Detecting Computer Attackers: Recognizing Patterns of Malicious, Stealthy Behavior

Robert K. Cunningham

Richard P. Lippmann, Nick A. Plante,

Seth E. Webster, Harry M. Wolfson, Marc A. Zissman

MIT Lincoln Laboratory

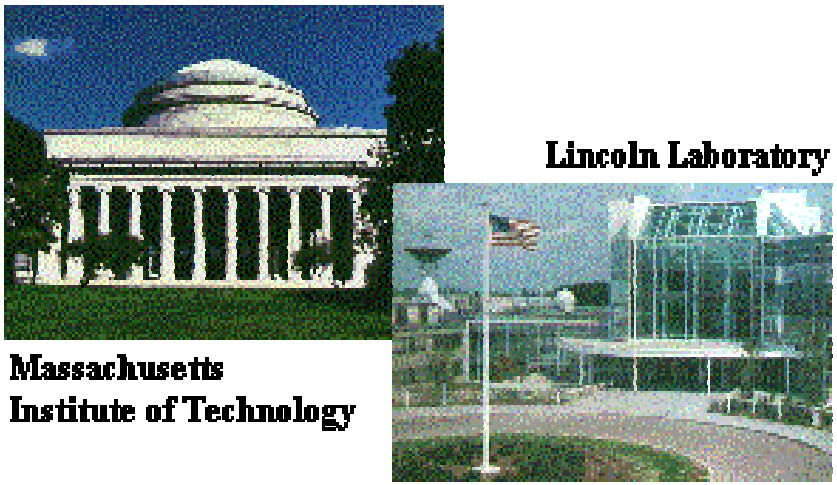
Presentation to

CERIAS

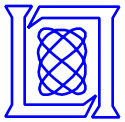
29 Nov 2000

This work was sponsored by the Department of the Air Force under contract F19628-00-C-0002. Opinions, interpretations, conclusions, and recommendations are those of the authors and are not necessarily endorsed by the United States Air Force.

MIT Lincoln Laboratory



- Federally-funded research and development center (FFRDC)
- Major research areas: electronics, control, communications, radar
- About 2200 MIT employees, including 1100 technical staff (scientists/engineers)
- Technical staff: 50% Ph.D., 25% M.S., 25% B.S.
- Sponsors: 80% DoD, 15% FAA, 5% other
- Budget: Roughly $325M, about 30% of MIT's budget
- Web site: www.ll.mit.edu

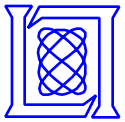


Information Systems Technology Group

- 60% Information Assurance Technology
  - Intrusion Detection R&D for wired and wireless networks
  - Security Tools Accuracy Evaluations
  - Distributed Simulation Systems

- 40% Speech and Language Technology
  - Speaker/Language Identification
  - Language translation
  - Inter-lingual information extraction

- Personnel: 33 Technical Staff
  - 3 Group Leaders
  - 5 Senior Staff
  - 24 Associate/Assistant or regular staff
  - 2 Paid Consultants
  - 4 Students



Outline

- Motivation
- Computer attacks
  - Who performs them, and what are they after?
  - Distributed Denial-of-Service Attacks
- Detecting attackers *after* an attack is launched
  - Signature Matching Technique
  - Bottleneck Verification Technique
  - Evaluation
- Detecting attackers *before* an attack is launched
  - Motivation and Requirements
  - Approach
  - Feature Extraction
  - Evaluation
- Summary and Challenges

Motivation

- Information systems are critical to the functioning of nations
  - Repositories for critical data (e.g., financial, military)
  - Control of infrastructure (e.g., power, cable, inventory, personnel)
- But these systems are vulnerable!
- As these systems are more heavily relied upon, their vulnerability becomes more of a concern

> We live in an age when one person sitting at one computer can come up with an idea, travel through cyberspace and take humanity to new heights. Yet, someone can sit at the same computer, hack into a computer system and potentially paralyze a company, a city or a government.
>
> - President Clinton, 1/7/2000, announcement of three-year cyberspace defense plan

What is Information Assurance?

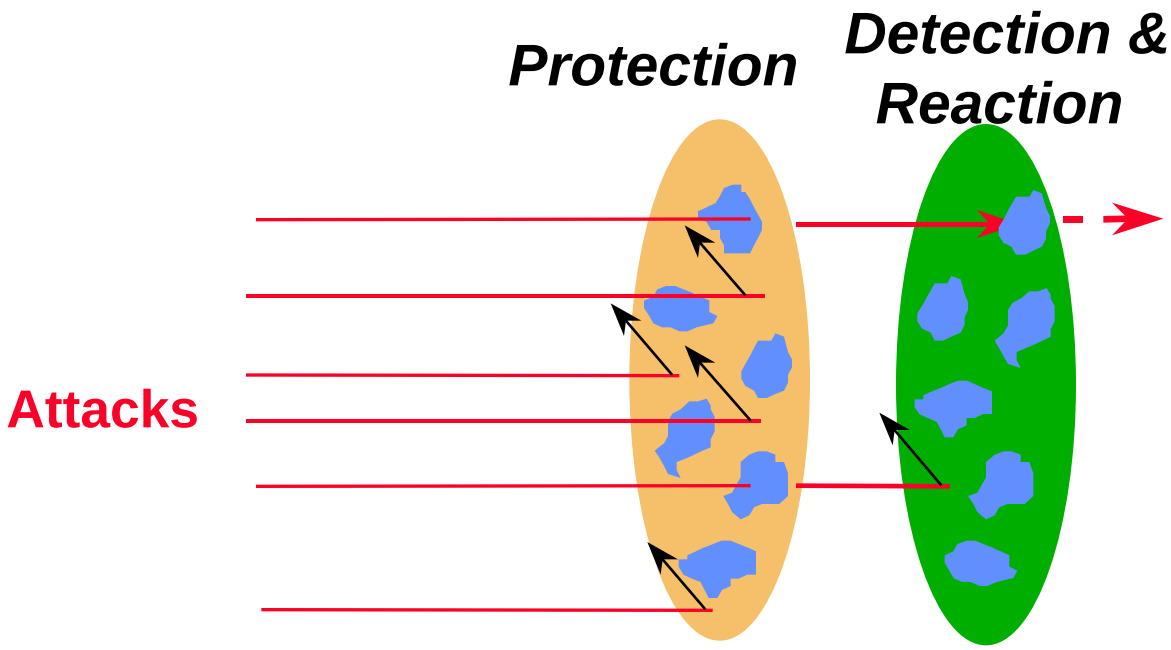
- Information Assurance is the process by which one ensures the
  - Integrity
  - Confidentiality
  - Availability

  of information assets

  (computers, networks, processes and data)

