Evaluating Software Sensors for Actively Profiling Windows 2000 Computer Users

Jude Shavlik
Mark Shavlik
Michael Fahland
Motivation and General Approach

- Identify \(\sim\) unique characteristics of each user/server’s behavior
- Every second, measure 100's of Windows 2000 properties
  - in/out network traffic, programs running, keys pressed, kernel usage, etc
- Predict \(\text{Prob}(\normal | \text{measurements})\)
- Raise alarm if recent measurements seem unlikely for this user/server
Goal: Choose “Measurement Space” that Widely Separates User from General Population
Initial Experiment

- **Subjects:** 10 users at Shavlik Technologies
  - Unobtrusively collected data for 6 weeks
  - 7 GBytes archived
- **Task:** Are current measurements from user X?
- **Initial Focus:** Keystroke data
  - Which key pressed?
  - Time key down
  - Time since previous key press
Training, Tuning, and Testing Sets

- **Very important in machine learning to not use testing data to optimize parameters!**

- **Train** Set: first two weeks of data
  - Build a (statistical) model

- **Tune** Set: middle two weeks of data
  - Choose good parameter settings

- **Test** Set: last two weeks of data
  - Evaluate “frozen” model
Our Intrusion-Detection Template

Last $W$ (window width) keystrokes

If $\text{prob(current keystroke)} < T$ then raise “mini” alarm

If $\# \text{ “mini” alarms in window} > F$ then predict intrusion

Use $tuning$ set to choose good values for $T$ and $F$
Alarm #1 - