

# Machine Learning and Intrusion Detection

*Wenke Lee*

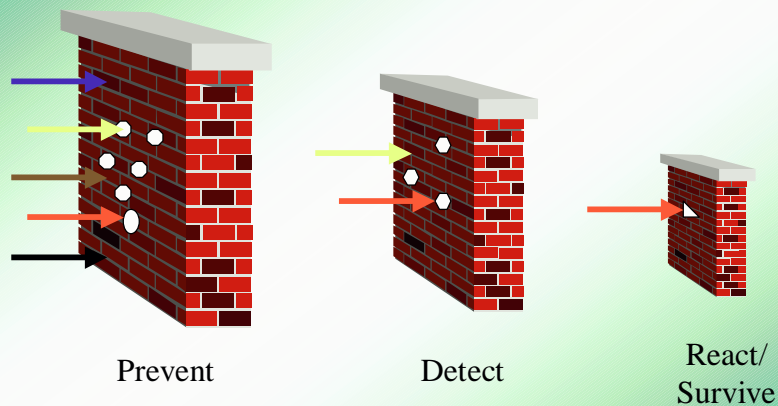
[wenke@cc.gatech.edu](mailto:wenke@cc.gatech.edu)

[\*http://www.cc.gatech.edu/~wenke\*](http://www.cc.gatech.edu/~wenke)

## What Is Intrusion Detection?

- Security in computing
  - Confidentiality, integrity, availability.
- Intrusion
  - Malicious activity aimed to compromise the security of computing and network resources.
- Intrusion detection
  - The process of identifying and responding to intrusion.

## Why Is Intrusion Detection Necessary?

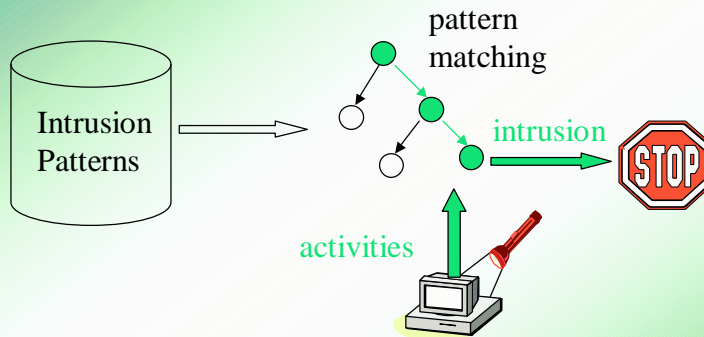


*Security principles: layered mechanisms*

## Intrusion Detection Is Hard

- Causes of intrusions
  - Design flaws, implementation errors, operational oversights, and *malicious intents*.
- Finding/removing bugs is hard, preventing/detecting intrusions is even harder
  - The adversaries are intelligent.
- Use audit logs
  - Capture *all* activities in network and hosts.
  - But the amount of data is huge!
- Can we automate?

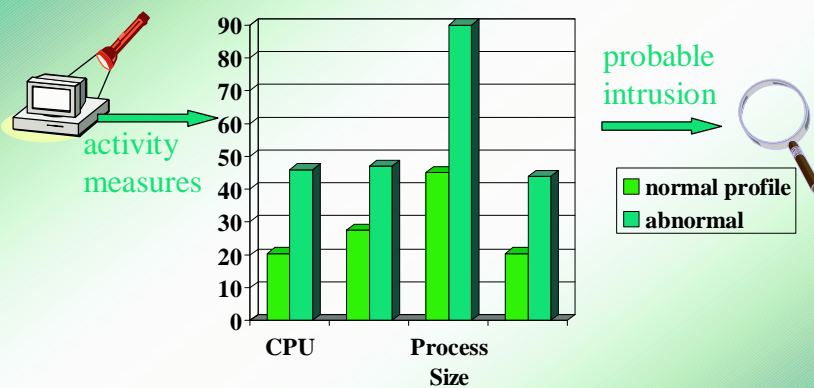
## Misuse Detection



Example: *if* (ip.src == ip.dst) *then* "land attack"

Can't detect new attacks

## Anomaly Detection



Relatively high false positive rate - anomalies can just be new normal activities.

## Building Intrusion Detection Models

- The most critical element of ID
  - High detection rate, low false alarm rate, efficient, etc.
- Manual
  - Analyze attack programs/behaviors
  - Slow and error-prone
- Automated
  - Apply machine learning algorithms to training data (normal and/or intrusion data)
  - Fast
  - Data-dependent ...

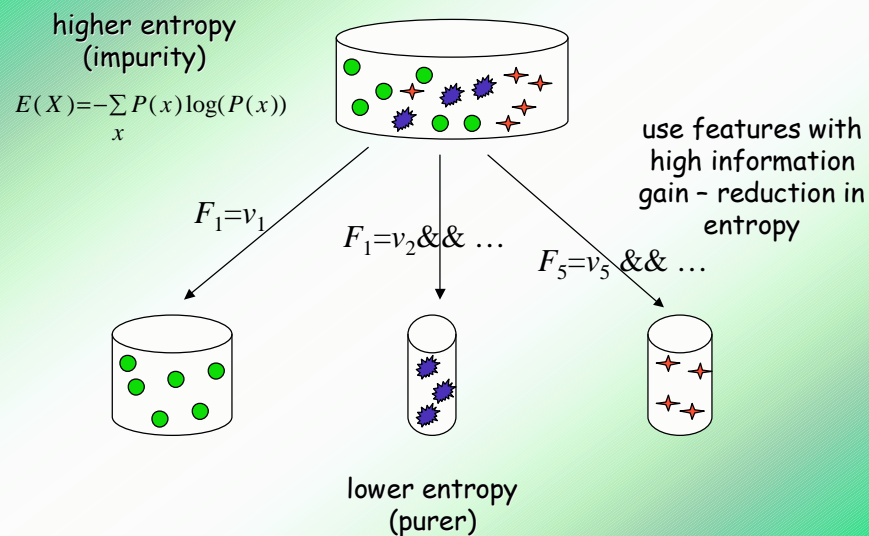
## ID as a Classification Problem

- Misuse detection
  - Learn rules to classify data into normal or one of the known intrusions or “unknown” (new normal or new intrusion).
- Anomaly detection
  - Learn rules to specify normal behaviors
    - e.g. what is normally the next system call (the class) given the previous n system calls (the features)
    - A large number of classification errors: anomaly.

## Classifiers as ID Models: Basic Requirements

- **Accurate**
  - Generalizable: detect variants (or a whole class) of attacks
- **Human readable:**
  - No black-box: “sanity” check
    - Eliminate stupid rules: due to artifacts of data
- **Efficient in (re-)training:**
  - Handle huge amount of data; fast
- **Efficient in execution**
  - Cost (model execution time) considered