- Information Assurance must be done all of the time

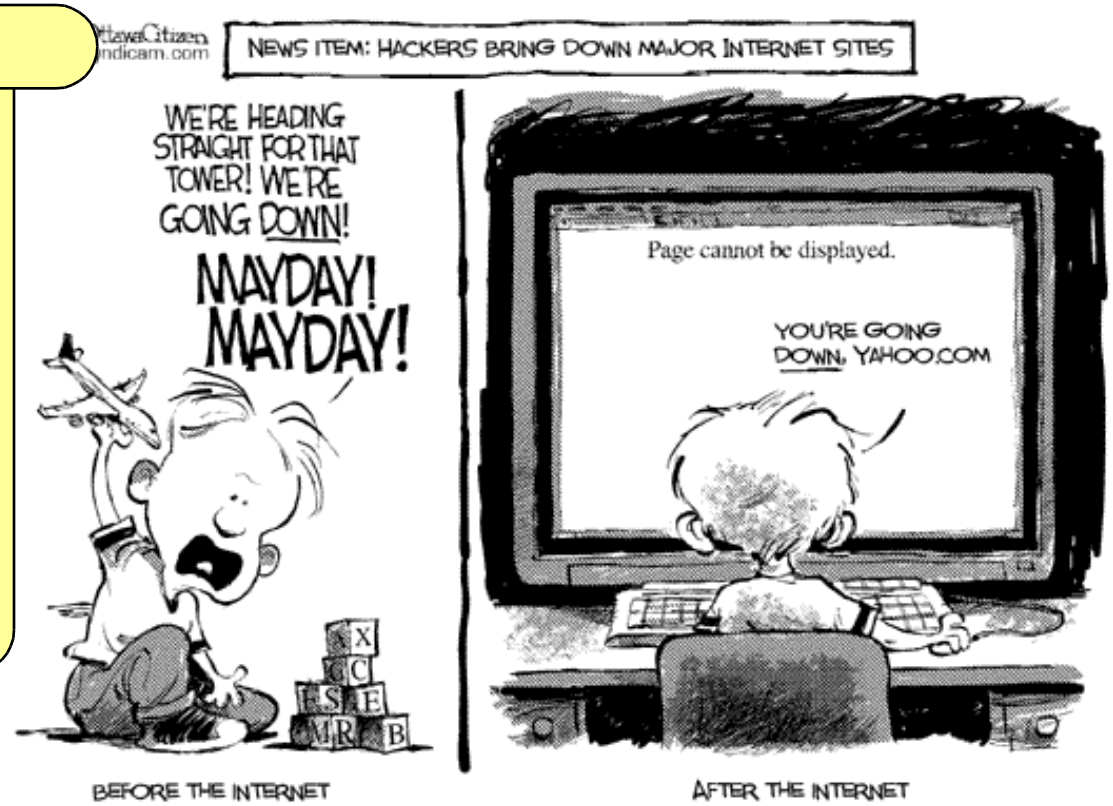
Components of Information Assurance: Protection, Detection and Reaction



- Protect against known attacks and vulnerabilities
- Detect (and react to) attacks where defenses have failed

Who are the computer attackers, and what are they after?

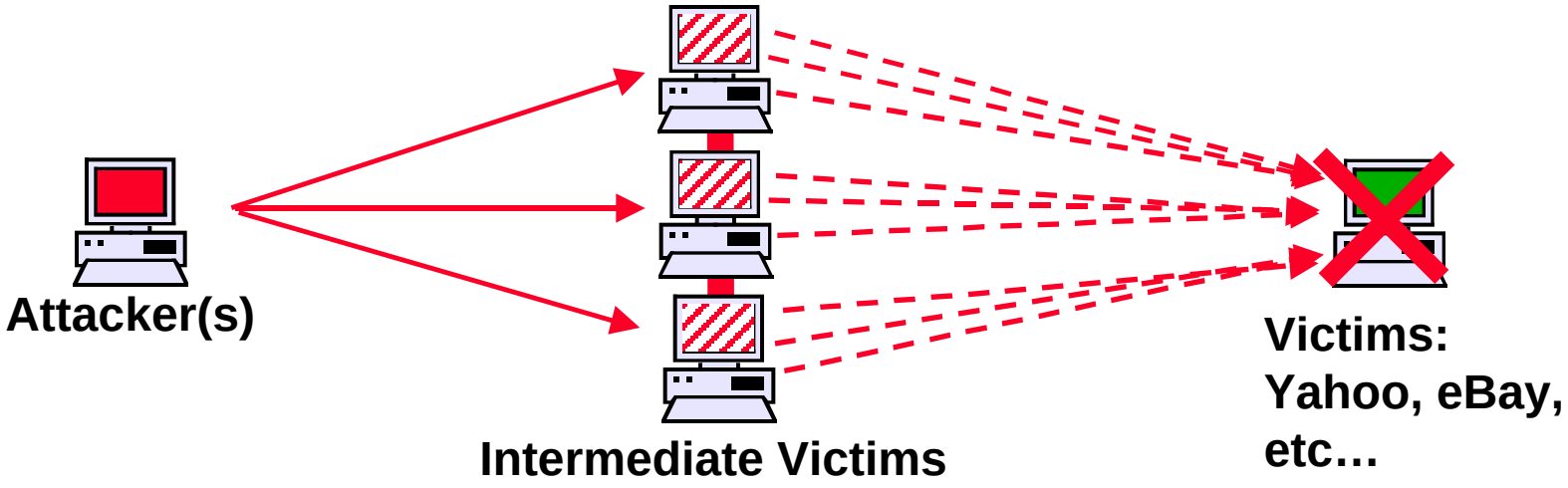
The CIA says it sees growing signs that countries such as Russia and China are developing tools to attack commercial computer networks at the heart of U.S. economic might.
- Reuters, 2/23/2000



- **Attackers vary in their skills and motivation**
  - **Adversaries attacking economic or military infrastructure from safe havens**
  - **Teenagers who are experimenting with others' computers**
- **But it's hard to determine who initiated an attack**

