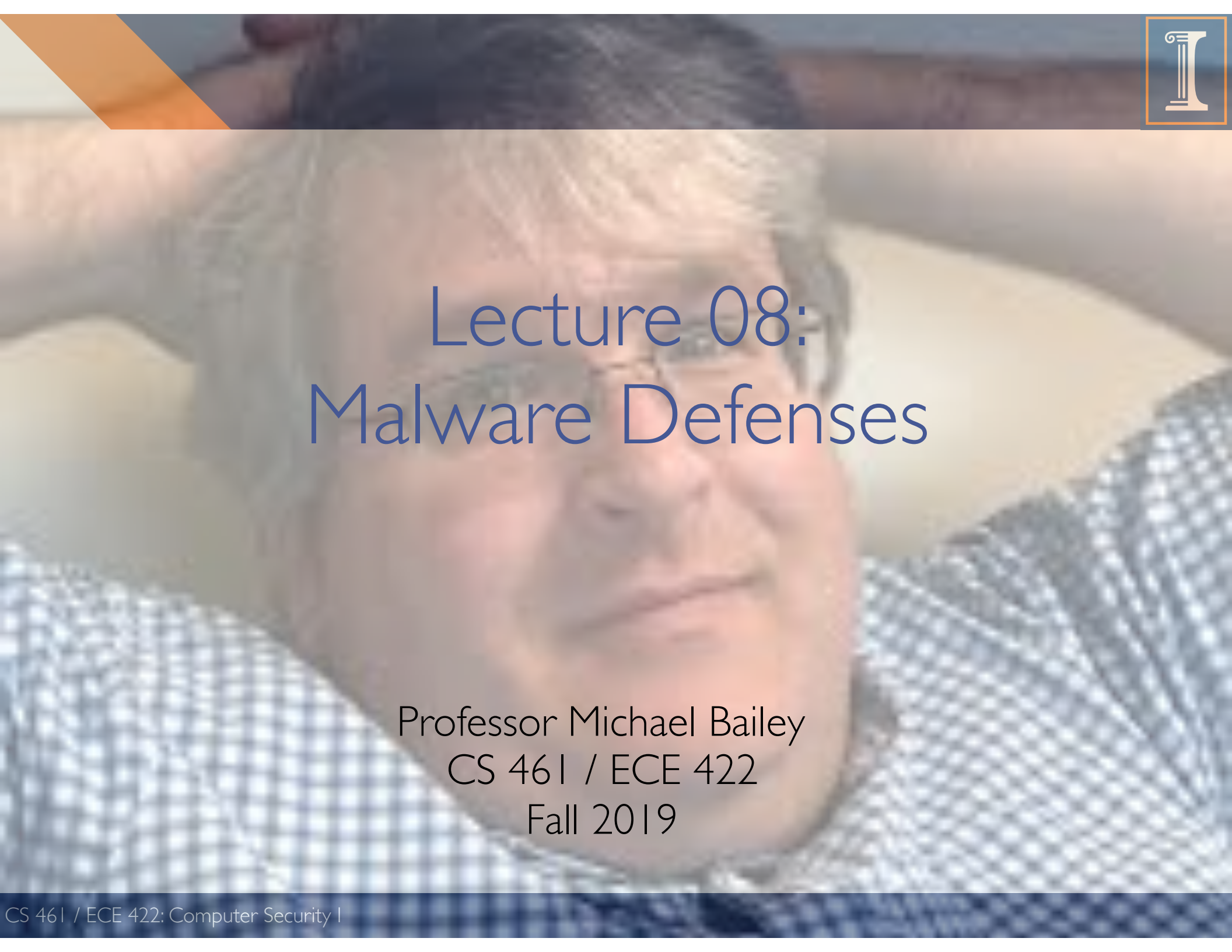


Lecture 08: Malware Defenses

Professor Michael Bailey

CS 461 / ECE 422

Fall 2019

Goals for Today

- Learning Objectives:
  - Understand how one typically defends against malware
  - Understand the approaches used by signature-based and anomaly-based system
  - Articulate the tradeoffs between these systems
  - Recognize or provide examples of signature-based and anomaly-based systems
- Announcements, etc:
  - Office hours all the time! M-F 5-7pm, 4405 Siebel
  - Prof Bates does not have office hours this week
  - MP1 is live!
    - Checkpoint #2: **Due Sept 18 at 6pm**

Reminder: Please put away devices at the start of class

What to do?