## The Classification Process



## The Feature Construction Problem

*dst ... service ... flag*

h1	http	S0
h1	http	S0
h1	http	S0
h2	http	S0
h4	http	S0
h2	ftp	S0

syn flood

normal

*dst ... service ... flag %S0*

h1	http	S0	70
h1	http	S0	72
h1	http	S0	75
h2	http	S0	0
h4	http	S0	0
h2	ftp	S0	0

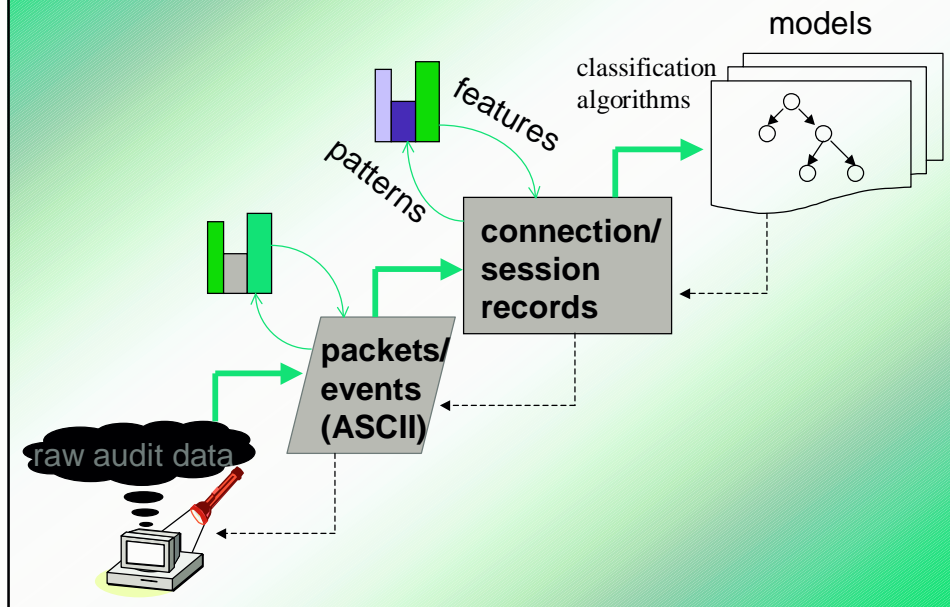
existing features  
useless

construct features with  
high information gain

How? Use temporal and statistical patterns, e.g., "a lot of S0 connections to same service/host within a short time window"

1. Compute patterns
2. Identify intrusion patterns
3. Construct features accordingly

## The Data Mining Process of Building ID Models



## RIPPER

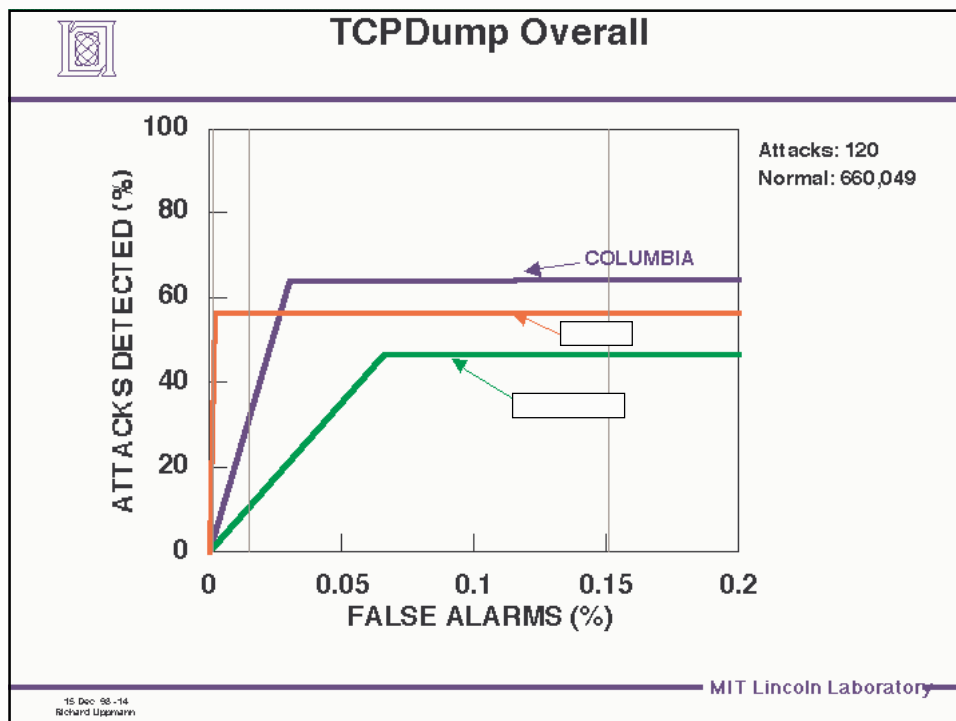
- By William Cohen.
- Outputs rule sets.
- Propositional rule learning algorithm
  - Performs efficiently on large noisy datasets
  - Overfit-and-simplify:
    - Grow: repeatedly adds conditions to a rule to maximizes information gain
    - Prune: repeatedly deletes conditions of a rule until its error rate goes up
    - Training data: growing set 2/3; pruning set: 1/3.

## Example RIPPER Rules for ID

- If-then-else rules:
  - pod :- wrong\_fragment  $\geq$  1, protocol\_type = icmp.
  - smurf :- protocol = ecr\_i, count  $\geq$  3, srv\_count  $\geq$  3.
  - ...
  - normal :- true.