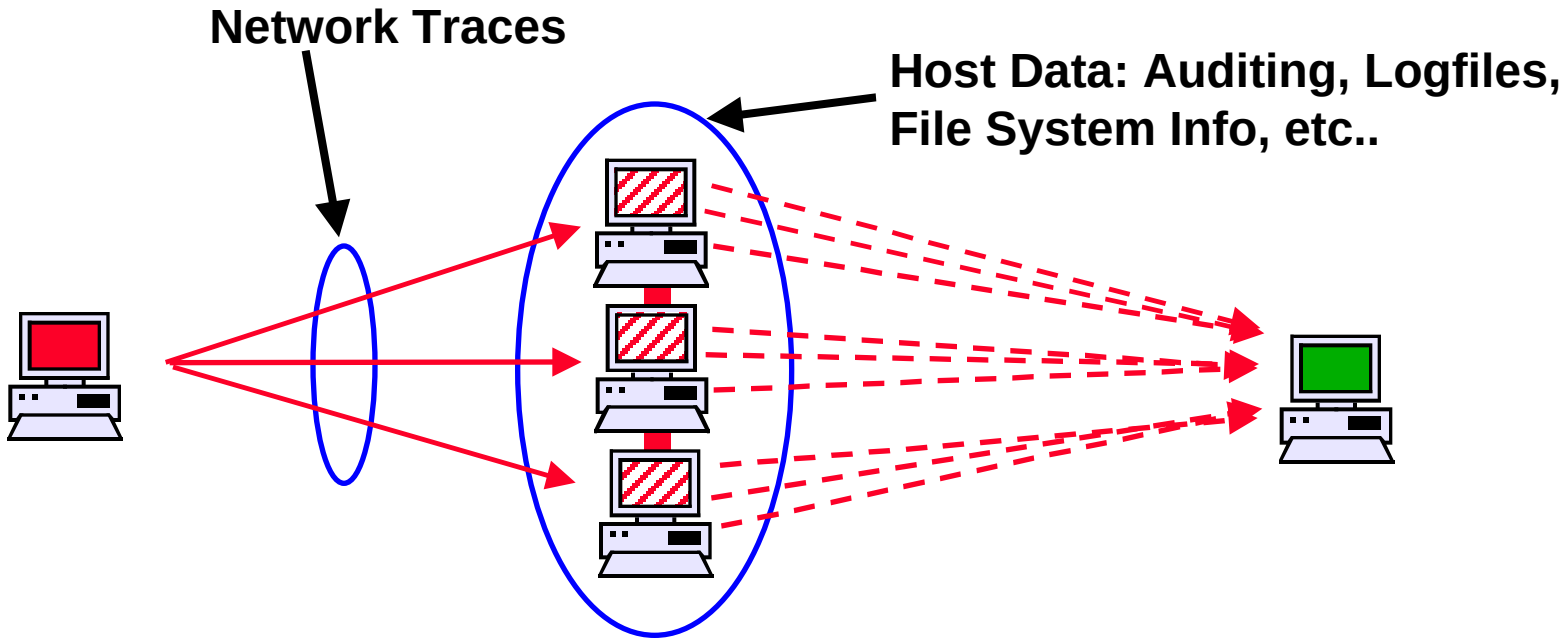
Recent Attack Scenario: Feb 2000 “DDOS”

- Attacker(s) compromised hundreds of intermediate victims which were used in a coordinated packet flood attack on eCommerce sites



Attack type: “DDOS” – Distributed denial of service

Preventing and Detecting the DDOS Attack

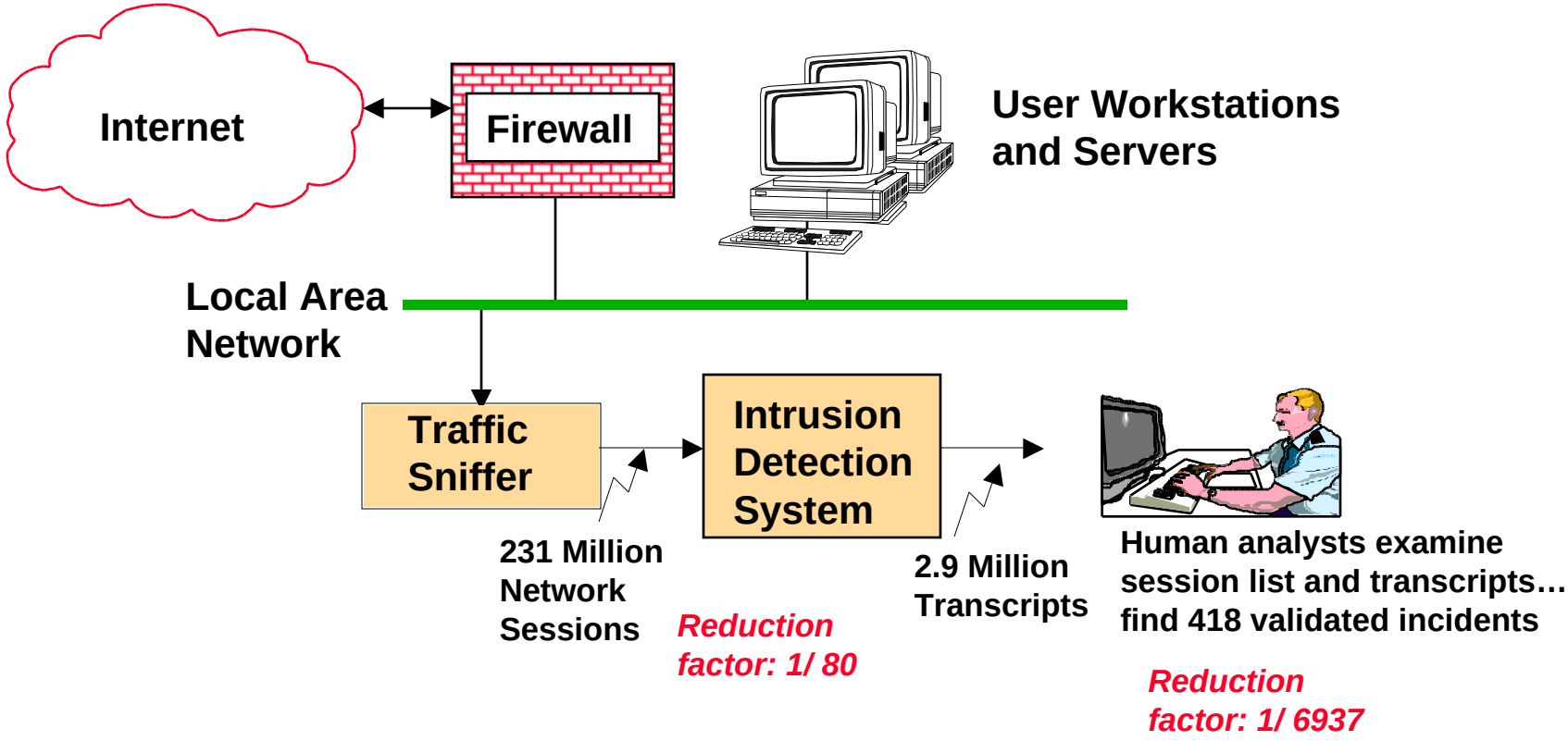


- Attack could be prevented if early stages of probing and software installation are detected (and if known security flaws are corrected)
- Intrusion detection systems analyze many data sources to discern normal traffic/actions from malicious traffic/actions in order to detect and stop attacks

