Lecture 18 – Malware Defenses

Stephen Checkoway

Oberlin College

Slides based on Bailey's ECE 422

Malware review

- How does the malware start running?
  - Logic bomb?
  - Trojan horse?
  - Virus?
  - Worm?

Malware review

- What does the malware do?
  - Wiper?
  - Spyware?
  - Ransomware?
  - Rootkit?
  - Dropper?
  - Bot?

MALWARE DEFENSES

Introduction

- 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)
  - Some tools do both (e.g., Snort)
- Anomaly vs. Misuse, Rule-based

Signatures: A Malware Countermeasure

- Scan and 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

Heuristic Analysis

- Useful to identify new and “zero day” malware
- Code analysis
  - 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
  - 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

Allow/Block Listing

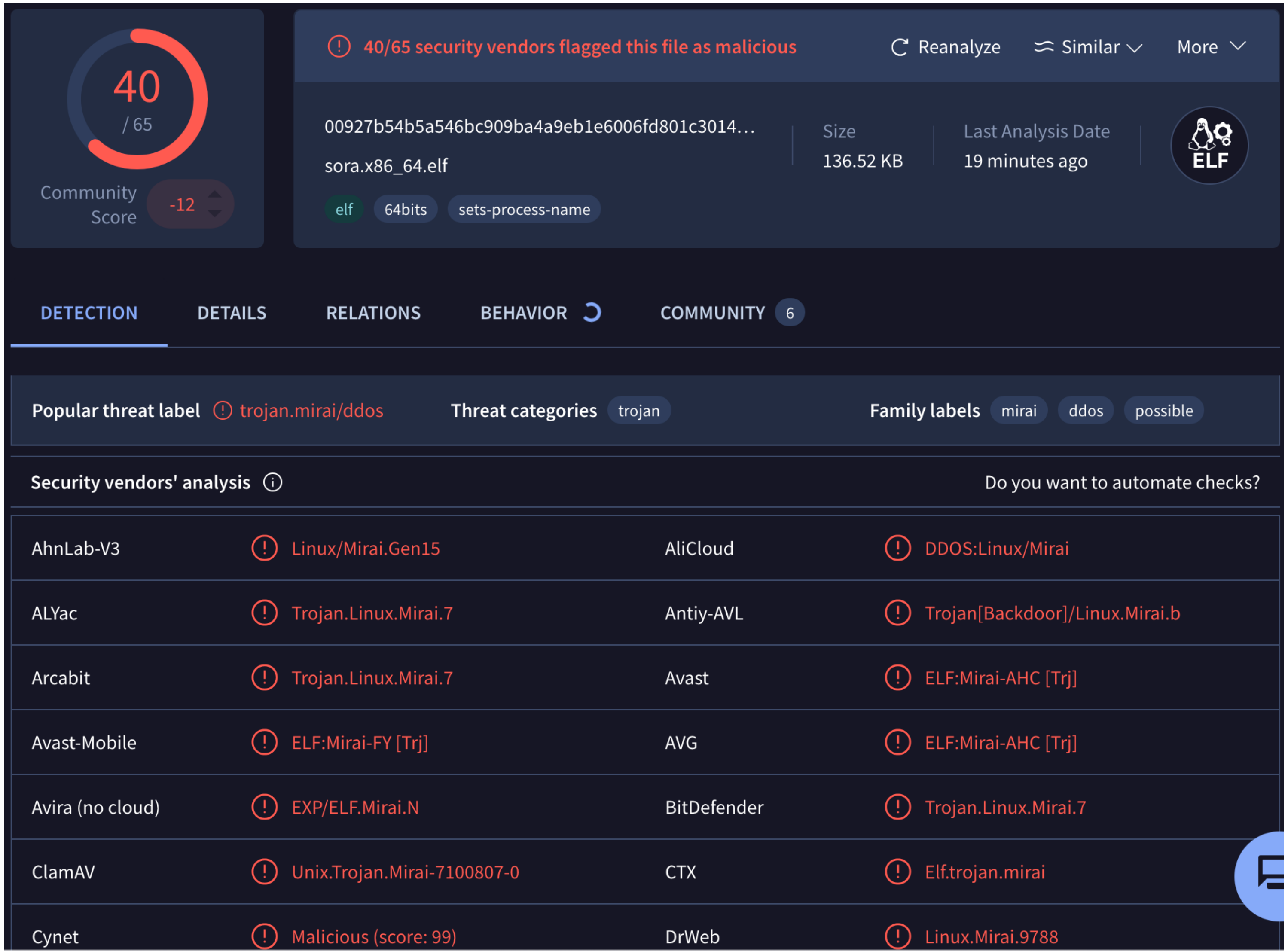
- Maintain database of cryptographic hashes for
  - Operating system files
  - Popular applications
  - Known infected files
- Compute hash of each file (md5, sha1, or sha256)
- Look up into database
- Need to protect the integrity of the database