- Secure design, coding
- Automatically find and fix bugs (testing)
- Monitor a system at runtime to detect and prevent exploits of bugs (HIDS)
- Accept that programs will have bugs and design the system to minimize damages (isolation)

Software security flaws

- any stage of the software development lifecycle
  - Not identifying security requirements up front
  - Creating conceptual designs that have logic errors
  - Using poor coding practices
  - Deploying the software improperly
  - Introducing flaws during maintenance or updating
- have a scope beyond the software
  - The software and its associated information
  - The operating systems of the associated servers
  - The backend database
  - Other applications in a shared environment
  - The user's system
  - Other software that the user interacts with

Security by Design Principles

- Minimize attack surface area
- Establish secure defaults
- Principle of Least privilege
- Principle of Defense in depth
- Fail securely
- Don’t trust services
- Separation of duties
- Avoid security by obscurity
- Keep security simple
- Fix security issues correctly

Secure Coding Standards

```
void setfile(FILE *file) {
    myFile = file;
}
```

?

-or-

```
errno_t setfile(FILE *file) {
 if (file && !ferror(file) && !feof(file)) {
    myFile = file;
    return 0;
  }

  /* Error safety: leave myFile unchanged */
  return -1;
}
```

Functions should validate their parameters
Enforce type safety
Do not reuse variable names in subscopes
…

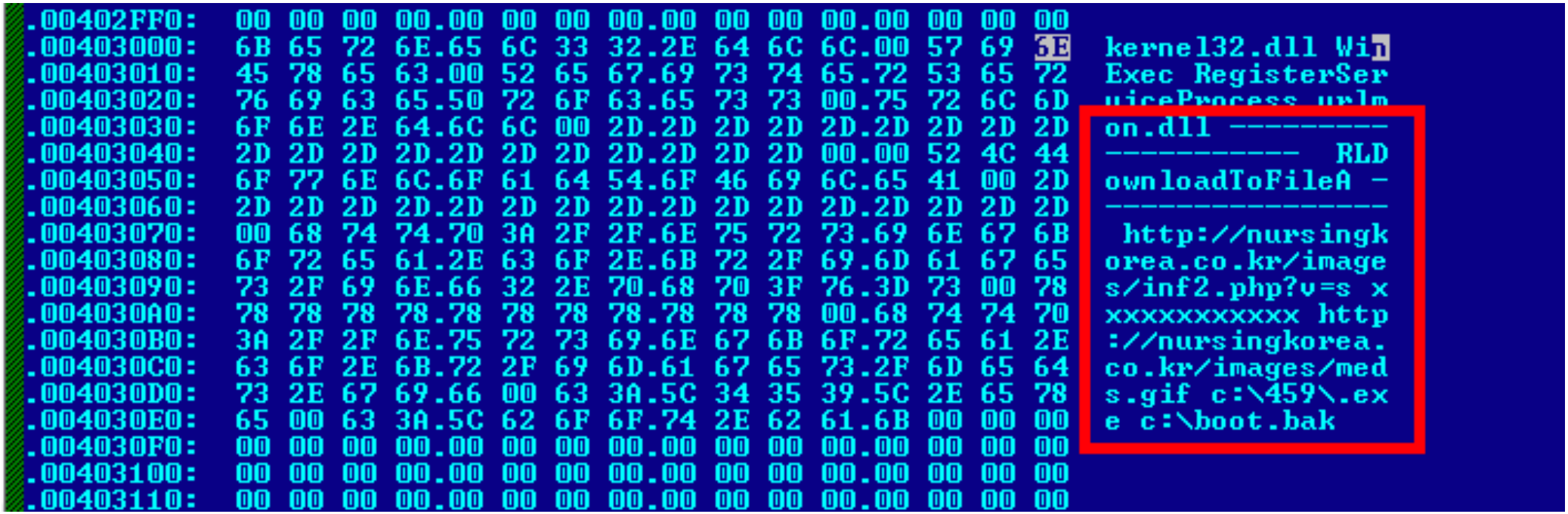
HIDS, AV

- Terminology
  - IDS: Intrusion detection system
  - IPS: Intrusion prevention system
  - HIDS/NIDS: Host/Network Based IDS
- Difference between IDS and IPS
  - Detection happens after the attack is conducted (i.e. the memory is already corrupted due to a buffer overflow attack)
  - Prevention stops the attack before it reaches the system (i.e. shield does packet filtering)
- Anomaly, heuristic, behavior-based vs. Misuse, Rule-based, Signature-based

Signatures: A Malware Countermeasure

- Scan compare the analyzed object with a database of signatures
- A signature is a virus fingerprint
  - E.g.,a string with a sequence of instructions specific for each virus
  - Different from a digital signature
- A file is infected if there is a signature inside its code
  - Fast pattern matching techniques to search for signatures
- All the signatures together create the malware database that usually is proprietary

Signatures: A Malware Countermeasure



Signatures: A Malware Countermeasure

Prodigy.268

CN: An overwriting, 268-byte direct infector containing the texts '*.COM', 'Pr0diGy VeEr0oZ (c) 1995' and 'HaPpY nEw YeAR! SeE U iN HeLL...'.

```
Prodigy.268          890E C301 8916 C101 BA00 01B4 40B9 0C01 CD21 90B8 0157 8B16
```