One Type of Conventional Intrusion Detection: Signature Spotting

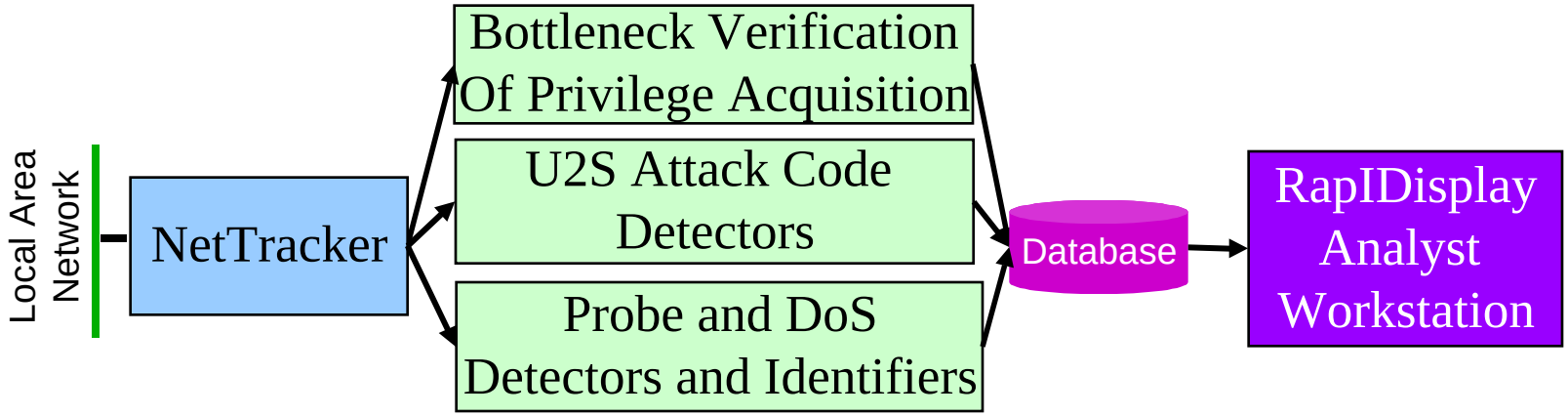
- Approach: Spot suspicious signatures in the network traffic
- Typical signatures for Unix systems
  - `Login incorrect`
  - `finger`
  - `passwd`
  - `.rhosts`
- Problems with signature systems
  - Signatures are chosen by experts with limited regard for impact on false alarm rate
  - No use of context around the signature occurrence
  - Need advance knowledge of the signatures, so can't detect new attacks
- Deployment status: in use at hundreds of military installations world-wide as well as many commercial establishments

Data Volume Example
>100 Military Bases: 1998



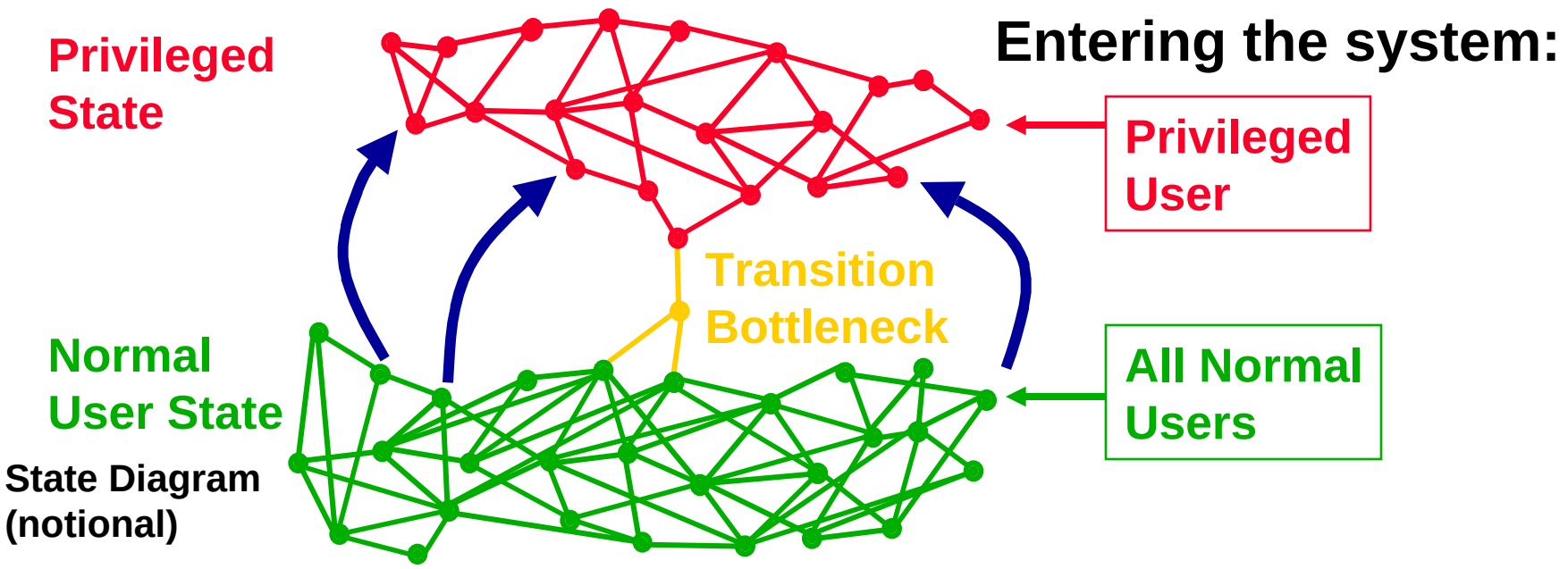
- Transcripts created only for sessions with high warning values
- Many connections (231M), fewer transcripts (2.9M), and very few serious incidents (418)
- Critical need for improved intrusion detection accuracy

