CSE509 : Computer System Security

Intrusion Detection

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

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 (false alarm ratio)
  - 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;
```

Sample execution:

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

- 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

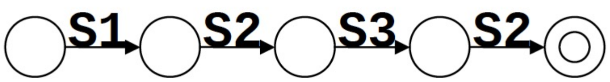
- Finite-state automata
  - Even an infinite number of sequences of unbounded length can be represented
  - Naturally capture program structures such as loops, if-then-else, etc.
- Extended finite-state automata
  - FSA + a finite number of state variables that can remember event arguments
- Push-down 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

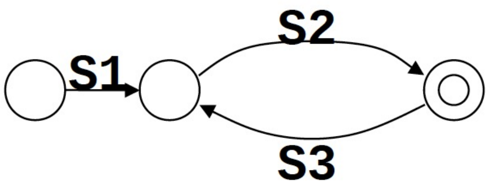
Model extraction approaches

- 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:

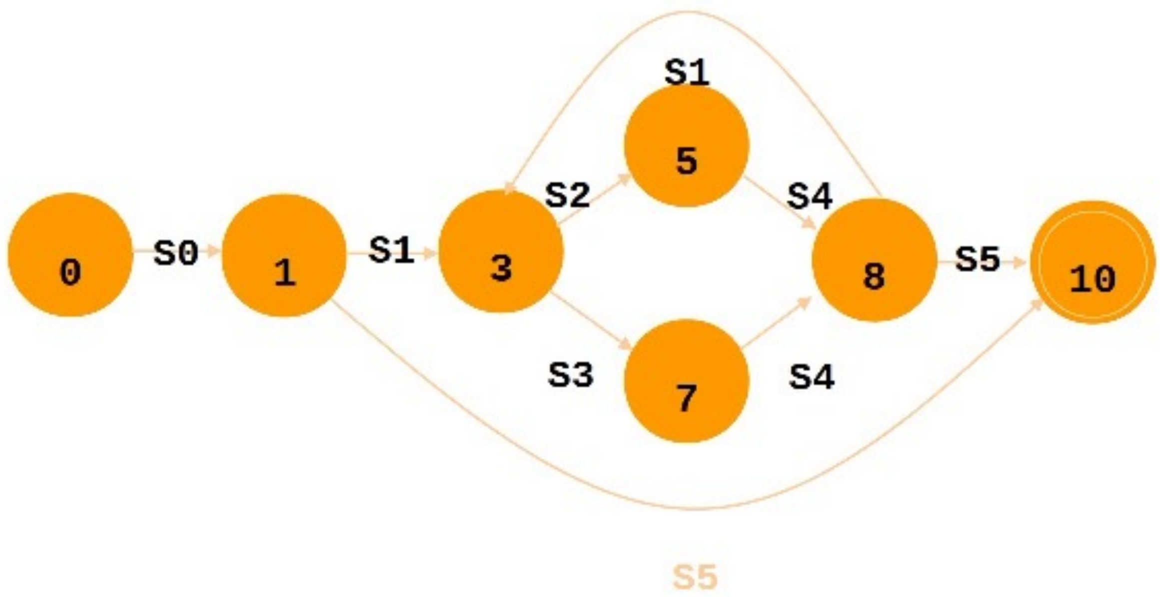
> For learning program behaviors, additional information can be used to simplify the problem:
>
> exploit program counter value to obtain state information

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
- Dynamic linking poses 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

Questions?