Lecture 18 – Malware Defenses

Ryan Cunningham
University of Illinois

ECE 422/CS 461 – Fall 2017

Announcement

- Midterm:
  - Monday, Oct. 16th 7-9pm (one week!)
  - ECEB 1002 (here)
- Conflict
  - Friday Oct. 13th 4-6pm
  - Siebel Center 4405
  - MUST have an email from you

Security News

- Apple emergency patch for High Sierra disk encryption bug
- Disqus breached in 2012

Security Alert: User Info Breach

Posted by Jason Yan on October 06, 2017

Yesterday, on October 5th, we were alerted to a security breach that impacted a database from 2012. While we are still investigating the incident, we believe that it is best to share what we know now. We know that a snapshot of our user database from 2012, including information dating back to 2007, was exposed. The snapshot includes email addresses, Disqus user names, sign-up dates, and last login dates in plain text for 17.5mm users. Additionally, passwords (hashed using SHA1 with a salt; not in plain text) for about one-third of users are included.

We sincerely apologize to all of our users who were affected by this breach. Our intention is to be as transparent as possible about what happened, when we found out, what the potential consequences may be, and what we are doing about it.

Timeline Of Events:

- **Thursday, October 5, 2017 at 4:18 PM PDT**, we were contacted by an independent security researcher, who informed us that the Disqus data may be exposed.
- **Thursday, October 5, 2017 at 4:56PM PDT** we obtained the exposed data and immediately began to analyze the data and verify its validity.
- **Friday, October 6, 2017**, we started contacting users and resetting the passwords of all the users that had passwords included in the breach.
- **Friday, October 6, 2017, before 4:00PM PDT**, we published this public disclosure of the incident.

Potential Impact For Users:

- Right now there isn't any evidence of unauthorized logins occurring in relation to this. No plain text passwords were exposed, but it is possible for this data to be decrypted (even if unlikely). As a security precaution, we have reset the passwords for all affected users. We recommend that all users change passwords on other services if they are shared.
- Email addresses are in plain text here, so it's possible that affected users may receive spam or unwanted emails.
- At this time, we do not believe that this data is widely distributed or readily available. We can also confirm that the most recent data that was exposed is from July, 2012.

What We Are Doing to Address This:

As a precautionary measure, we are forcing the reset of passwords for all affected users. We are contacting all of the users whose information was included to inform them of the situation.

We've taken action to protect the accounts that were included in the data snapshot. Right now, we don't believe there is any threat to a user accounts. Since 2012, as part of normal security enhancements, we've made significant upgrades to our database and encryption in order to prevent breaches and increase password security. Specifically, at the end of 2012 we changed our password hashing algorithm from SHA1 to bcrypt.

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
  - **Prevention** stops the attack **before** it reaches the system

Intrusion Detection Systems (IDS)

- Components:
  - Sensors - collect data
  - Analyzers - evaluates
  - User interface
- Can Be:
  - Anomaly based
  - Signature/Heuristic based

Bad detection

- False positives
  - Report activity as an intrusion, but it isn’t
  - Reduce by loosening intrusion detection rules
- False negative
  - Miss reporting bad behavior as an intrusion
  - Reduced by tightening intrusion detection rules

IDS Approaches

- Anomaly detection
  - Collect data on legitimate users
  - Benchmark expected legitimate/illegitimate activity
  - Flag behavior that is anomalous (weird)
- Signature/Heuristic detection
  - Examine known attack patterns
  - Develop signatures to detect them
  - Can only identify known attacks

Anomaly Detection

- Statistical
  - measure legitimate activity
  - use univariate or multivariate models
  - advantage - simplicity
  - disadvantage - usually not a good model (hard to tune FPR)
- Knowledge-based
  - Develop a series of rules based on observations
  - advantage - robust and flexible
  - disadvantage - requires a lot of data/human intervention