Macroscope
Functional Diagram



- **Sensor: NetTracker - Robust packet assembly and statistics**
- **Algorithms**
  - **Bottleneck Verification - detection of attacks that acquire privilege, regardless of method**
  - **Attack Code Models - detection of code to acquire privileges**
  - **Probe and DoS Detect/Ident - detection of attacks that map the network and absorb its resources**
- **Database - easy access/retrieval of information**
- **RapIDisplay - clear display of attack information**

Model-based Intrusion Detection: Bottleneck Verification

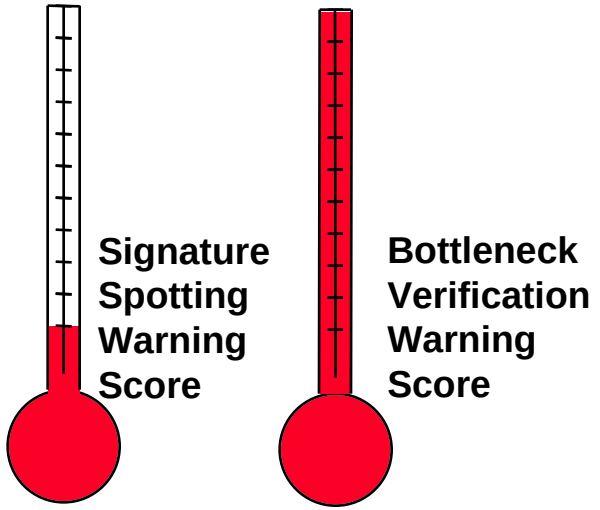


- Bottlenecks are desirable for system security
- Well-designed systems have one bottleneck for transitioning from normal to privileged access
- Key Idea: Detect bottleneck bypasses
- *Need not know all possible ways to evade bottleneck, so new attacks can be found!*

Real Air Force Session Returning Attacker

```
login: mismith
Password: banana1

liontamer% cd .netscape
liontamer% ./ksh
# csh
csh: Permission denied
# mail
No mail for root.
#
```



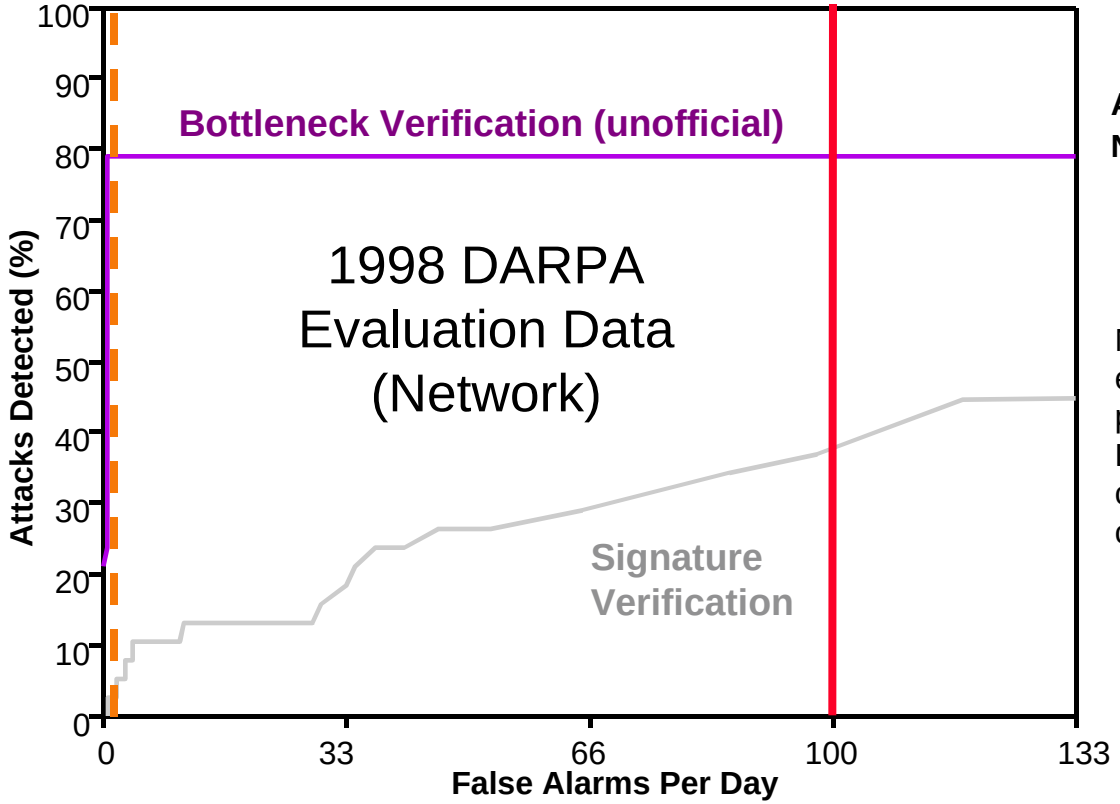
- User types nothing suspicious, so signature spotting score is low
- Bottleneck verification sees that the user has obtained root privileges

1998 DARPA Intrusion Detection Evaluation Simulation Network Overview

1000's Hosts, 100's Users
Normal and **Attack** Traffic

- UNIX Workstations
- CISCO Router

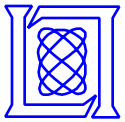
Detecting Local Users Illegally Becoming Root



Attacks: 38
Normal: 660,049

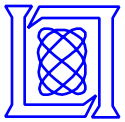
Note: this is not a blind evaluation, as the data is produced by MIT Lincoln Laboratory. Do not compare with other DARPA contractor results.

- **Bottleneck Verification is better at finding attacks and rejecting false alarms**
- **At 1 False Alarm/day, BV detects 79%; Signature 2.6%**
- **At 100 False Alarms/day, BV detects 80%; Signature 38%**



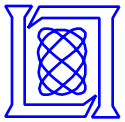
Earlier Detection is Better

- **Bottleneck Verification does a terrific job of finding an attack *after* it has been launched**
- **But, it would be nice to detect someone preparing to attack**
- **Detecting attacks in source code is a first step**
  - **Many features present in source code are also present in compiled code, but feature extraction is easier for source code**
  - **If desirable, features could be extracted from executables**



What Type of Attack Source Code?

