Covert Channels and Side-Channel Attacks

Covert Channels

- **Confidential information may be leaked via channels that may be missed easily**
  - Implicit flows in a program
  - Timing channels
  - Steganographic techniques
- **Examples**
  - transmit info by file name or metadata (e.g., timestamp)
    - Information retrieved by checking file presence or stat
      - No need to read the file (or have read permissions on the file)
  - "Port-knocking"
    - Transmit info by probing network ports in a certain sequence
  - tcp acks or retransmissions, packet fragmentation, …

Side-channel attacks

- **Critical info may be leaked inadvertently**
  - Error messages, e.g., invalid username vs password
  - Timing information
    - How long it took to verify a password
    - How long it took to encrypt a number
    - Virtualization overhead for computation vs I/O
  - Power-monitoring attacks
    - Use thermal imaging of a chip to monitor which circuits are being used and/or how much power is being used
    - Or simply monitor the power supply
  - Differential fault analysis
    - Force a particular fault (e.g., make a data line to be a “1” always) and examine how the program changes its behavior.
  - Last two attacks motivate tamper-resistance in the context of building secure devices
    - Military equipment used in the field
    - Other devices that carry secrets and may be lost

Emanations

- **Electromagnetic emanations**
  - In old days, CRTs produced a lot of emanations that can be used to figure out what someone is doing from a distance
- **Keyboard emanations**
  - Researchers have shown it is possible to steal passwords using a microphone in a nearby office!
- **Power-line emanations**
  - Correlates fluctuations in power use (or EM waves on the powerline) with computations being performed
- **Snooping using telescopes**
  - Not just on-screen images, but reflections on a cup etc.

Remanence

- **malloc after free, or reuse of stack variables**
  - Exposes secrets that may be private to one program component to another.
- **Allocation of physical page for one process after it is used by another process**
  - Exposes secrets across processes
  - Can be avoided by immediately erasing confidential data
    - Beware: the compiler may eliminate this during optimization
    - Cache contents are flushed across process switch, so not a problem
- **Retained memory contents after power off**
- **Residual effects on hard drives**
  - may be data is just unlinked, not even overwritten
  - even after overwrite, it is often possible to recover old data

Network-Based Attacks

Historical ...

No powerful infrastructure for attackers to use

- Typically, single host with dial-up access

Needed to exploit vulnerabilities in protocol implementations to carry out effective <u>attacks</u>

IP-layer

- Teardrop: overlapping IP fragments

ICMP

- Ping of death: Oversized ICMP ECHO packets
- Smurf: ICMP ECHO REPLY Flood

TCP

- SYN-flooding
- IP-spoofing

More Recent Attacks

Distributed Denial of Service (DDoS)

- Attacks carried out by a large number of “bots” that were previously compromised by the attacker
- Bots run software (installed by attacker) that permit the attacker to command them
- Bots initiate connections to avoid being noticed
  - make use of common protocols such as IRC or WWW
- Bots don’t need to hide their identity
  - but it does help, as it allows them to operate longer before being discovered and taken down
- Bots can carry out any attack, but most common are attacks that saturate a victim’s network link
  - The victim can do nothing to protect itself – it needs network help

Bots can be used for other things besides DDoS

- SPAM

Reflection attacks

- Bots send a query to well-known servers (e.g., DNS) all over the world with the spoofed IP-address of the victim

Self-propagating Attacks

E-mail viruses

- Started with Melissa (1999)
  - Cost of clean-up: $10 billion!
- Recently, email viruses used to establish backdoors
  - These are later used to deploy more sinister malware
- Most email viruses exploit “social engineering”
  - Solution to such attacks necessarily relies on user education

Self-propagating Attacks

Worms

- Morris – over 6000 hosts (1988)
- Code Red -- ~300K hosts (2001)
- Nimda -- ~200K (2001)
- Slammer -- ~75K hosts in 10 minutes! (2003)

Types of worms

Differences arise primarily due to scanning and propagation techniques

- Random scanning
- Localized scanning
- Pregenerated hit lists
- Permutation scanning
  - Combining hit lists with permutation scanning can produce Warhol worms that can spread within 15 minutes!
- Topologically-aware
- Flash worms (spreads in one minute or less!)

Multi-vector (contagion worm)

- Spread from server to client, and then from client to server
- Nimda (email, IIS vulnerability, browser vulnerability, and other vulnerabilities and backdoors)

For more details, see

- "How to 0wn the Internet in your spare time," 2002 USENIX Security Symposium

Classes of Attacks

Probing: Reconnaissance before attack

- Port sweeps
- OS/application finger printing

Denial of Service (DoS)

Privilege escalation

- Remote to user
  - attacker without any access to the victim machine gains access as a normal user, e.g., userid *nobody*
- User to root
  - attacker with access as normal user gains administrative privileges through an attack
- These two privilege escalation attacks may be chained
- Remote-to-user attacks typically exploit server applications (e.g., web server), while user-to-root attacks exploit other applications.
- They are rarely caused by OS errors or errors in network protocol implementations

Techniques for protecting networks

Firewalls

- Expose only the most secure servers that need to be accessible from outside
- Limit access to those users that need access
- Use VPN/NAT

Server configuration

- If possible, enable firewall capability on server
- Use service configuration tools (e.g., /etc/inetd.conf) to limit access to only those servers that need to be accessible
- Never underestimate the importance of backups
- Key: Data integrity is often much more important than confidentiality in commercial and educational enterprises
- Pay attention to file system permission settings
- patch server software
- Do not install non-essential software/services
- Virus detection
- File system integrity checks (detect Trojan software)

Techniques for protecting networks

Auditing

- Maintain careful logs of all accesses
  - on firewalls, servers, desktops, …

Vulnerability analysis

- scan for network/software vulnerabilities

Intrusion detection

- The technique of last resort

Intrusion Detection

Intrusion Detection

Some attacks will get through in spite of every protection measure. Intrusion detection is targeted to detect such attacks.

Detection is a solution of last resort

Assumption: Behavior of a system changes when it is subjected to attack

Approach: Detect these changes in behavior

Intrusion Detection Issues

Detection rate

- What fraction of attacks are detected

False alarm rate

- May be measured in multiple ways
  - how many false alarms per day
  - what fraction of normal behavior is flagged as attack
  - what fraction of behavior reported as attack is *not* an attack
- Considerable disagreement on which measure to use
  - but the third criteria is probably the best
  - But IDS vendors (and may be researchers) don't like it
    - Will you buy a system will FA rate of 98%?
    - But you may not mind 10 false alarms a day!

Intrusion Detection Techniques

Anomaly detection

- Use machine learning techniques to develop a profile of normal behavior
- Detect deviations from this behavior
- Can detect unknown attacks, but have high FA rate

Misuse detection

- Codify patterns of misuse
- Attack behaviors usually captured using signatures
- Can provide lower false alarm rate, but ineffective for unknown attacks

Behavior (or policy) based detection

- Specify allowable behavior, detect deviations from specifications
- Can detect new attacks with low FA, but policy selection is hard

Intrusion Detection Algorithms

Pattern-matching

- Most commonly used in misuse and behavior based techniques

Machine-learning

- Statistical
- Algorithmic
- Neural networks and other techniques

Intrusion Detection Behaviors

Behaviors of

- Users
- Systems
  - processes, kernel modules, hosts, networks, …

Intrusion Detection Observation Points

Network-based (Network intrusion detection systems)

- Benefits
  - Unintrusive: plug a dedicated NIDS device on the network
  - Centralized monitoring
- Problems
  - Encryption
  - Level of abstraction too low
  - Difference between data observed by NIDS and victim app.

Host-based

- Strengths/weaknesses complementary to NIDS
- May be based on
  - system-call interception
  - audit logs and other log files
  - file system integrity (TripWire)
  - keystrokes, commands, etc.

Network Intrusion Detection

Packet-based Vs Session-based

Signature-based Vs Anomaly detection

Example: SNORT (open source)

- Uses pattern-matching on individual packets

Some systems can block offending traffic

- This is often dangerous, as systems usually have high false alarm rates

Host-based Intrusion detection

System-call based characterizations most popular

Behavior-based

- System-call interposition plus wrappers
- Domain/Type Enforcement
  - Certain application classes can access only certain files
  - Can prevent many privilege escalation attacks
  - Used in SELinux

Anomaly detection

- Sequences (finite-length strings) of system calls
- FSA and PDA models of behavior
- System call arguments

Automata Models for Learning Program Behaviors

Background

Forrest et al showed that system call sequences provide an accurate and convenient way to capture security-relevant program behaviors

- Subsequent research has further strengthened this result

Key problem:

- What is a good way to represent/learn information about system call sequences?
  - Issues: compactness, accuracy, performance, …

Early Research

Forrest et al [1999] compared several methods for learning system call sequences

- Memorize subsequences of length N (N-grams)
- Markov models
- Data-mining (using RIPPER)

N-grams found to be most appropriate

- Markov models provided a slight increase in accuracy, but incurred much higher overheads

Illustration of N-gram Method

```
1.  S0;
2.  while (..) {
3.      S1;
4.      if (...) S2;
5.      else S3;
6.      if (S4) ... ;
7.      else S2;
8.      S5;
9.  }
10. S3;
11. S4;
```

3-grams learnt:

- S0 S1 S2
- S1 S2 S4
- S2 S4 S5
- S4 S5 S1
- S5 S1 S3
- S1 S3 S4
- S3 S4 S2
- S4 S2 S5
- S2 S5 S3
- S5 S3 S4

Sample execution:

- S0 S1 S2 S4 S5 S1 S3 S4 S2 S5 S3 S4
- S0 S3 S4

- **S0 S3 S4**

Drawbacks of N-gram Method

Number of N-grams grows exponentially

- N must be small in practice (N=6 suggested)
- Implication: difficult to capture long-term correlations
  - `S0  S3  S4  S2` never produced by program, but all of the 3-grams in this sequence are

Remembers exact set of N-grams seen during training --- no generalization

- necessitates long training periods, or a high rate of false alarms

Models without Length Limitations

- **Finite-state automata**
  - Even an infinite number of sequences of unbounded length can be represented
  - Naturally capture program structures such as loops, if-then-else, etc.
- **Extended finite-state automata**
  - FSA + a finite number of state variables that can remember event arguments
- **Push-down automata**
  - By capturing call-return info:
    - PDAs are more accurate than FSM
    - Models are hierarchical and modular:
      - Hierarchical nature facilitates presentation
      - Smaller program models
      - Reuse of models for libraries
  - Extend PDAs to incorporate variables

Model extraction approaches

Static analysis [Wagner and Dean]

- Pros: conservative
- Cons:
  - difficult to infer data values, e.g., file names
  - difficult to deal with libraries, dynamic linking, etc.
  - overly conservative
    - for intrusion detection, can detect only attacks that are outside of the semantic model used for analysis
    - specifically, buffer overflows, meta character attacks, etc

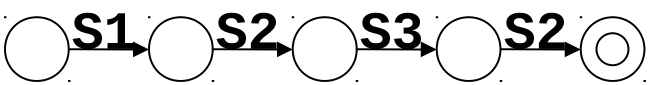
Machine learning by runtime monitoring

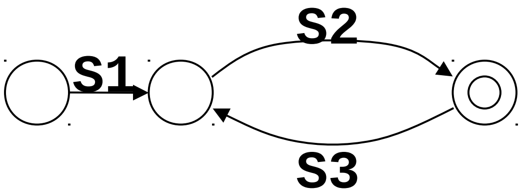
- Pros:
  - can detect a much wider range of attacks
  - can deal with libraries, dynamic linking
  - inferring data values is easier
- Cons:
  - False positives

Difficulty in Learning FSA from Strings

Strings do not provide any information about internal states of an FSA

- given S1 S2 S3 S2, which of the following FSA should we use?





•what is the criteria for determining the "better" FSA?

•even if we can answer this, the answer will depend on additional examples

  - e.g., sequences S1 S2 and S1 S2 S3 S2 S3 S2 will suggest that the second FSA is the right one

- Learning FSA from sequences is computationally intractable [Kearns & Valiant 89, Pitt & Warmuth 89]

Learning FSA Models: Graybox Techniques

Key insight:

For learning program behaviors, additional information can be used to simplify the problem:

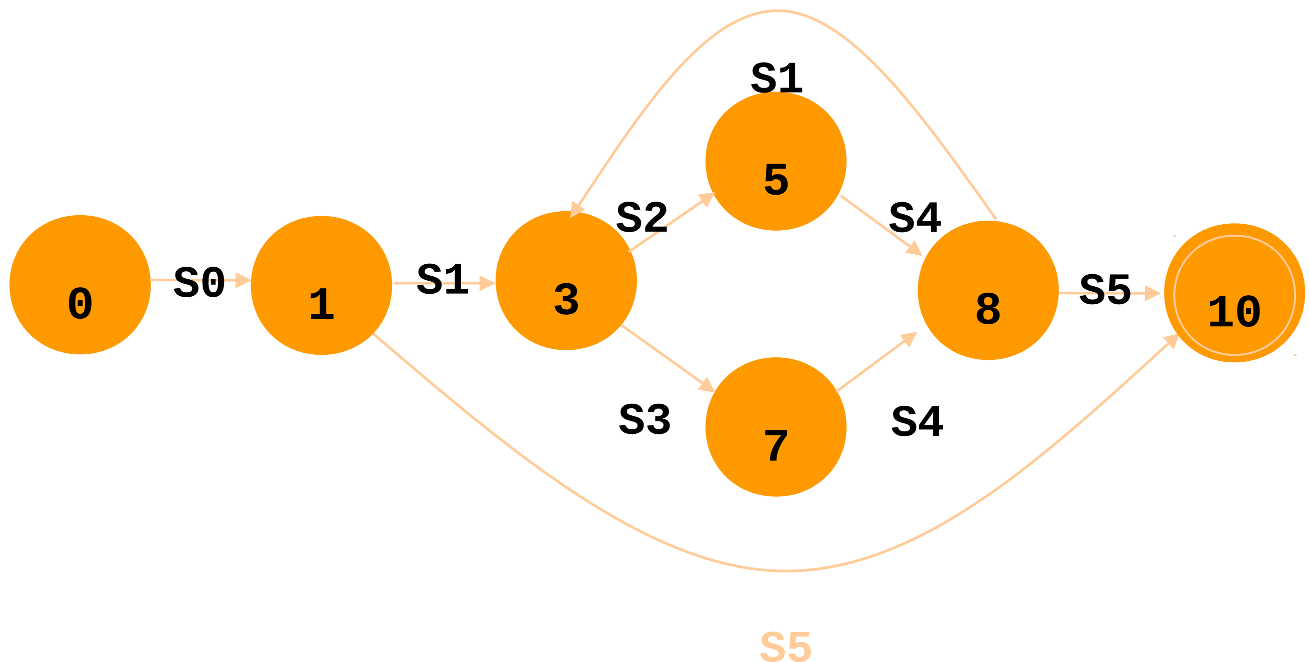
exploit program counter value to obtain state information

Learning FSA Models

A sample intercepted program behavior:

(S0,1) (S1,3) (S2,5) (S4,8) (S1,3) (S3,7) (S4,8) (S5,10)

```
1: S0;
2: while (…) {
3:   S1;
4:   if (…)
5:     S2;
6:   else
7:     S3;
8:   S4;
9: }
10: S5;
```



Approach Details

Interception of system calls using ptrace (Linux)

- same mechanism used by Forrest and other researchers

Examine process stack to obtain program counter information

Shared libraries and ASLR pose a problem

- same function may be loaded at different locations during different runs
- Solution: use program counter value corresponding to the code calling the dynamically loaded library
- Side benefit: ignoring library behavior makes FSA more compact

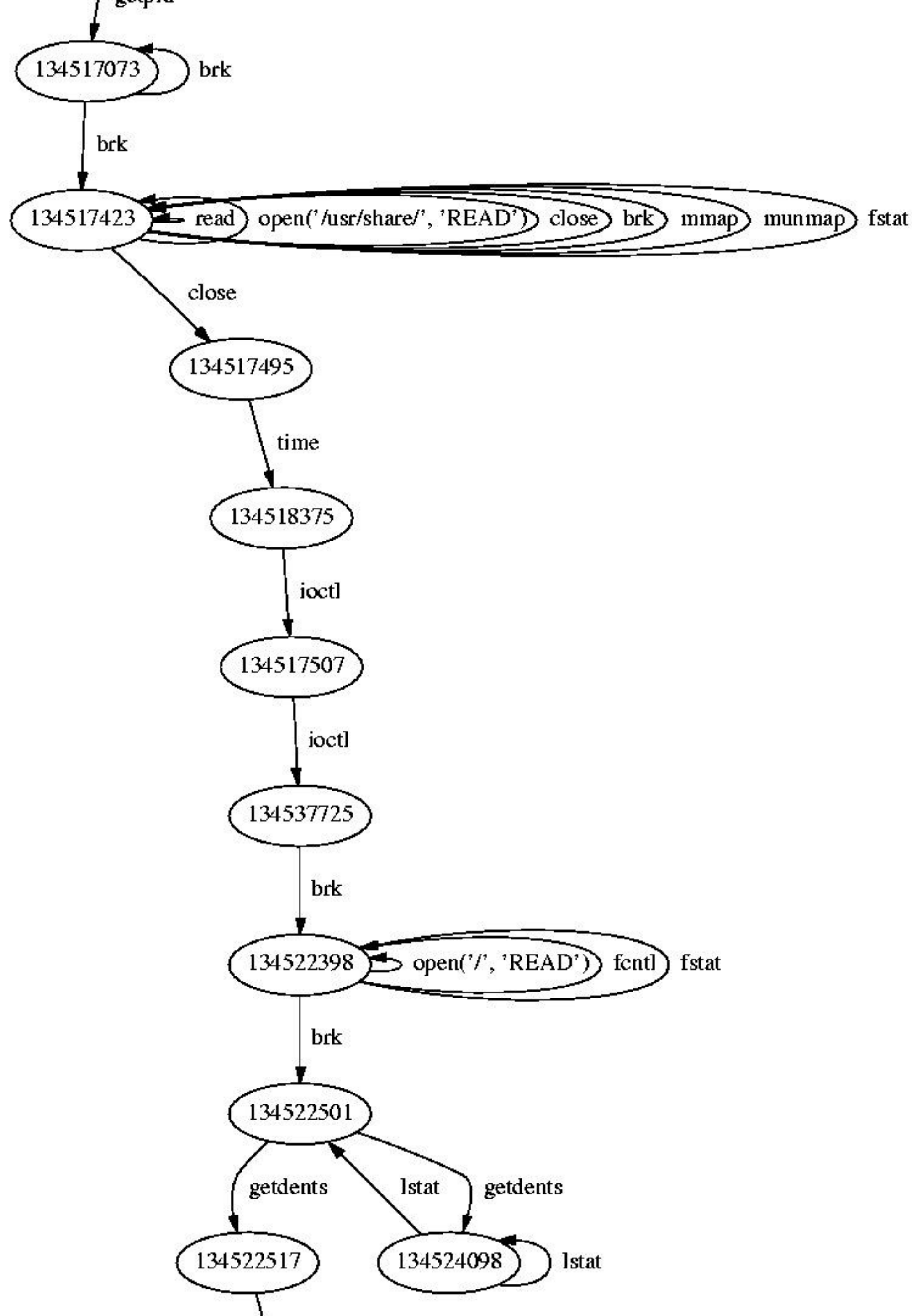
Approach Details (Continued)

Fork: Parent and child monitored with same FSA, but process contexts maintained

Exec: typically, a new FSA for the execve'd program is used.