Output of database

- If the file has been analyzed previously, the database returns a classification: known malware or not known to be malware

- If it’s malware, it returns a label as well
  - Labels are per vendor (example on the next slide)

VirusTotal output (in 2026) for known malware (Mirai botnet)



40 / 65

Community Score -12

40/65 security vendors flagged this file as malicious

00927b54b5a546bc909ba4a9eb1e6006fd801c3014...

sora.x86_64.elf

elf 64bits sets-process-name

Size 136.52 KB

Last Analysis Date 19 minutes ago

Popular threat label trojan.mirai/ddos

Threat categories trojan

Family labels mirai ddos possible

Security vendors' analysis

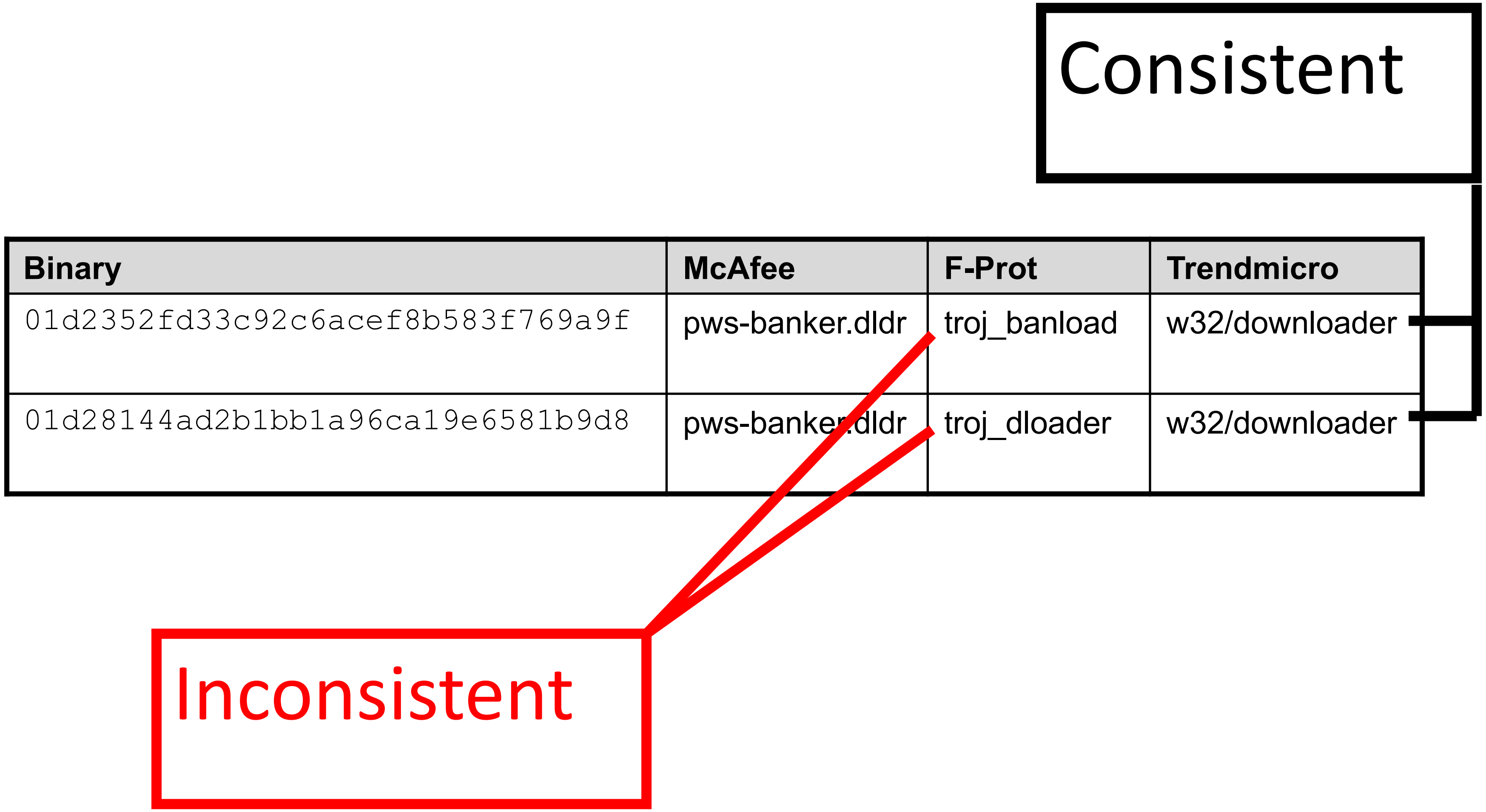
Do you want to automate checks?