Saha.2382

CR: An overwriting, 2382-byte virus which infects COM files and modifies EXE files with the same name as COM targets. The virus appends nine bytes (the string ' Sahand ') to EXE programs.

```
Sahand.2382          B918 048D 1618 058B 1EBF 09B4 40CD 21B9 8F00 8D16 3009 8B1E
```

Simple.331

CN: An encrypted, appending, 331-byte, direct infector with the text '*.COM'. Infected programs ends with the string 'SIMPLE'.

```
Simple.331           60E8 0000 5E81 EE32 01B9 2E01 2EF6 1446 E2FA 61C3 5349 4D50
```

Spartak.1360

CEN: An encrypted, appending, 1360-byte virus containing the texts 'COWIIBAIAVDRWEADHICH', 'Spartak Virus by Crazy Punk (C) | v1.0 beta', 'Moscow, Russia, 06/10/1998', 'F_C_S_M.COM', '*.com' and '*.exe'. The last two bytes are XOR-ed together give the value of 0FFh.

```
Spartak.1360         B919 031E 06E8 0000 FA5D 81ED 0E01 0E1F BA40 0052 07B8 ????
```

Spartak.1453

CEN: An encrypted, appending, 1453-byte virus with the texts 'COWIIBAIAVDRWEADHICH', 'Spartak-II Virus by Crazy Punk (C)', 'Moscow, Russia, 06/10/1998', '*.com' and '*.exe'. The last two bytes are XOR-ed together give the value of 0FFh.

```
Spartak.1453         1EB9 4804 06BA 4000 E800 00FA 5D81 ED11 010E 1F52 07B8 ????
```

Spy.447

CN: A 447-byte appender with the texts 'host.com', 'Opening file:', 'Unable to open file.', 'Storing first three bytes:', 'Storing file size...', 'Appending virus code...' and 'Setting jump to virus code...'.

```
Spy.447              B440 B9BF 018D 9600 01CD 21C3 8D9E 9E02 E82D 008B 8601 012D
```

Variola

MDR: An boot sector virus which infects MBRs on hard disks and DOS Boot Sectors on diskettes. It has the encrypted text 'PeaceMaker by VaRiOLa'. The virus stores the original boot sectors encrypted.

```
Variola              8BD9 D1E9 4B8A 248A 0032 E132 C126 8805 2688 2146 474B E2EC
```

Wild.2406

CER: An appending, 2406-byte virus.

```
Wild.2406            B873 0BBB 7373 CD21 80FC 7374 03E9 6B08 0E58 1E5B 2BC3 7518
```

XM.2401

CE: A polymorphic, 2401-byte appender with the texts '[XyeBo_MHe], (c)MidnighÅPr0wler -=Version', 'COMMAND.COMDOS4GW.EXEIBMBIO.COMCOMEXEcomexe' and 'c:\autoexec.bat'. Infected files have the word E958h at offset 0000h (COM) and the word FAFAh at offset 0010h (EXE). The following template detects the virus in memory only.

```
XM.2401              B961 0953 E80E FE5B 2E89 0E1E  0406 1FB4 BFBA 1C00 E862 FAE
```

https://www.virusbulletin.com/uploads/pdf/magazine/1999/199901.pdf

Coarse Signatures: White/Black Listing

- Maintain database of cryptographic hashes for
  - Operating system files
  - Popular applications
  - Known infected files
- Compute hash of each file
- Look up into database
- Needs to protect the integrity of the database