Anomaly Detection

- Machine learning
  - Give algorithm examples of positive and negative behavior
  - advantage - flexible and adaptable
  - disadvantage - high FPR, high resource cost
- Many different approaches
  - Bayesian networks
  - Neural networks
  - Probabilistic modeling
  - Support vector machines
  - Clustering/outlier detection

Host-Based IDS (HIDS)

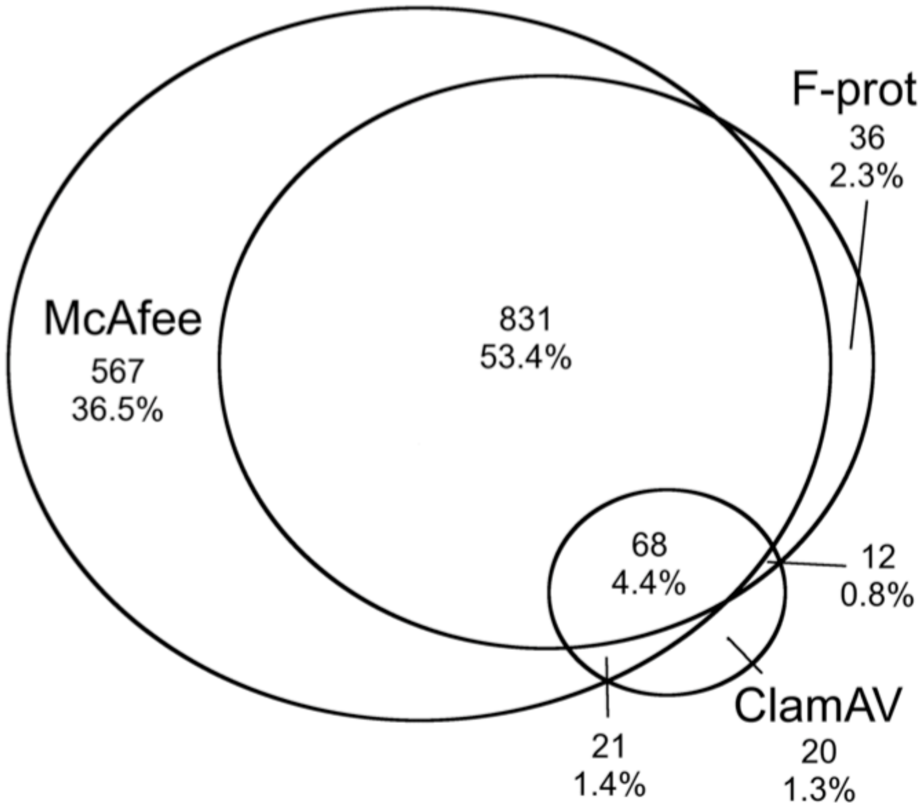
- Data sources and sensors
  - System call traces (Unix/Linux)
  - Audit records (log files)
  - File integrity checksums
  - Registry accesses (Windows)
- Anomaly HIDS
  - typically Linux based
  - train on system calls “95-99% detection rate with <5% FPR”
- Signature HIDS
  - antivirus

Signatures: Malware Countermeasure

- Scan compare the analyzed object with a database of signatures
- Signature is a virus fingerprint
  - E.g. a sequence of instructions unique to virus (***not*** a digital signature)
- File is infected if there is a signature inside its code
  - Fast pattern matching techniques to search for signatures
- All the signatures together form database (usually is proprietary)