AhnLab-V3	Linux/Mirai.Gen15	AliCloud	DDOS:Linux/Mirai
ALYac	Trojan.Linux.Mirai.7	Antiy-AVL	Trojan[Backdoor]/Linux.Mirai.b
Arcabit	Trojan.Linux.Mirai.7	Avast	ELF:Mirai-AHC [Trj]
Avast-Mobile	ELF:Mirai-FY [Trj]	AVG	ELF:Mirai-AHC [Trj]
Avira (no cloud)	EXP/ELF.Mirai.N	BitDefender	Trojan.Linux.Mirai.7
ClamAV	Unix.Trojan.Mirai-7100807-0	CTX	Elf.trojan.mirai
Cynet	Malicious (score: 99)	DrWeb	Linux.Mirai.9788

Properties of a good labeling system

- **Consistency.** Identical items must and similar items should be assigned the same label
- **Completeness.** A label should be generated for as many items as possible

Consistency example



Consistent

Binary	McAfee	F-Prot	Trendmicro
01d2352fd33c92c6acef8b583f769a9f	pws-banker.dldr	troj_banload	w32/downloader
01d28144ad2b1bb1a96ca19e6581b9d8	pws-banker.dldr	troj_dloader	w32/downloader

Inconsistent

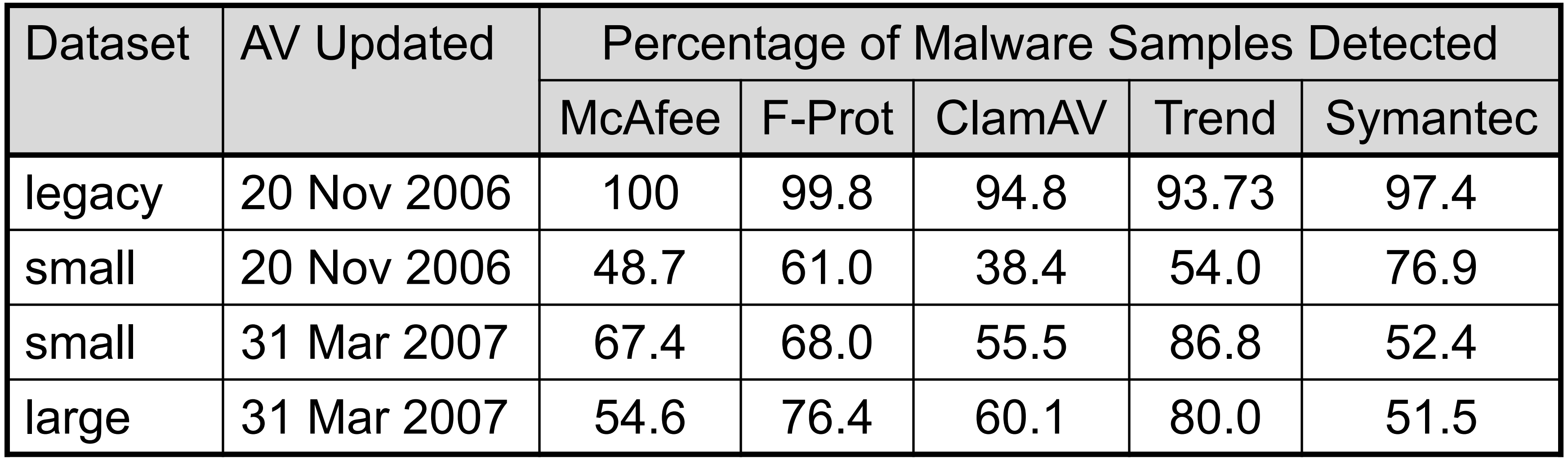
Consistency

- The percentage of time two binaries classified as the same by one AV system are classified the same by other AV systems.
- ***<u>AV system labels are inconsistent</u>***