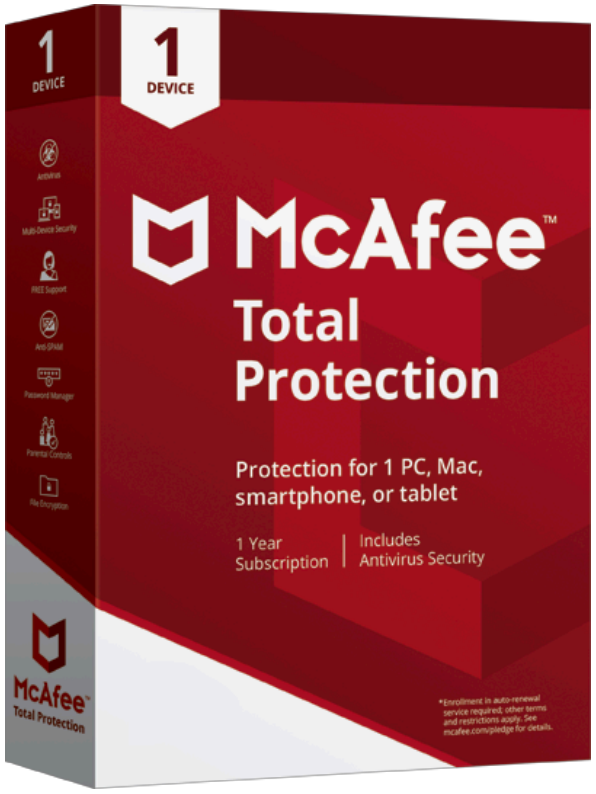
Heuristic Analysis

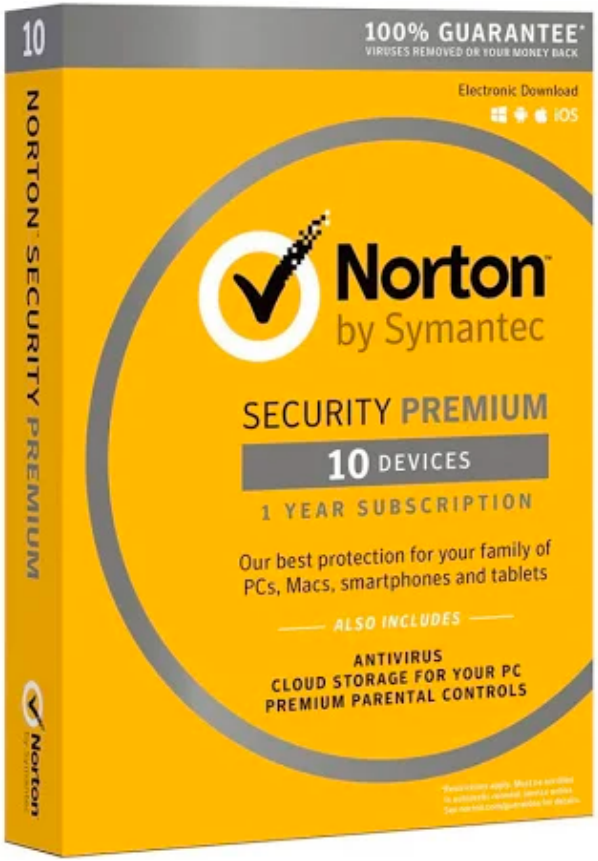
- Based on what it DOES
- Useful to identify new and “zero day” malware
- Code analysis; what code MIGHT do
  - Based on the instructions, the antivirus can determine whether or not the program is malicious, i.e., program contains instruction to delete system files,
- Execution emulation; what code ACTUALLY does
  - Run code in isolated emulation environment
  - Monitor actions that target file takes
  - If the actions are harmful, mark as virus
- Heuristic methods can trigger false alarms
  - E.g., Ransomware versus compression, full disk encryption

Example Heuristics

Network	Process	Files	Registry
connects to 80	execs cmd.exe	writes winhlp32.dat	uses wininet.dll
connects to 25	execs IEXPLORE.EXE	writes tasklist32.exe	uses PRNG
connects to 6667	execs regedit.exe	writes change.log	modifies registered applications
connects to 587	execs tasklist32.exe	writes mirc.ini	modifies proxy settings
scans port 80	execs svchost.exe	writes svchost.exe	modifies mounted drives