Detection time

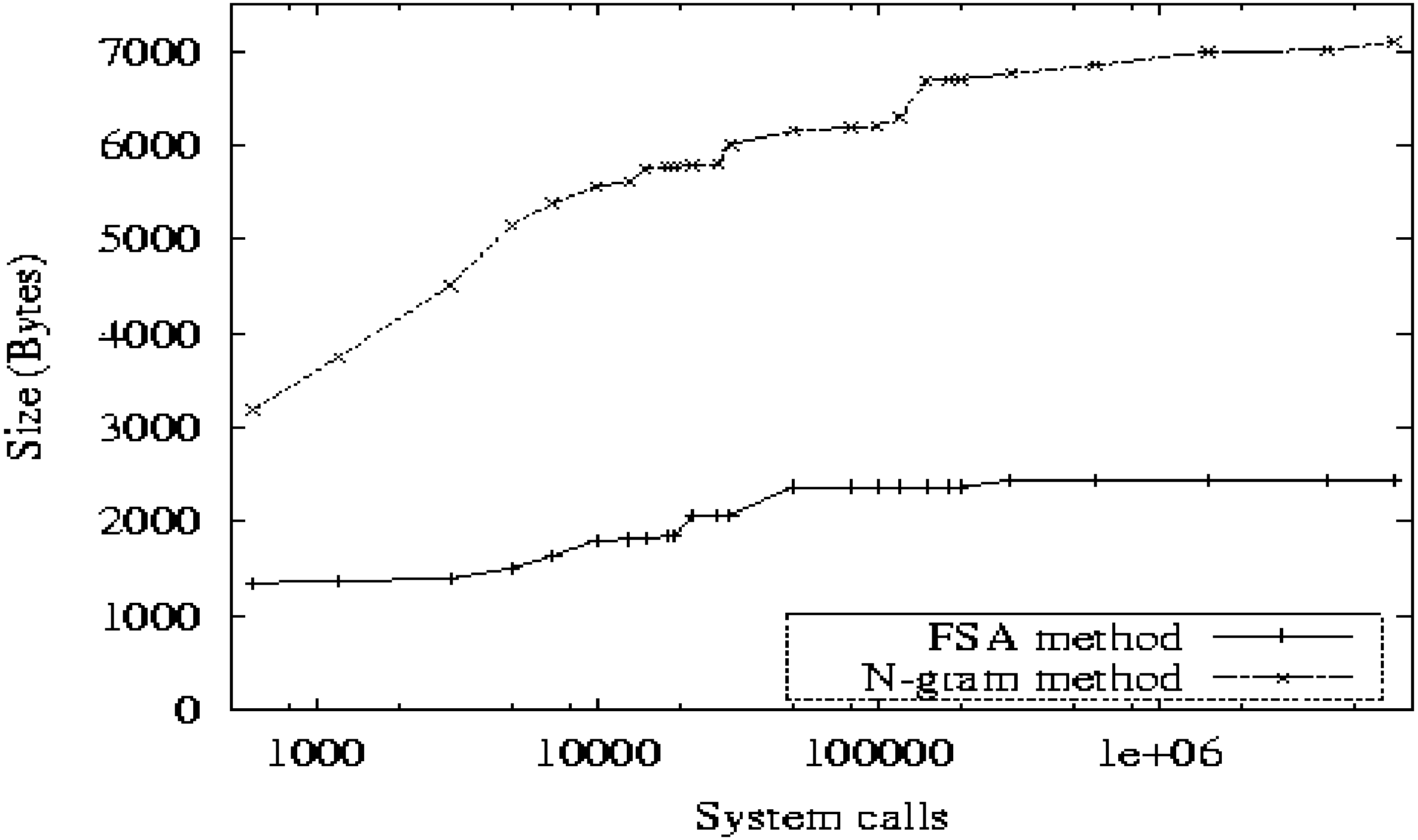
- mismatch may occur in terms of either the system call or program location
- use leaky bucket algorithm for aggregation
- program counter helps resynchronize even after observing behavior not seen during training


getpid
134517073
brk
brk
134517423
read
open('/usr/share/', 'READ')
close
brk
mmap
munmap
fstat
close
134517495
time
134518375
ioctl
134517507
ioctl
134537725
brk
134522398
open('/', 'READ')
fcntl
fstat
brk
134522501
getdents
lstat
getdents
134522517
134524098
lstat