AV	McAfee	F-Prot	ClamAV	Trend	Symantec
McAfee	100	13	27	39	59
F-Prot	50	100	96	41	61
ClamAV	62	57	100	34	68
Trend	67	18	25	100	55
Symantec	27	7	13	14	100

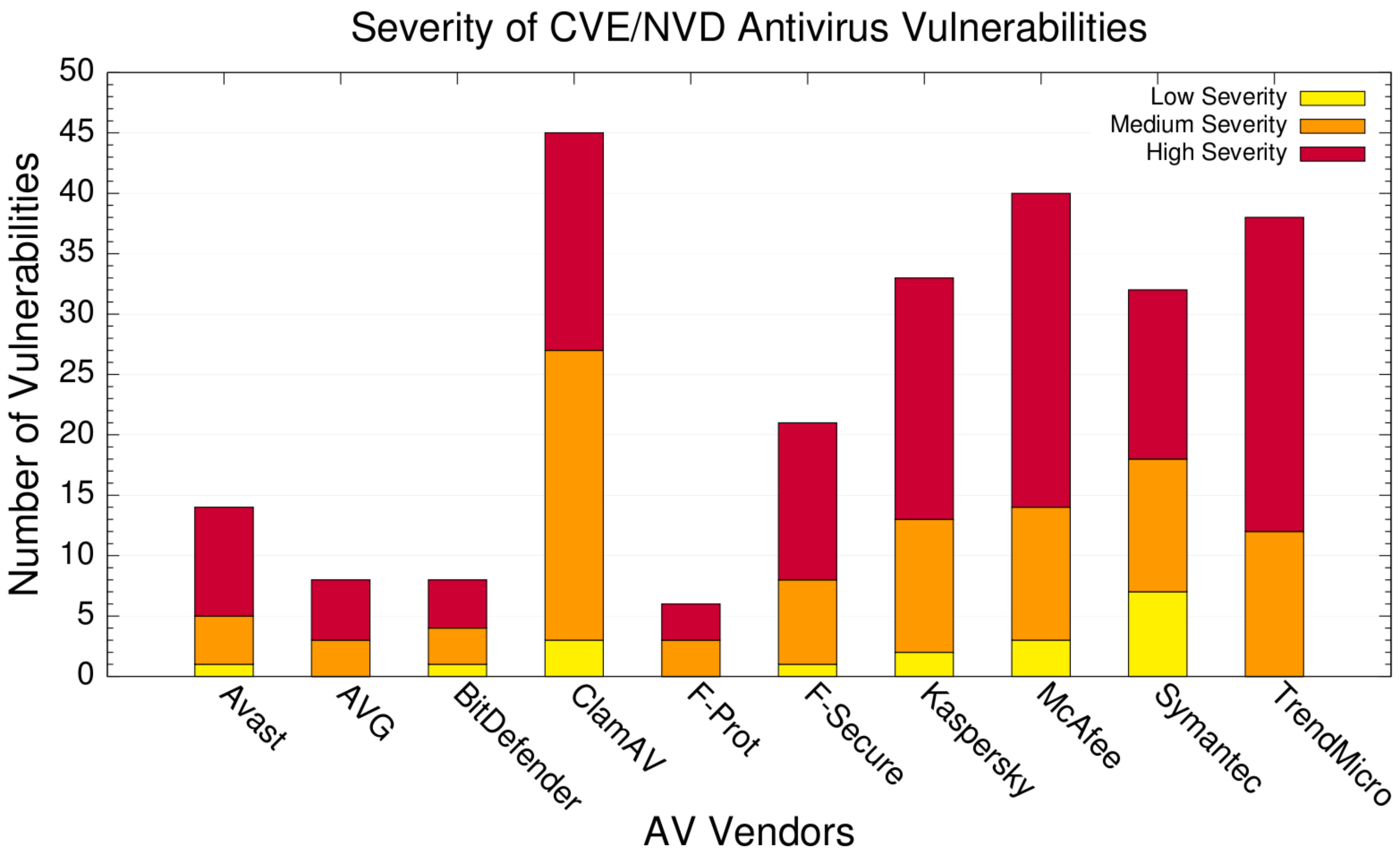
Completeness

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



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