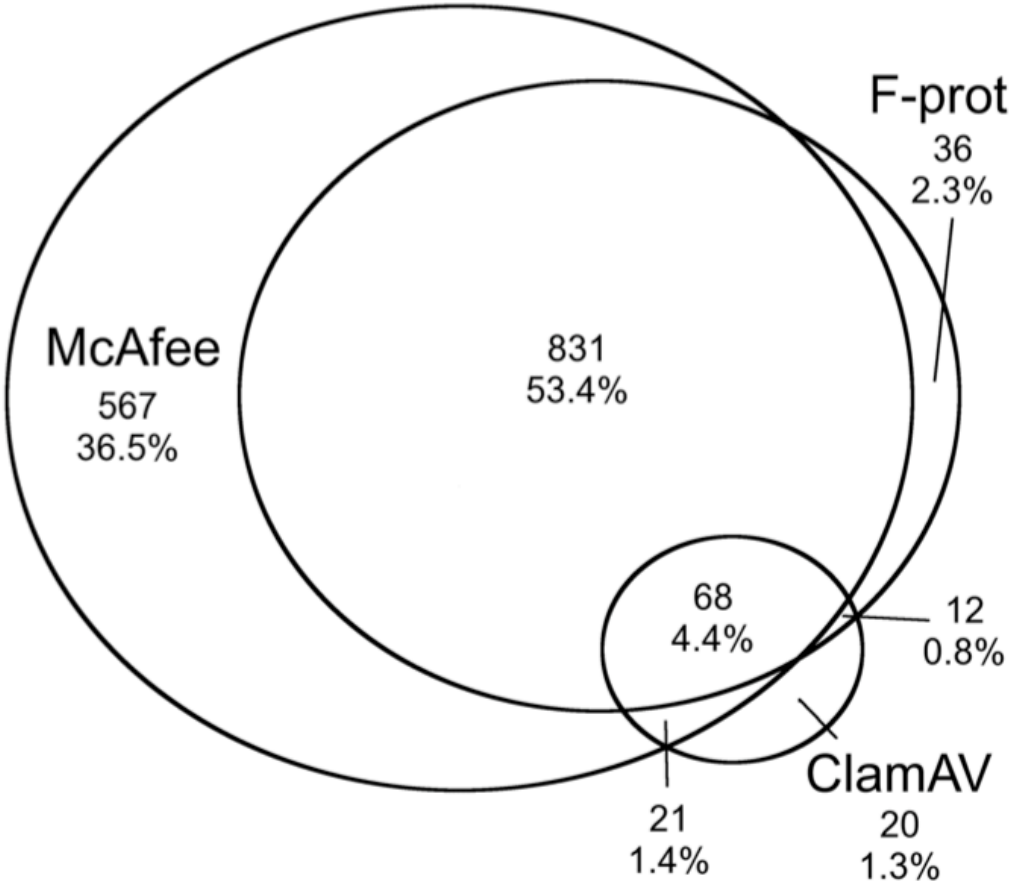
Antivirus


1
DEVICE
1
DEVICE
McAfee™
Total
Protection
Protection for 1 PC, Mac,
smartphone, or tablet
1 Year
Subscription
Includes
Antivirus Security
McAfee
Total Protection


10
100% GUARANTEE
VIRUSES REMOVED OR YOUR MONEY BACK
NORTON SECURITY PREMIUM
Electronic Download
iOS
Norton
by Symantec
SECURITY PREMIUM
10 DEVICES
1 YEAR SUBSCRIPTION
Our best protection for your family of
PCs, Macs, smartphones and tablets
ALSO INCLUDES
ANTIVIRUS
CLOUD STORAGE FOR YOUR PC
PREMIUM PARENTAL CONTROLS

SDBot

- Via manual inspection find all SDBot variants, and alias detected by McAfee, ClamAV, F-Prot