SDBot

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



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>***

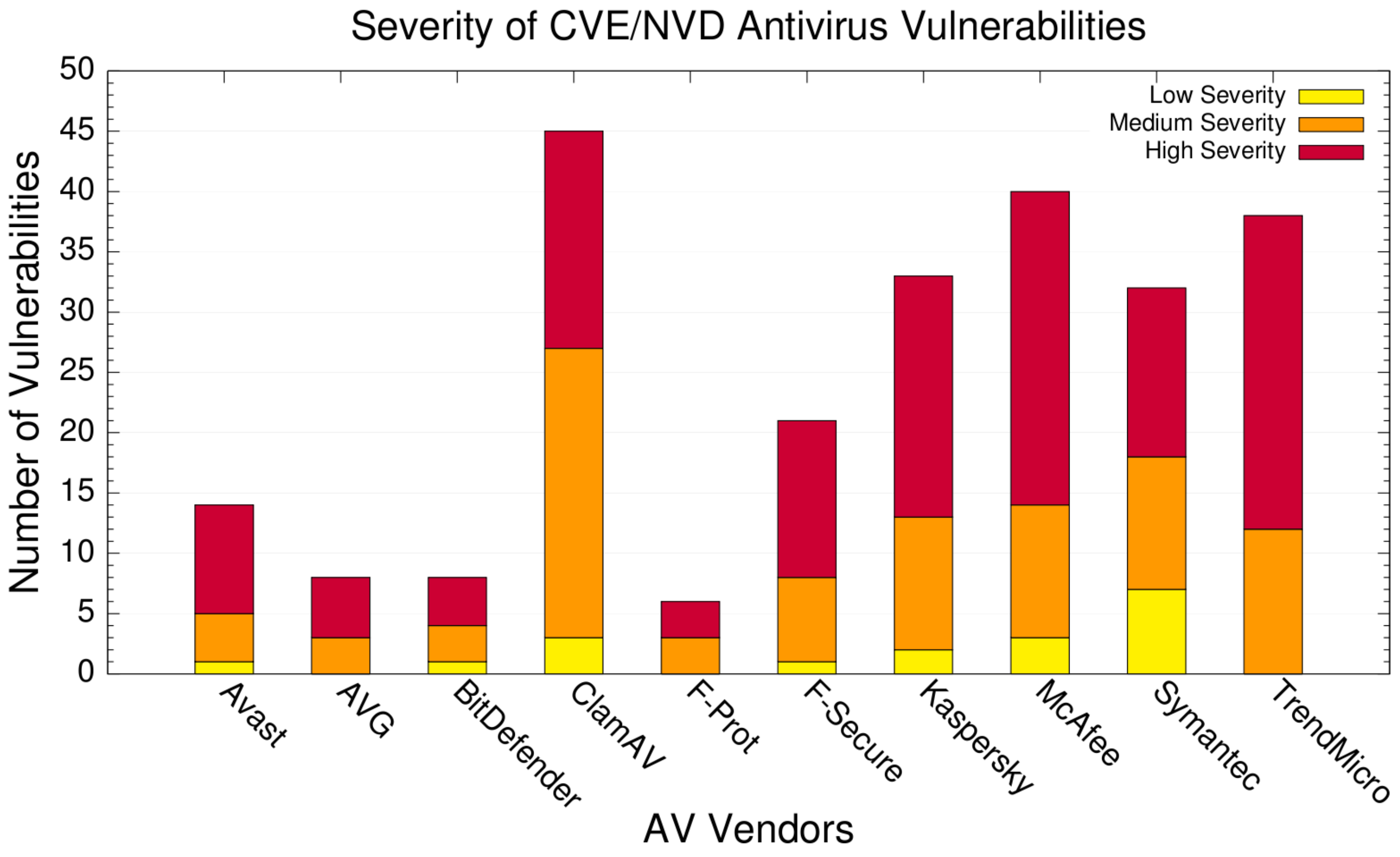
AV	McAfee	F-Prot	ClamAV	Trend	Symantec
McAfee	100	13	27	39	59
F-Prot	50	100	96	41	61
ClamAV	62	57	100	34	68
Trend	67	18	25	100	55
Symantec	27	7	13	14	100

Completeness

- The percentage of malware samples detected across datasets and AV vendors
- ***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)