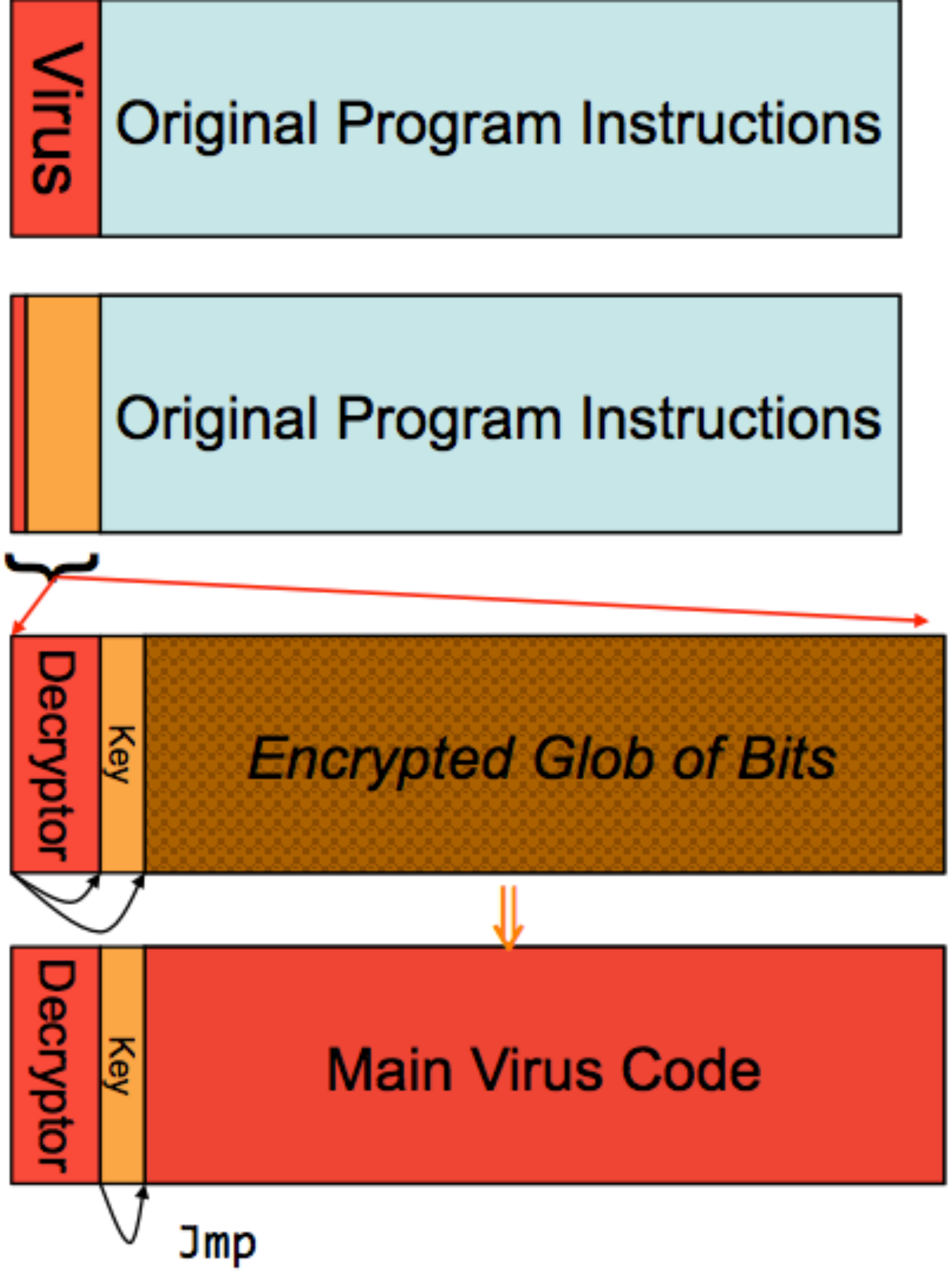


Antivirus engines vulnerable to
numerous local and remote exploits

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

Concealment

- Encrypted virus
  - Decryption engine + encrypted body
  - Randomly generate encryption key