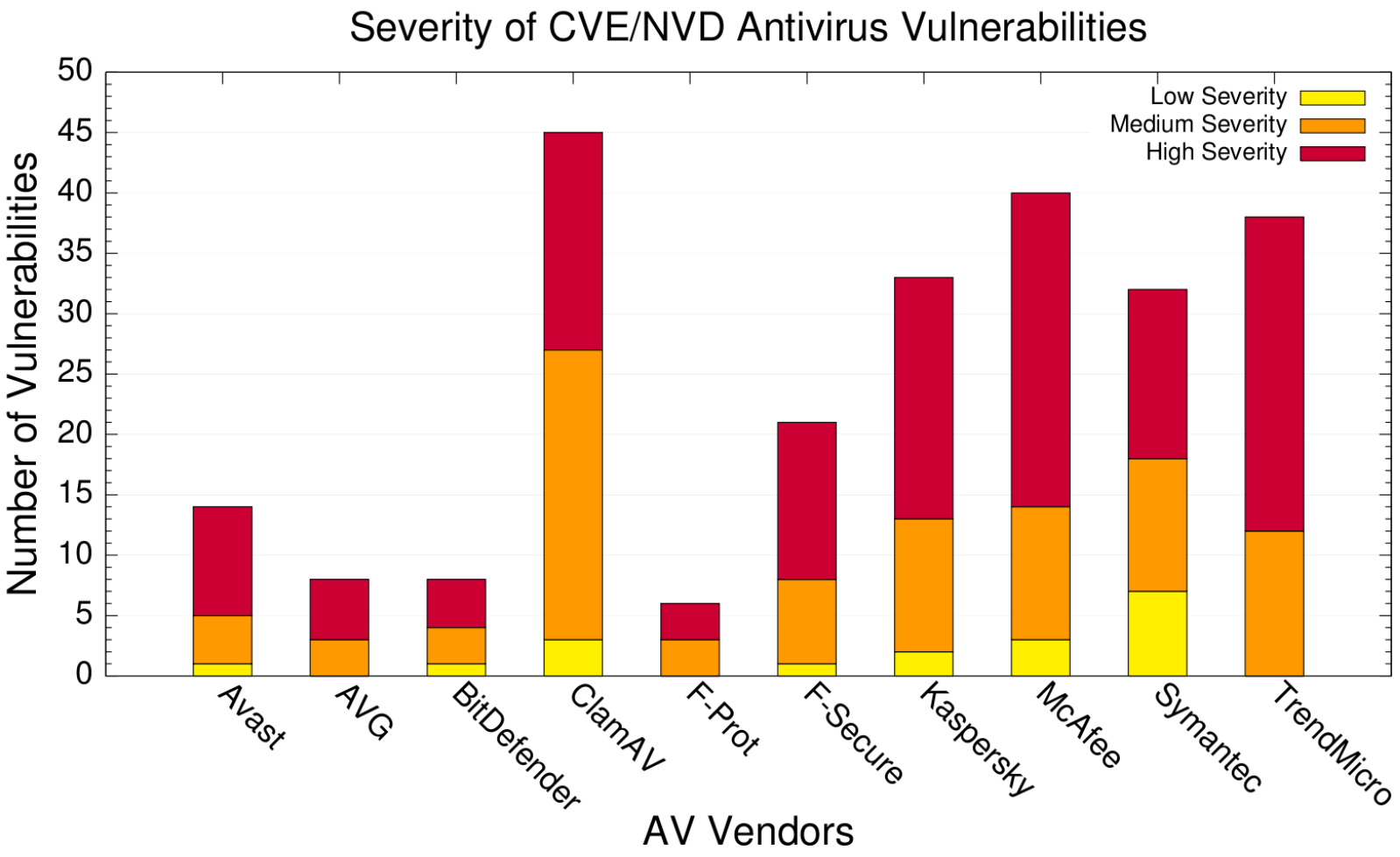


Completeness, Accuracy

- The percentage of malware samples detected across datasets and AV vendors
- ***AV system labels are incomplete***

Dataset	AV Updated	Percentage of Malware Samples Detected				
		McAfee	F-Prot	ClamAV	Trend	Symantec
legacy	20 Nov 2006	100	99.8	94.8	93.73	97.4
small	20 Nov 2006	48.7	61.0	38.4	54.0	76.9
small	31 Mar 2007	67.4	68.0	55.5	86.8	52.4
large	31 Mar 2007	54.6	76.4	60.1	80.0	51.5