White/Black Listing

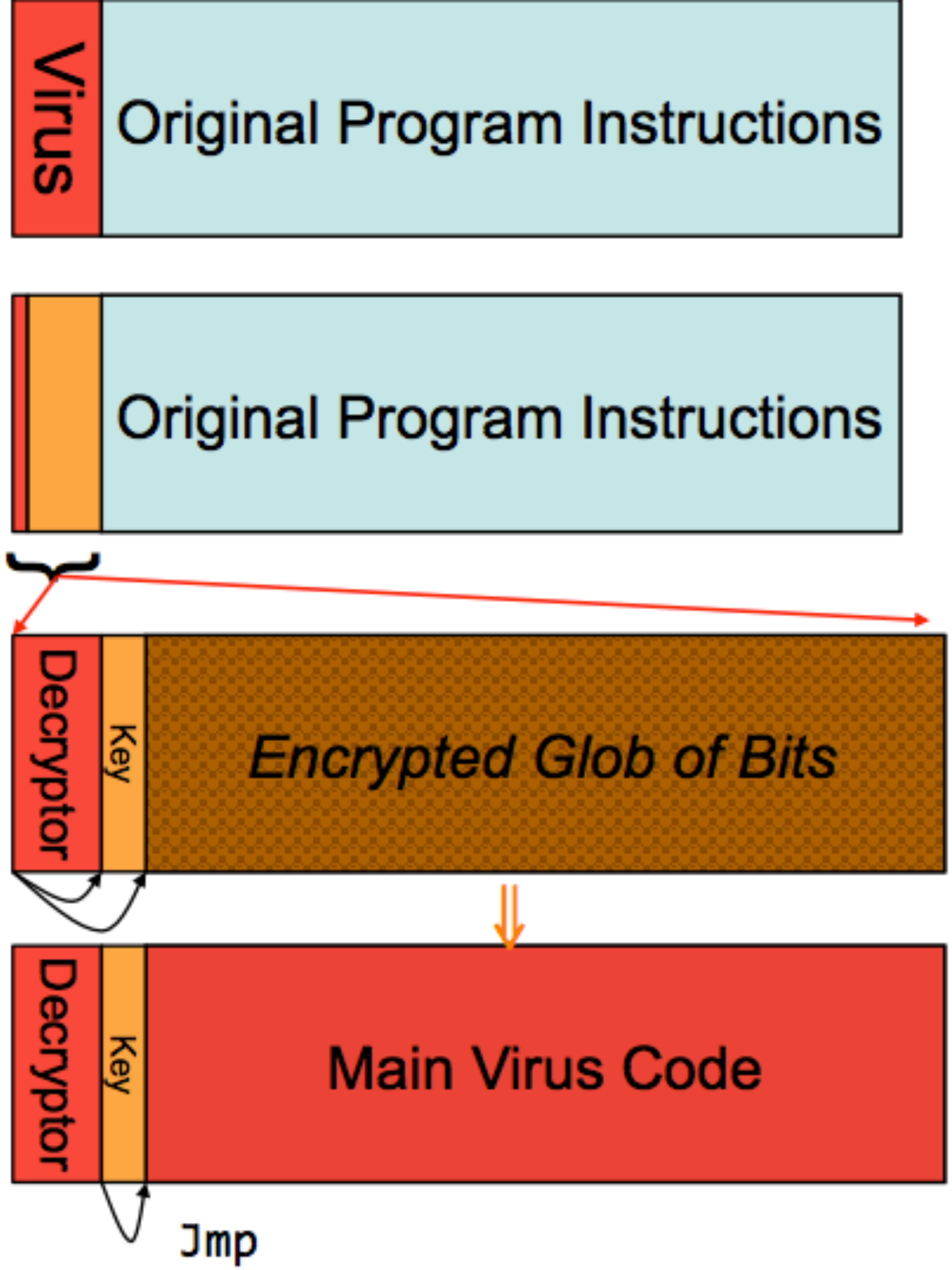
- Maintain database of signatures (cryptographic hashes) for
  - Operating system files
  - Popular applications
  - Known infected files
- Compute hash of each file
- Look up into database
- Need to protect the integrity of the database