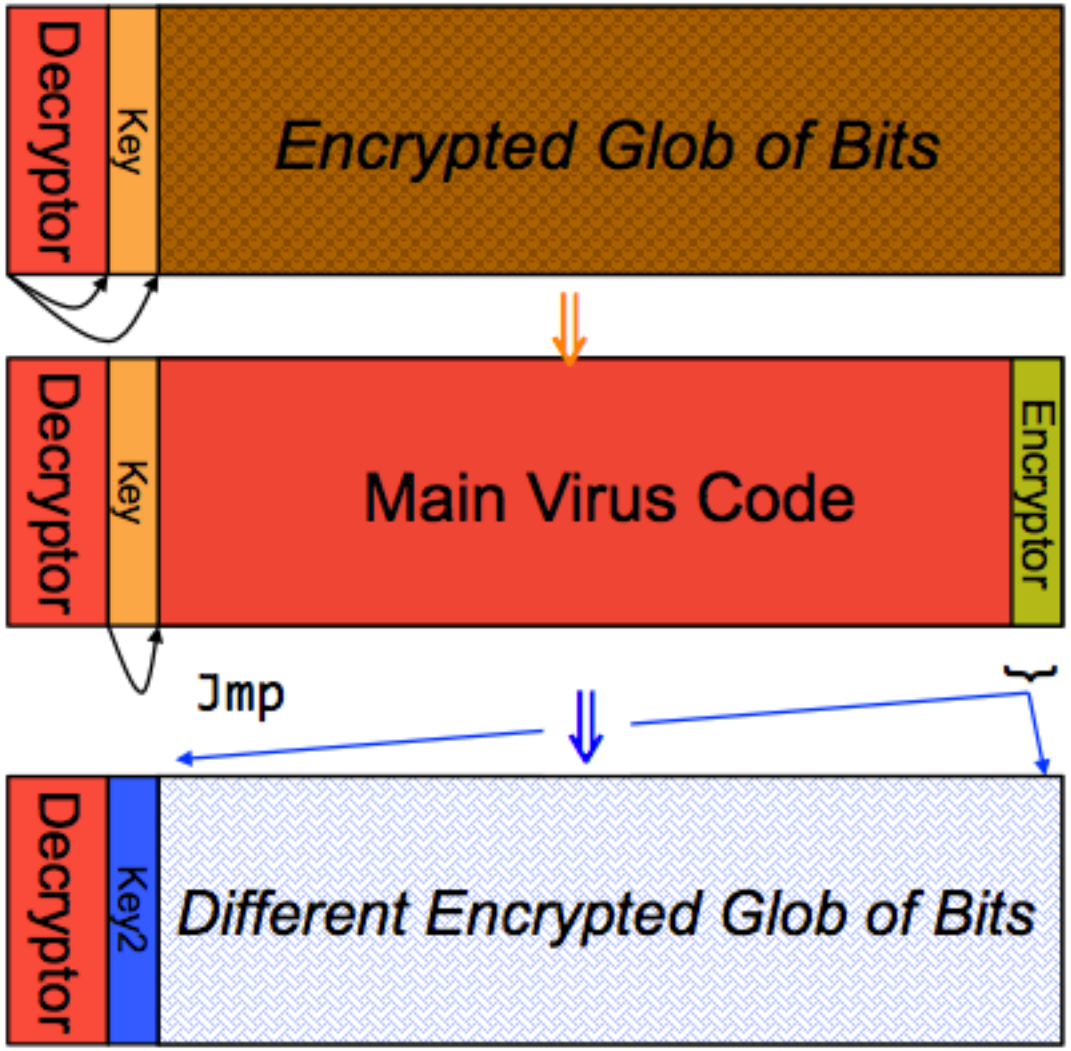
Instead of this …

Virus has *this* initial structure

When executed, decryptor applies key to decrypt the glob …

… and jumps to the decrypted code once stored in memory

Encrypted Virus Propagation



Once running, virus uses an *encryptor* with a new key to propagate

New virus instance bears little resemblance to original

Concealment

- Encrypted virus
  - Decryption engine + encrypted body
  - Randomly generate encryption key
  - Look for the decryption engine rather than the virus body

Concealment (polymorphic virus)

- Polymorphic virus
  - Encrypted virus with random variations of the decryption engine (e.g., padding code)
  - [How can we detect this?]