- **Probes, denial-of-service and remote-to-local attacks can be developed and executed remotely, so defender may not see code in any form**
- **Trojans are of interest, but limited examples (at time of initial work, many examples recently)**
- **User-to-superuser attacks are launched on a system that is being defended**
  - **Could see source code, compilation, executables**
  - **Defender controls the system environment and could monitor all incoming data**
  - **We know Unix well, so start there**

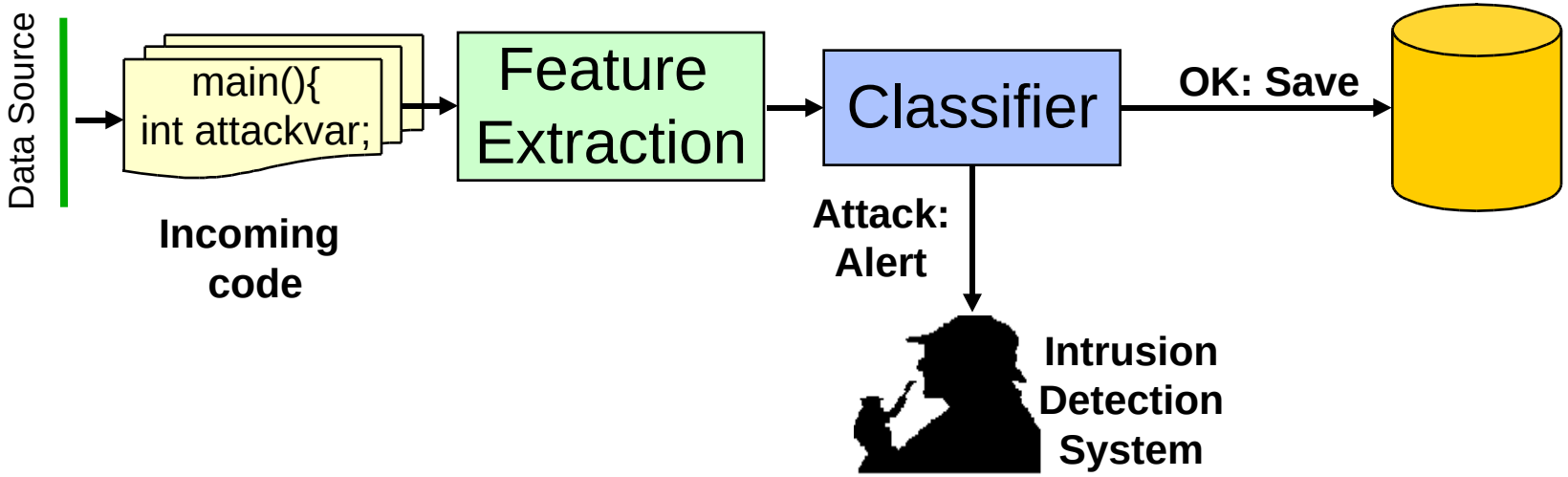


Requirements

- Select features that will generalize across attack examples
  - But lightweight feature search--no detailed analysis
  - Single pass is best; perhaps single pass with preprocessor support

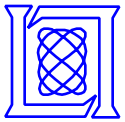
- Use technique that can be improved with additional data, but avoid “one attack gets one feature” approach

Approach: Distinguishing U2S Attack from Normal Code



- Count selected attack-like operations
- Use pre-trained classifier to label attack-like code
- Detect/Prevent attack-like code from infecting host
- Alert Intrusion Detection System

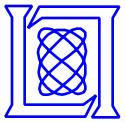
- Key elements
  - Selecting the correct attack-like operations
  - Training a classifier to generalize across attack classes



Data Set 1: Shell Software

- **Attack Software (20 examples: 19 bash,ksh,sh; 1 csh)**
  - **All Unix attacks at rootshell.com from Jan 1, 1998 to April 1, 2000 (6 files)**
  - **All Unix attacks at anticode.com (6 files)**
  - **Misc. Unix attacks from web from 1996 onward (8 files)**

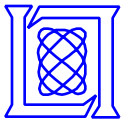
- **Normal Software (382 examples)**
  - **SHELLdorado (http://www.oase-shareware.org) (192 files)**
  - **RedHat Linux 6.1 scripts in init.d, rc[0123S].d (134 files)**
  - **Portable Shell Programming (Bruce Blinn) scripts (32 files)**
  - **Learning the BASH Shell (Newham and Rosenblatt) (13 files)**
  - **Learning the Korn Shell (Rosenblatt and Loukides) (11 files)**



Data Set 2: C Software

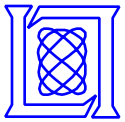
- **Attack Software (73 examples)**
  - **All Unix attacks stored at rootshell.com from Jan 1, 1998 to April 1, 2000 (18)**
  - **All Solaris 2.5.1-2.7,x86,General/RedHat Linux, SCO, and AIX attacks at anticode.com (41)**
  - **BugTraq exploits available between April 1 and June 1 (14)**

- **Normal Software (>1700 files)**
  - **Filesystem tools (fileutils-4.0)**
  - **User interaction tools (shell-utils-2.0)**
  - **Process control (bash-2.04, gdb-4.18)**
  - **Network interaction (sendmail-8.10.0, apache-1.3.12)**
  - **Lexical analysis (flex-2.54)**
  - **Window manager interaction (emacs 20.6)**



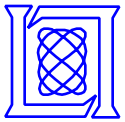
Feature Extraction

- Goal: represent all equivalent actions in one feature
- But, languages have many ways to express same operation
- For now, perform one-pass scan of a single file
  - Separate comment sections from code sections
  - Ignore static call graph



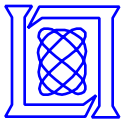
Language Independent Features

- Comments: “sploit, exploit, vulnerab”
- File or directory creation or deletion
  - Link, mkdir
  - unlink, rmdir
- File permission modification
  - setuid, setgid
  - chown, chgrp
- Process initiation: exec, system and variants



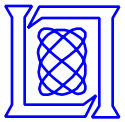
Shell Language Specific Features

- **Embedded Low-level language**
  - **Assembly using the asm directive**
  - **Lines of “C” code**
- **Accessing passwd, shadow password file**
- **Accessing a core file**
- **Calling the shell in interactive mode**
- **Creating a shared library**
- **Using touch to set date/time**
- **Altering local security**
  - **.rhosts, /etc/hosts (performing the alteration)**
  - **Localhost, 127.0.0.1 (accessing the host)**



C Language Specific Features

- **Embedded low-level language**
  - **Using the asm directive**
  - **Using arrays of executables stored as data**
- **Adding passwd, shadow password entry**
- **Using ptrace**
- **Access environment variables**