Training Convergence

FSA method converges faster than N-grams

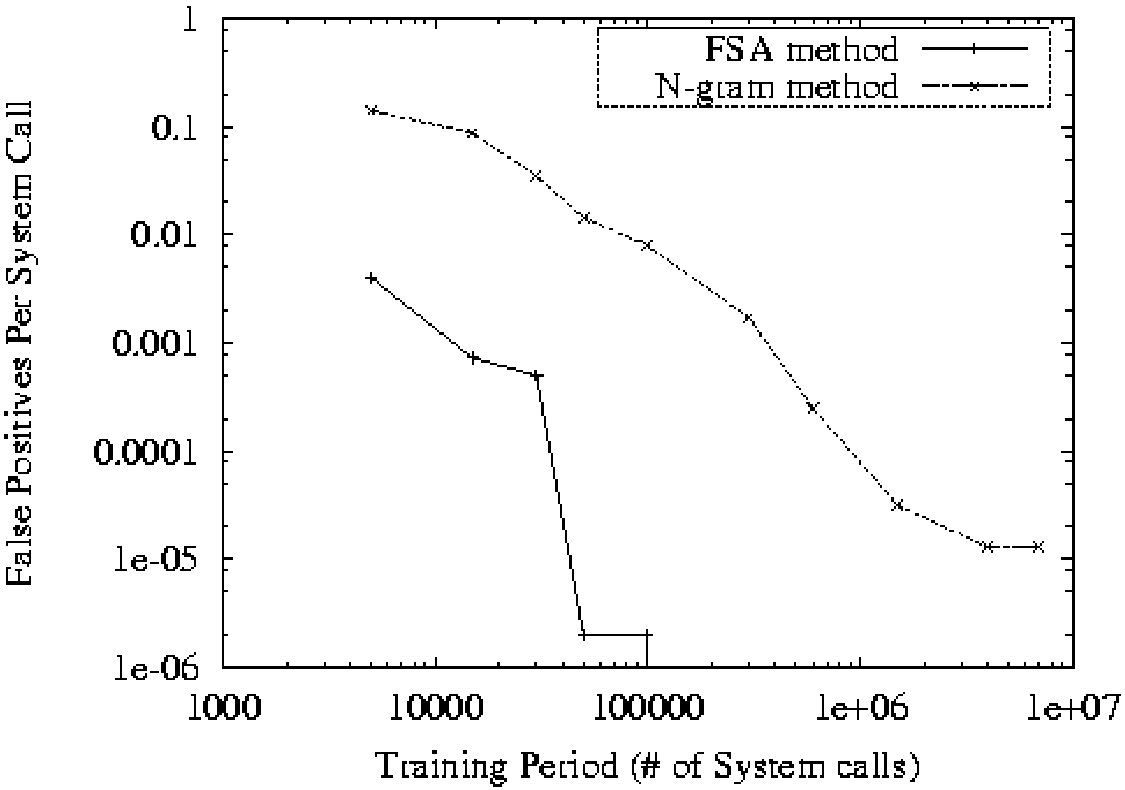
- roughly speaking, FSA method can do with roughly an order of magnitude less training period than N-gram method



False Positive Rate

FP results are similar to convergence

- for a given FP rate, FSA method requires an order of magnitude less training than N-gram method



Extracting PDA models

Basic idea: Examine the entire call stack, not just the most recent return address

- FSA technique already does this partially to discard library behaviors

This enables call/returns to be identified

- Not all calls/returns captured, since our visibility is limited to system-call invocation points

[Gao et al 2004] develops such a PDA technique

[Feng et al 2003] developed an alternative stack-based model called VtPath

Giffin et al develop similar techniques, but based on static analysis rather than runtime learning

PDA Vs FSA models

- **FSA models don't capture call-returns accurately**
  - They are both represented as "goto" transitions
  - Resulting model admits behaviors where a function invoked from one program location can return to any other call site
    - "Impossible Path Problem"
- **Inaccuracies due to libraries**
  - Libraries are essentially "inlined" at their call site
    - All system calls become "self-loops" at call site, causing accuracy loss
    - Behavior of a library function has to be learnt independently at each call site – impacts convergence
- **Use of "wrapper" functions poses a problem for FSA technique**

Mimicry Attacks

Attacks crafted with knowledge of IDS

Execute only system call sequences that would be permitted by the model

A mimicry attack can be developed from an attack sequence by inserting "junk" system calls that make it appear as if a legitimate sequence is generated

- Junk system calls made possible by using bad system call arguments

Graybox IDS complicate mimicry attacks due to the need to fake call site

- Control does not return to attack code after a call!