## 1998 DARPA ID Evaluation

- The data (prepared by MIT LL):
  - Total 38 attack types, in four categories:
    - DOS (denial-of-service), e.g., syn flood
    - Probing (gathering information), e.g., port scan
    - r2l (remote intruder illegally gaining access to local systems), e.g., guess password
    - u2r (user illegally gaining root privilege), e.g., buffer overflow
  - 40% of attack types are in test data only, i.e., “new” to intrusion detection systems



## Efficient Execution of ID Models

- Reducing “operational cost”:
  - A multiple-model approach:
    - Build multiple rule-sets, each with features of different cost levels;
    - Use cheaper rule-sets first, costlier ones later only for required accuracy.

## A Multiple Model Approach: Details

- 3 cost levels for features:
  - Level 1: beginning of an event
    - Cost 1 or 5.
  - Level 2: middle to end of an event
    - Cost 10.
  - Level 3: multiple events in a time window
    - Cost 100.

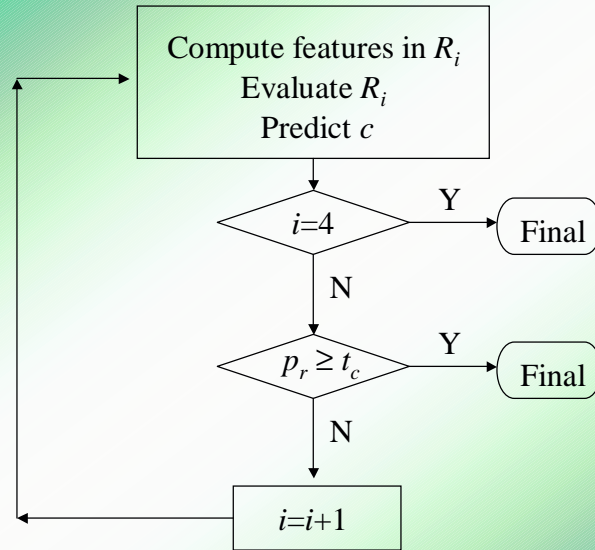
## A Multiple Model Approach: Details (continued)

- Ordering of a rule-set:
  - Ordered rule-set:
    - “**if**  $rule_1$  **then**  $class_1$  **elseif** ... **else**  $class_n$ ”;
    - Succinct rules, sequential checking;
    - Order class by +/- frequency
      - -frequency is more efficient: “normal” most frequent
  - Unordered rule-set:
    - “**if**  $rule_1$  **then**  $class_1$ ; ...; **if**  $rule_n$  **then**  $class_n$ ”;
    - Less efficient but more accurate rules; parallel rule checking.

## A Multiple Model Approach: Details (continued)

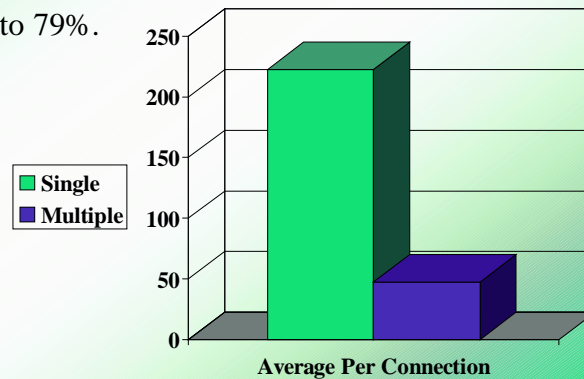
- 4 rule-sets with different feature cost levels:
  - $R_1$ : cost 1 features;  $R_2$ : cost 1 and 5;  $R_3$ : cost 1, 5, and 10;  $R_4$ : cost 1, 5, 10, and 100;
  - $R_{1-3}$ : unordered rule-sets;  $R_4$ : ordered;
  - $p_r$ : the precision of each rule  $r$  in  $R_{1-3}$ ;
  - $t_c$ : the precision threshold for each class  $c$ .

## Execution of Multiple Models



## Results

- Reduction of Operation Cost:
  - Compare the multiple-model approach with single-model approach;
  - $rdc\%: (single - multiple)/single$ ;
  - range: 57% to 79%.



**Thank You!**