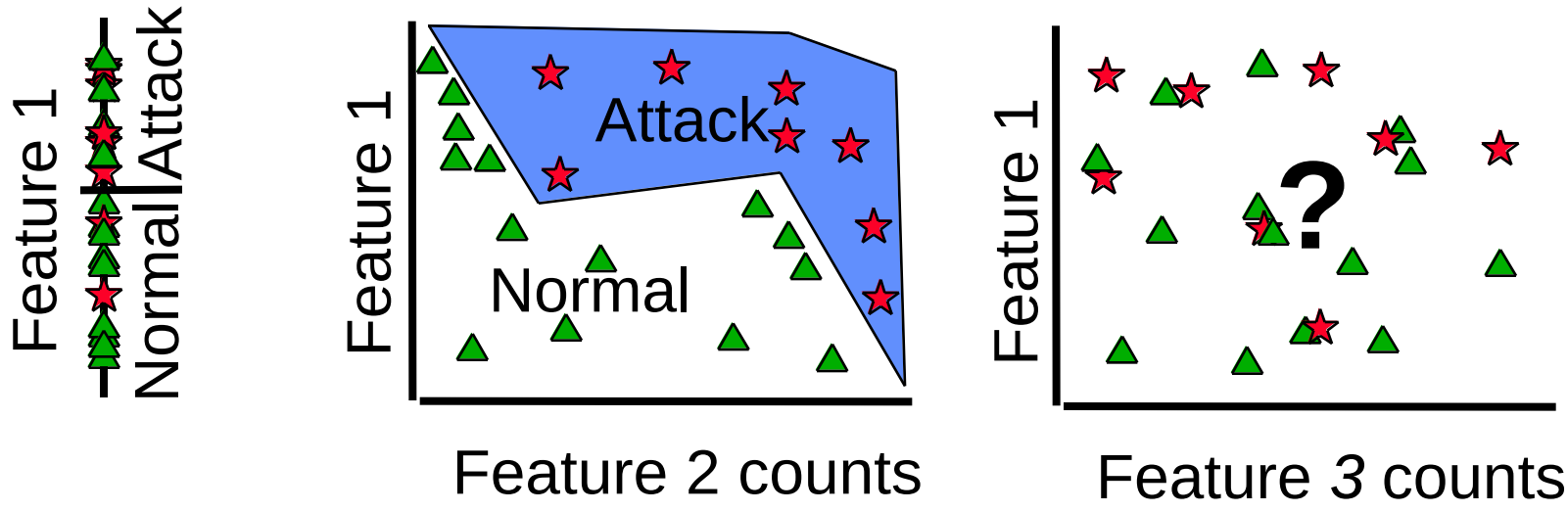


Feature Scanning Implementation Details

- Example feature patterns (as perl regexp)
  - Simple: File Link feature
    - Shell: \b((?<!un)link|ln)\b
    - C: \b((?<!un)link)\b
  - Complicated: Detecting set user/group id
    - Sh:\bchmod\b.*?(((u|g|ug|gu)[+=]\S*?s)|(?:\d*?[4567]\d\d\d)))
    - C: \bset(e?uid|e?gid)\b
    - Differences: shell affects file, whereas C affects running program

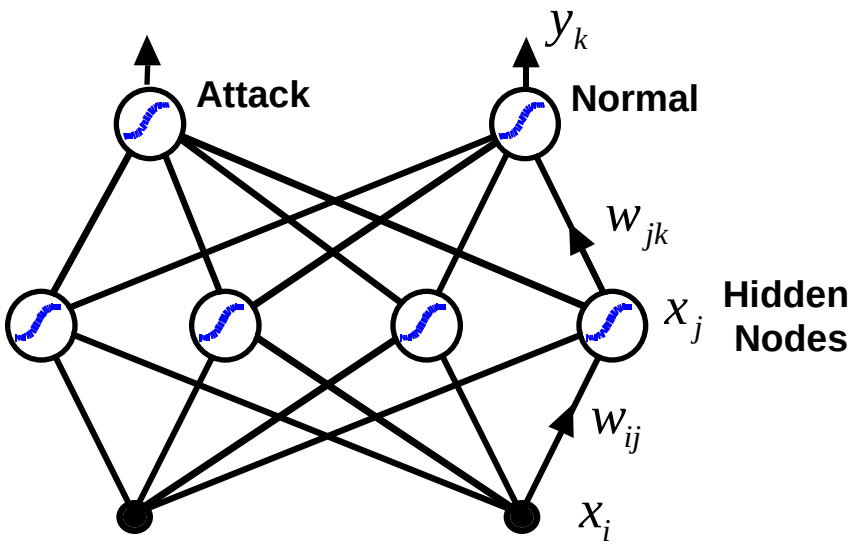
- Observations from developing feature patterns
  - Different mechanisms to accomplish similar things are counted together
    - link and ln; chmod numeric and symbolic permissions
  - Common use of macros in C makes feature selection hard without preprocessor support
  - Feature patterns could be targeted to specific architetures

Accurate Labeling of Source Requires Selecting Best Features



- **Signature combinations can be better than isolated features**
- **Some *combinations* separate attacks from normal data, others don't**
- **Classifiers learn a label for a given *combination* of features**
  - **Some classifiers also provide posterior probabilities**
  - **We can use this to select the best features…**

Feature Selection Process: Classifier



- Two-layer perceptron
  - K input nodes, feature input normalized to 0 mean, std dev=1
  - 2K hidden nodes
  - 2 outputs

- Train weights *w* using backpropagation of errors with weight updating after each training pattern