But can still be made to work

Known mimicry attacks based on memory corruption+injected code

Learning System Call Arguments

Earlier methods focus on control-flows

- System call sequences (N-grams)
- Automata models of behavior
  - FSA or PDA, with transitions labeled with system calls
- System call arguments largely ignored

Detects usual control-hijack attacks

Don’t detect most attacks that modify resources access by a system call

- Non-control data attacks
- Race condition attacks
- Mimicry attacks
- …

CSE 509

Course Summary

Cryptography Basics

- **Algorithm Vs Key**
- **Symmetric key ciphers (DES, AES, …)**
  - Block vs stream ciphers
- **Public key techniques (RSA, …)**
- **Hash functions (MD5, SHA, …)**
- **Random number generation**
- **Applications**
  - Encryption (Block vs Stream Ciphers)
  - Key generation
  - Authentication
  - Digital signatures
  - Certificates

OS Security: User Authentication

- **Something you know (secret), have (badge, smartcard) or are (biometrics)**
- **Password-based authentication**
  - Storing encrypted passwords, Offline/online Dictionary attacks
  - Ease of remembering Vs guessing
  - Password theft and trusted path
  - Variants and Improvements
    - Master password (ssh, browsers, …)
    - One-time passwords
    - Multi-factor authentication
    - Visual passwords

OS Security: Authentication: Biometrics

- Fraud, insult rates
- Techniques
  - Handwritten signatures
  - Fingerprint
  - Iris
  - Face
  - Voice
  - Speech
- Use in identification Vs authentication

OS Security: Access Control

- **Discretionary Access Control**
  - Access control matrix
  - ACLs
    - UNIX permission model
  - Capabilities
    - Limited use in OSes
- **Trojan Horse and Mandatory Access Control**
  - MLS: Bell-La Padula, Biba models
    - Benefits and drawbacks
  - Domain and Type Enforcement
    - Benefits and drawbacks
    - SELinux
  - Clark-Wilson policy
  - Chinese wall policy
- **Delegation and trust management**

Principles of Secure System Design

- Least privilege
- Fail-safe defaults (default deny)
- Economy of mechanism (simplicity => assurance)
- Complete mediation (look out for ways in which an access control mechanism may be bypassed)
- Open design (no security by obscurity)
- Separation of privilege (similar to separation of duty)
- Least common mechanism (avoid unnecessary sharing)
- Psychological acceptability (onerous security requirements will be actively subverted by users)

Software Vulnerabilities: Memory Errors

- **Memory corruption exploits**
  - Stack-smashing
  - Heap overflows
  - Format-string bugs
  - Integer overflows
- **Exploit defenses**
  - Canaries
  - Separating control data from other data
  - Randomization
    - Address-space (absolute or relative address)
    - Data-space
    - Instruction-space
- **Preventing memory errors**
  - Definition of memory error
  - Spatial vs Temporal Errors
  - Spatial error defenses
    - “Smart” pointers
    - Out-of-band metadata
    - Jones/Kelly
    - CRED
  - Temporal errors
    - Can be addressed using garbage collection (where feasible)

Injection Vulnerabilities

- **Example attacks**
  - SQL injection
  - Command injection
  - XSS
  - Path traversal
  - Format string bugs
  - Memory corruption/code injection attacks
- **Defenses**
  - Static taint analysis
  - Runtime fine-grained taint-tracking
  - Taint-aware policy enforcement

More Software Vulnerabilities …

- **Browser attacks**
  - XSS
  - CSRF
- **CWE-25**
- **File-name based attacks**
  - Symlink attacks
  - TOCTTOU attacks
    - How to succeed in races …

Program Transformations for Security

- **General idea**
  - Maintain additional metadata, check policies using this
- **Source-to-source transformations**
  - Guarding techniques
  - Absolute and Relative-address randomization
  - Full memory error detection
  - Fine-grained taint-tracking

Program Transformation on Binaries

- **Key challenges compared to source code**
- **Static rewriting**
  - disassembly techniques and challenges
  - rewriting challenges
- **Dynamic translation**
  - Dynamo Rio, Valgrind, Qemu, Pin, …
  - How it achieves speed
- **Applications**
  - Program shepherding
  - Taint-tracking
- **Issues and limitations**

Static Analysis for Vulnerability Detection

- **Techniques to identify potential bugs and vulnerabilities**
- **Requires a model of what is good behavior, or bad behavior**
  - "Good behaviors" are typically application specific, and hard to come by
  - "Bad behaviors" can be somewhat more generic
    - Common software vulnerabilities
      - Buffer overflow, SQL injection, …
    - Inconsistencies
      - Access check or locking on some program paths, but not others

Static Analysis

- **Usually require source code**
  - Binary code analysis will be quite limited due to absence of type and bounds information, as well as higher level control structures
- **Most program properties are undecidable**
  - Static analysis has to approximate in order to terminate. Approximation means that analysis can be sound or complete, but not both.
  - Sound: Guaranteed to find all vulnerabilities
  - Complete: No false positives

Static Analysis

- **Techniques**
  - Data-flow analysis
  - Abstract interpretation
  - Type infererence
  - Model-checking
- **Key challenges**
  - False positives and negatives
  - Range of properties
  - Scalability
    - How long does it take? How much memory does it need?
    - State-space explosion