Heuristic Analysis

- Useful to identify new and “zero day” malware
- Code analysis
  - e.g. scan program for 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

Countermeasure: 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



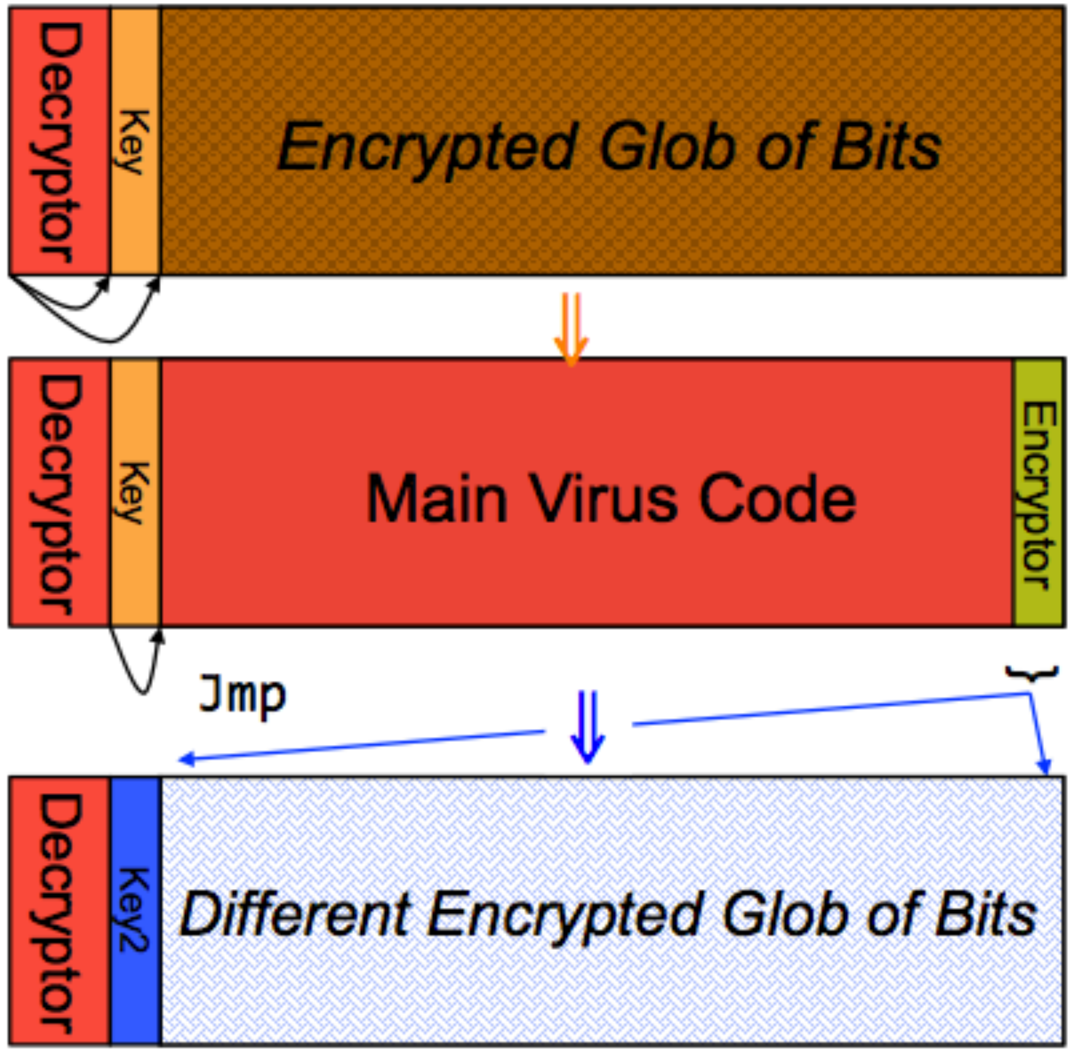
Instead of this …

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

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

- 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? (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
- i.e Creating a new process, accessing a file etc.