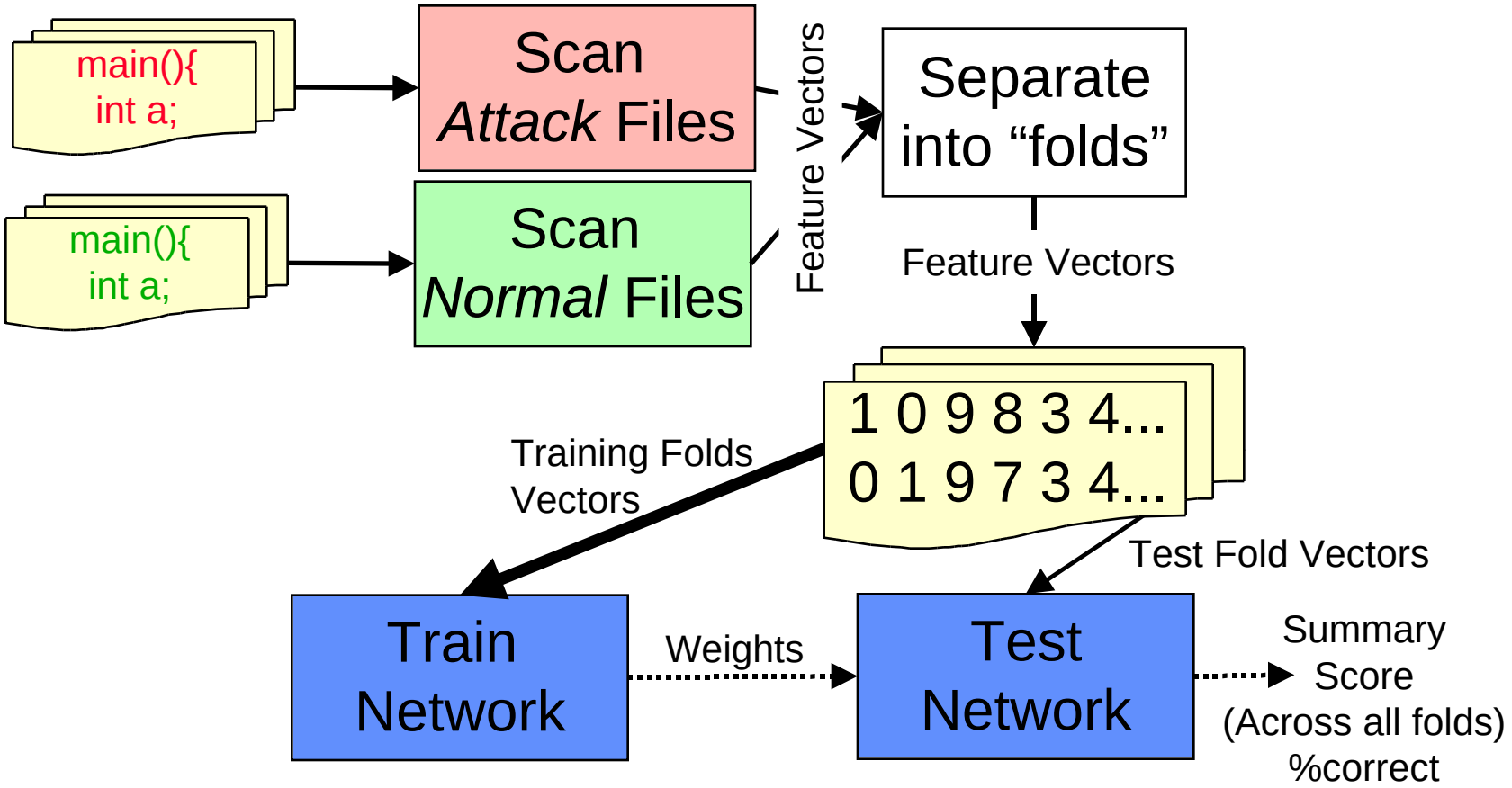
Feature Selection Process: Algorithm

- Improve detection of attacks by adjusting class a-priori probabilities to 70% attacks and 30% normal data

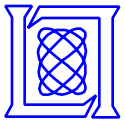
- Train network with each feature, select best single feature
- Add a second feature, select best two features
- Continue until no more improvements

- Train and test using 20-fold cross-validation
  - Subdivide data into 20 samples
  - Use 19 for training, 1 for testing
  - Cycle through training/testing groups to determine how well feature generalizes on all of the data

Feature Selection Overview



- **Train and test on *different* data to understand how well approach can generalize**



Results: Shell Source Code

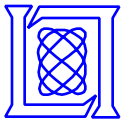
- With comments, best performance with 2 features
  - Comment about exploit
  - Accessing a core file
  - 20/20 detections, 1 false positive (382 Normals)
- Without comments, best performance with 7 features
  - Making a file set-user/group-id
  - Accessing a core file
  - Modifying .rhosts or /etc/hosts
  - Creating a shared library object
  - Copying a shell
  - Logging back into the local host
  - Creating a link
  - 17/20 detections, 1 false positive
  - Misses included csh attack, shell attacks with embedded C

- Note: each feature does NOT detect one attack; it is the combination of features that is important

Results: C Source Code

- With comments, best performance using only 2 features
  - Comment about exploit
  - Unlink command
  - 68/73 attacks detected, 4 false positives (>1700 Normals)

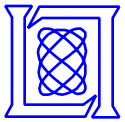
- But, Comments won't appear in executable
- Without comments, best performance used 2 features
  - setuid/setgid creation
  - Embedded _asm_ line
  - 53/73 attacks detected, 4 false positives



Spy vs. Spy

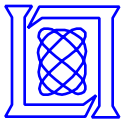
- To hide: exploit feature extraction shortcuts to
  - Reduce feature counts
    - Use alternative mechanism to accomplish the same action
    - Use subroutines
    - Put portions of software in different files
  - Increase feature counts
    - Insert `#ifdef 0` with useless code
    - Insert subroutines that are not called

- To respond: improve feature extraction
  - Expand feature pattern to include new technique
  - Perform static analysis of subroutines
  - Parse `#ifdefs` in addition to comments
  - Support cross-file analysis
  - Perform dynamic analysis of subroutines in sandbox



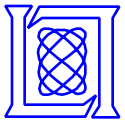
Deployment Options

- **Network Intrusion Detection System**
  - **Our network IDS (Macroscope) can scan incoming mail and ftp-data, looking for attacks**
  - **But data rates in these services are high, and traffic can be encrypted**
- **On each host**
  - **Periodic file system scans**
  - **Wrappers that perform a scan before making a file executable**
    - **Reworked chmod from shell**
    - **Chmod, open, creat library calls when executable file is closed**
- **This finds some attacks *before* they are launched; an IDS finds attacks *after* they are launched**
- **This is defense-in-depth!**



Summary

- **Substantial R&D efforts in intrusion detection research and other IA areas are underway**
- **Model-based intrusion detection algorithms work well**
  - **Bottleneck Verification allows one to detect attacks without prior knowledge of the attack**
  - **U2S Attack software source code can (mostly) be differentiated from normal software using neural networks**
- **Formal evaluations of such systems are being run, and these evaluations help guide the research**
- **But, these efforts are new, the problems are difficult, and attackers are working hard too...**



Some Current and Future Challenges

- Disruption of evidence collection
  - Bad guys: Encrypt data so that LAN sniffing is ineffective
  - Good guys: Move from sniffing to host-based audit logging
- Broadening the base of attack
  - Bad guys: Attack from multiple hosts over multiple sessions
  - Good guys: Correlate evidence across multiple hosts and sessions (inter-host communication, bottleneck verification)
- Social engineering and captured terminals
  - Bad guys: Get access to real passwords, masquerade as real users
  - Good guys: Look for seemingly legitimate but anomalous behavior
- Broadening the number of victim systems
  - Bad guys: Attack multiple hosts at the same time
  - Good guys: Correlate evidence across multiple targets