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

Concealment one step further

- Metamorphic virus
  - Different virus bodies
  - Approaches include code permutation and instruction replacement
  - Challenging to detect

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.
    - Usually monitor system calls [Why system calls?]
  - Monitor the process
    - Raise a flag if the program behaves abnormally

Why System Calls? (Motivation)

- 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
- e.g., Creating a new process, accessing a file

Model Creation Techniques

- Models are created using two different methods:
  - Training: The program’s behavior is captured during a training period, in which, there is assumed to be no attacks. Another way is to craft synthetic inputs to simulate normal operation.
  - Static analysis: The information required by the model is extracted either from source code or binary code by means of static analysis.
- Training is easy, however, the model may miss some of the behavior and therefore produce false positives.

N-Gram

- 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 its 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:

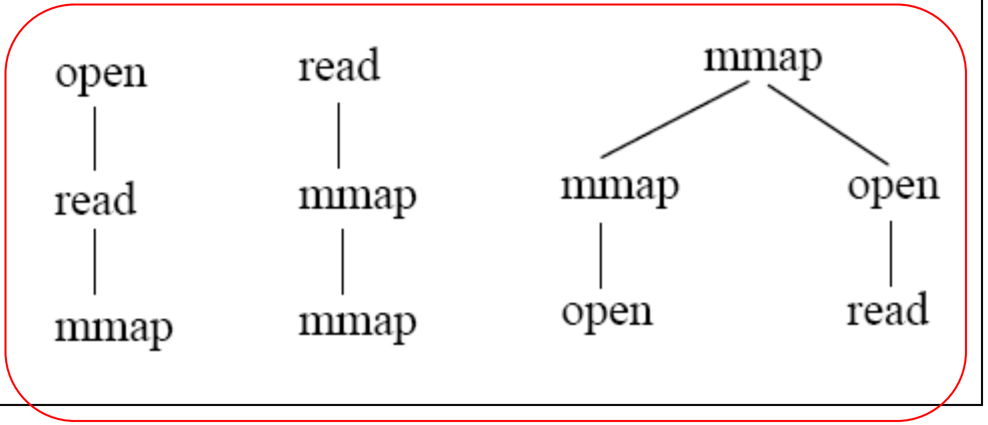
open, read, mmap, mmap, open, read, mmap

System Call trace

Unique Sequences

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

Database



open
read
mmap

read
mmap
mmap

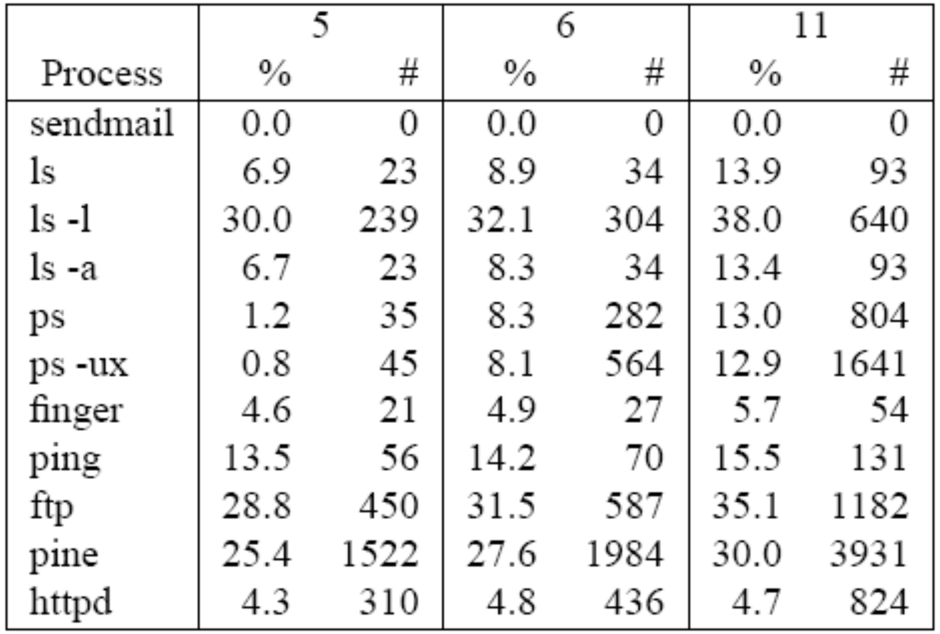
mmap
mmap
open
open
read

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 with respect to the total number of subsequences in the user session