Antivirus Vulnerabilities

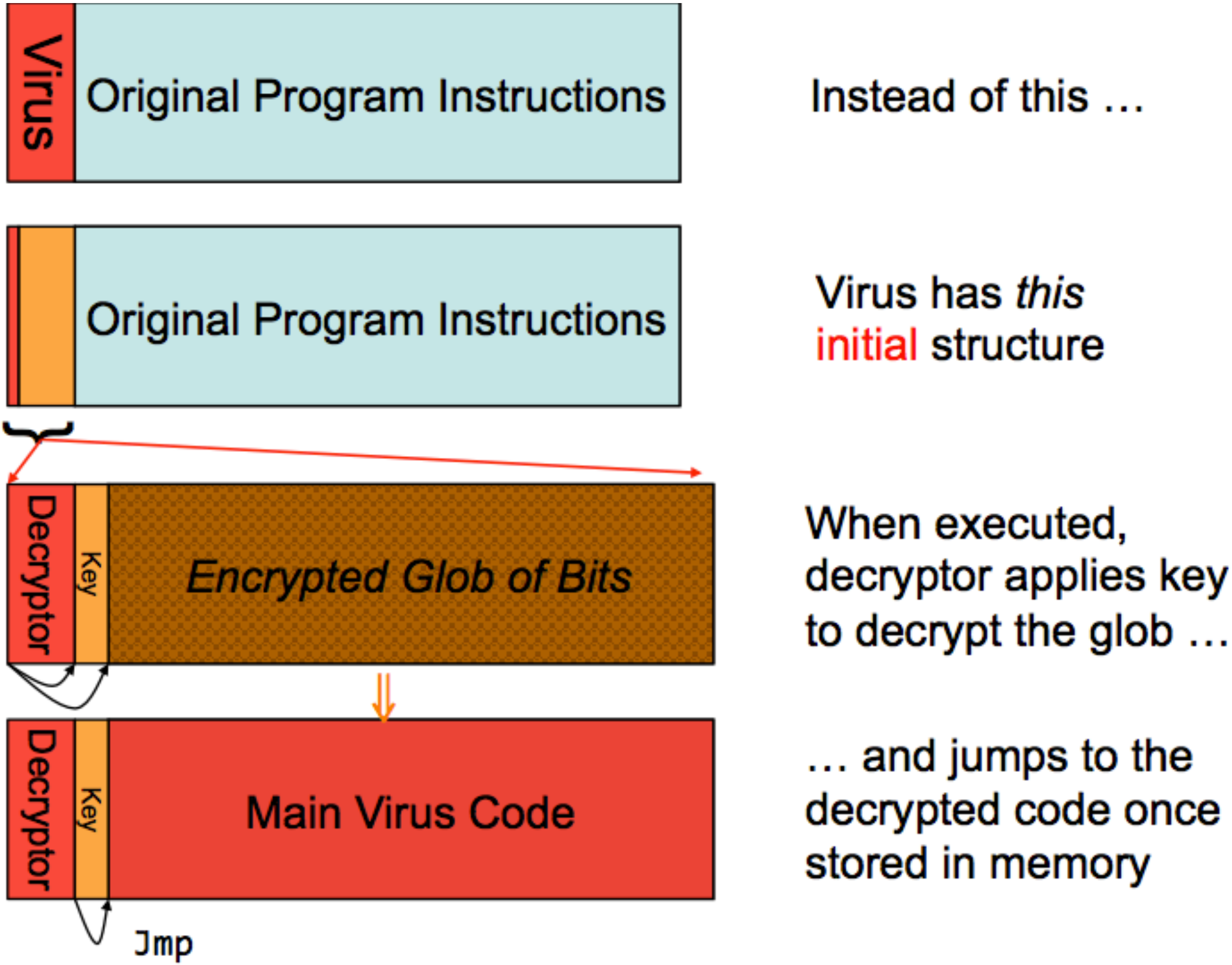


Antivirus engines vulnerable to numerous local and remote exploits

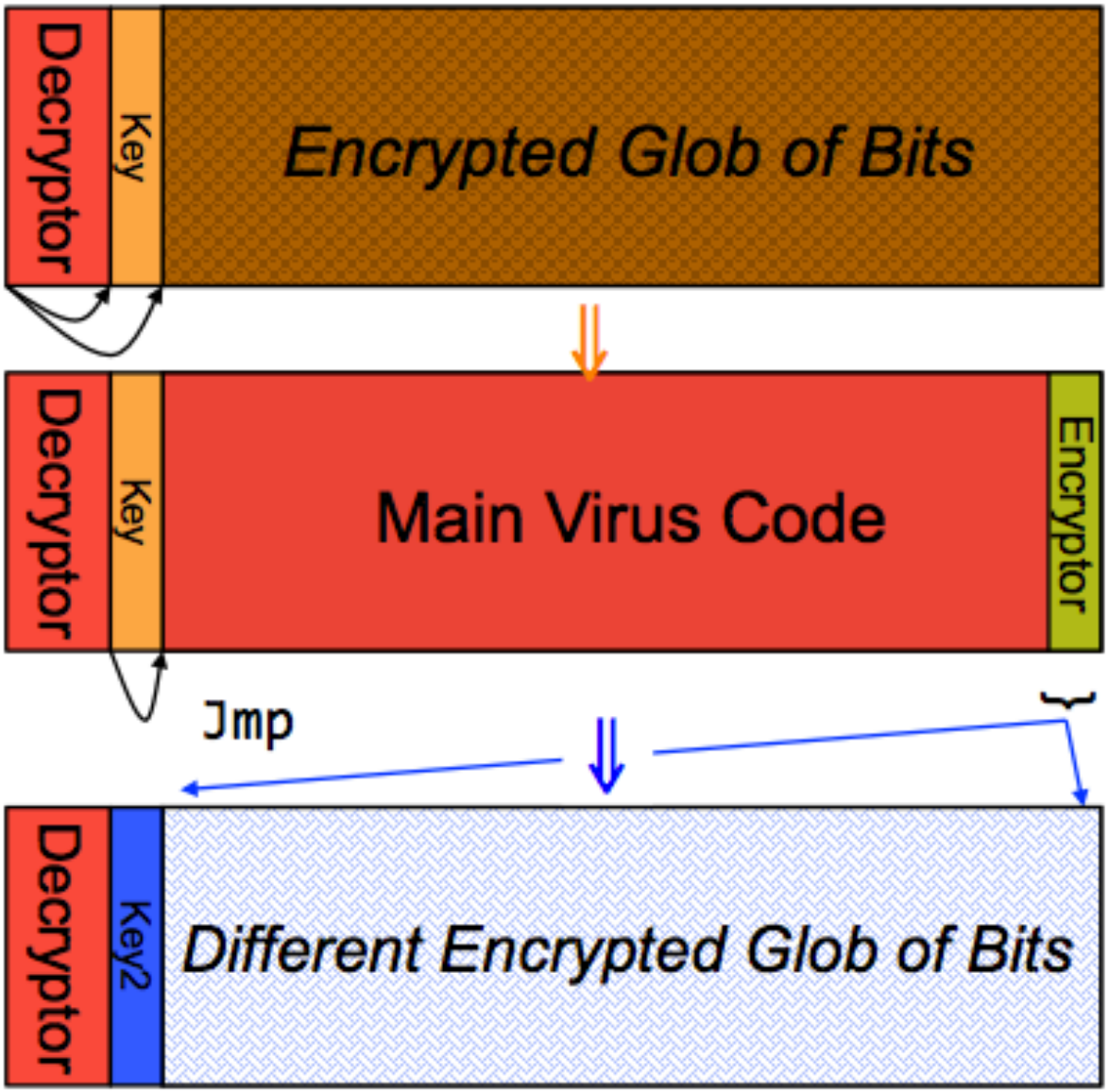
(number of vulnerabilities reported in NVD from Jan. 2005 to Nov. 2007)