Dynamic Analysis

- **Manual testing**
- **Random testing ("fuzz testing")**
  - Vulnerabilities often arise due to insufficient testing and optimistic assumptions about input
  - This means that incorrect inputs will cause unexpected behaviors
  - Random input will typically cause crashes
    - Using a debugger or other means, hackers can find additional information to turn the crash into an exploit
- **Coverage-guided fuzzing**
- **Manually assisted fuzz testing**
  - In many cases, random inputs don't work, as they get discarded very early
    - Most of the code is not exercised
  - Better to ensure that some parts of input are valid, so as to traverse more program paths
    - Remaining parts of input can be fuzzed

Directed Random Testing

- **“Intelligent” approach that chooses inputs to ensure more coverage**
  - Often based on some form of symbolic execution
    - Variables left unbound
    - As conditions are tested, constraints on unbound inputs are gathered, depending on whether “then” or “else” clause is taken
    - When multiple conditions are present on the value of a variables, use a constraint solving procedure to narrow down the range
  - Key challenges
    - Range of constraints that can be handled
    - state-space explosion
    - Many approaches choose to bind variables to concrete values when faced with these problems
- **Penetration testing**
  - Just another name for dynamic vulnerability testing

Malicious Code

- **Current threat environment: Profit-driven crime**
- **Types**
  - Viruses
  - Worms
  - Spam
  - Phishing
  - Botnets
  - Rootkits
  - Spyware
  - DDoS
  - Extortion
  - Cyberwar

Malicious code: Stealth Techniques

- **Stealth and Obfuscation**
  - Behavioral obfuscation
    - Anti-virtualization and anti-analysis techniques
    - Trigger-driven
  - Code obfuscation
    - Control-flow obfuscation
    - Data obfuscation
    - Encryption and packing
    - Polymorphism
    - Metamorphism

Untrusted code defense

- **Untrusted code implies strong adversary, requires correspondingly strong defenses**
  - Mechanisms
    - System-call interception
      - –Techniques and trade-offs
    - Inline-reference monitors
      - –Issues, challenges
      - –Software-based fault-isolation
    - RISC and CISC
      - –Control-flow integrity
- **Defenses**
  - Sandboxing (confinement policies)
    - Policies are hard to write!
    - Indirect attacks!
    - Example: Native Client
  - Isolation
    - Virtual machines
      - –VMware, Xen, KVM, Qemu
    - One-way isolation
      - –With copy-on-write
    - Complete isolation
      - –Smart phones
  - Information flow mediation
    - Vista (one-way)
    - MLS (two-way)

Isolating untrusted code: Virtual machines

- **Process Vs Namespace Vs System virtualization**
- **Type I and Type II VMMs**
- **Paravirtualization Vs full virtualization**
- **Implementation techniques**
  - Binary translation, paravirtualization, hardware-assisted virtualization
- **Memory virtualization**
- **Security applications**
  - Honeypots, sandboxes, malware analysis, high-assurance Vms
  - Protection from compromised OS

Untrusted code: Java, Javascript and Browser Security

- **Safe languages**
  - Java
  - Type safety, byte-code verification, loader checks
  - Sandbox model, stack inspection, doPrivileged
  - Javascript
  - Type safe language, better integration with browser, security based on removing OS access
- **Browser security**
  - HTTP protocol (GET/POST), cookies, authentication
  - HTML forms, parameters, server-side processing
  - Same origin policy, application to scripts, frames, network reads; Ajax and XmlHttpRequests
  - Reflected and persistent XSS; XSS vectors and defenses
  - Other injections (HTTP headers, ...)
  - CSRF and defenses

Side-channel attacks and physical security

- **Covert channels**
  - Intentionally embedded
  - Implicit flows, timing, steganographic techniques, ...
- **Side channel attacks**
  - Timing analysis, power monitoring
  - Differential fault analysis
  - Emanations (keyboard, power, screen/camera/microphone, shock sensor)
  - Remanence
- **Physical layer attacks and tamper resistance**
- **Trusted platform, software attestation**

Side-channel attacks and physical security

- **Covert channels**
  - Timing, implicit flows, DNS requests, ...
- **Side-channels**
  - Execution time

Intrusion Detection

- **Network intrusions**
  - Protocol attacks (Teardrop, Synflood, Smurf, …)
  - DDoS
  - Botnets
  - Reflection attacks
  - Worms
- **Attack stages**
  - Probing
  - DoS
  - Privilege escalation

Intrusion Detection

- **False positives and negatives**
- **Observation points:**
  - Host-based Vs Network intrusion detection
    - Benefits and drawbacks
- **Techniques**
  - Anomaly detection
  - Misuse detection
  - Specification-based detection
- **Algorithms**
  - Pattern-matching
  - Machine learning

Host-based Intrusion Detection

- **Models**
  - Strings, finite state automata, PDA, …
  - FSA based technique (using system calls)
- **Evasion: Mimicry attacks**
- **Dataflow Vs Controlflow models**