Model Creation Techniques

- Models are created using two different methods:
  - Training: The programs 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
- Define normal behavior for process using sequences of system calls
- Authors show short sequences of system calls cn distinguish between applications
- Construct model for process, monitor application at runtime for compliance
- *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:

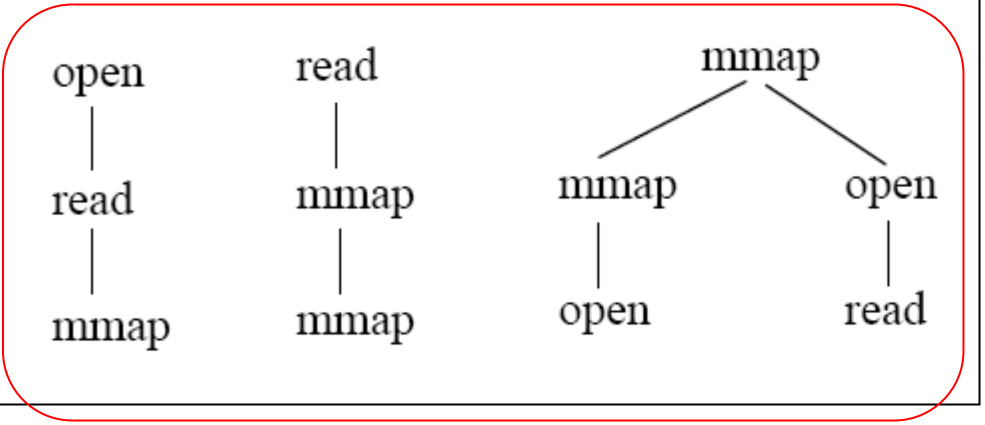
open, read, mmap, mmap, open, read, mmap

System Call trace

Unique Sequences

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

Database

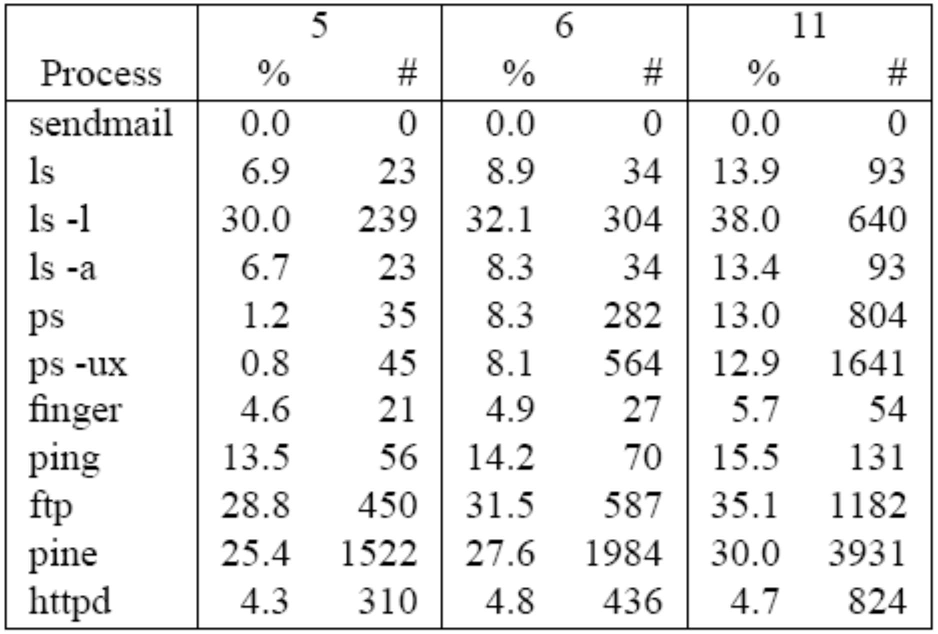


N-Gram: Monitoring

- 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:
  - intrusion
  - 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