Concealment

- Encrypted virus
  - Decryption engine + encrypted body
  - Randomly generate encryption key
  - Detection looks for decryption engine
- Polymorphic virus
  - Encrypted virus with random variations of the decryption engine (e.g., padding code)
  - Detection using CPU emulator
- Metamorphic virus
  - Different virus bodies
  - Approaches include code permutation and instruction replacement
  - Challenging to detect


Virus
Original Program Instructions
Instead of this …
Original Program Instructions
Virus has *this* initial structure
Decryptor
Key
Encrypted Glob of Bits
When executed, decryptor applies key to decrypt the glob …
Decryptor
Key
Main Virus Code
… and jumps to the decrypted code once stored in memory
Jmp

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

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")

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!***

Anomaly-Based HIDS

- Idea behind HIDS
  - Define normal behavior for a process
    - Create a model that captures the behavior of a program during normal execution.
  - Monitor the process
    - Raise a flag if the program behaves abnormally

Why System Calls?

- The program is a layer between user inputs and the operating system
- A compromised program cannot cause significant damage to the underlying system without using system calls
- i.e Creating a new process, accessing a file etc.

Syscall N-Grams

- Forrest et. al. A Sense of Self for Unix Processes, 1996.
- Tries to define a normal behavior for a process by using sequences of system calls.
- As the name of their paper implies, they show that fixed length short sequences of system calls are distinguishing among applications.
- For every application a model is constructed and at runtime the process is monitored for compliance with the model.
- *Definition:* The list of system calls issued by a program for the duration of it's execution is called a *system call trace*.

N-Gram: Building the Model by Training

- Slide a window of length N over a given system call trace and extract unique sequences of system calls.

Example:

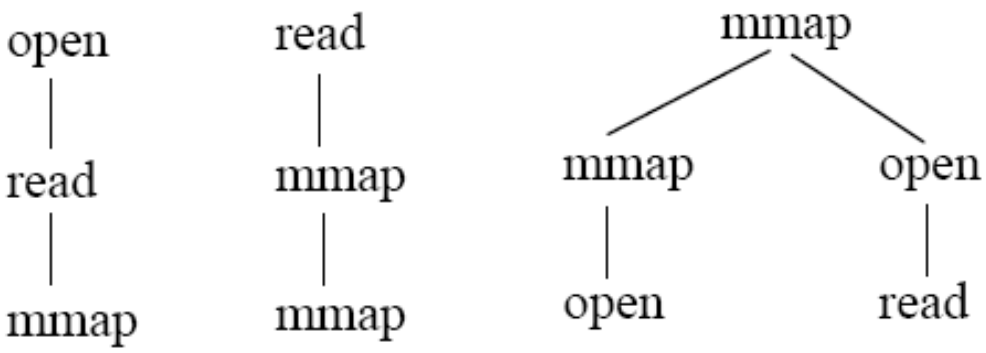
open, read, mmap, mmap, open, read, mmap

System Call trace

Unique Sequences

open, read, mmap
read, mmap, mmap
mmap, mmap, open
mmap, open, read

Database



N-Gram: Monitoring

- Monitoring
  - A window is slid across the system call trace as the program issues them, and the sequence is searched in the database.
  - If the sequence is in the database then the issued system call is valid.
  - If not, then the system call sequence is either an intrusion or a normal operation that was not observed during training (false positive) !!

Experimental Results for N-Gram

- Databases for different processes with different window sizes are constructed
- A normal sendmail system call trace obtained from a user session is tested against all processes databases.
- The table shows that sendmail's sequences are unique to sendmail and are considered as anomalous by other models.

	5		6		11	
Process	%	#	%	#	%	#
sendmail	0.0	0	0.0	0	0.0	0
ls	6.9	23	8.9	34	13.9	93
ls -l	30.0	239	32.1	304	38.0	640
ls -a	6.7	23	8.3	34	13.4	93
ps	1.2	35	8.3	282	13.0	804
ps -ux	0.8	45	8.1	564	12.9	1641
finger	4.6	21	4.9	27	5.7	54
ping	13.5	56	14.2	70	15.5	131
ftp	28.8	450	31.5	587	35.1	1182
pine	25.4	1522	27.6	1984	30.0	3931
httpd	4.3	310	4.8	436	4.7	824

The table shows the number of mismatched sequences and their percentage wrt the total number of subsequences in the user session

To Learn More …

- Books
  - Stallings and Brown, Chapter 6
  - Pfleeger and Pfleeger, Chapter 3
  - Goodrich and Tamassia, Chapter 4
  - Anderson, Chapter 21
  - Easttom, Chapter 5
- Papers
  - Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software - Newsome*
  - Efficient Software-Based Fault Isolation
  - Scheduling Black-box Mutational Fuzzing
  - Skyfire: Data-Driven Seed Generation for Fuzzing - Wang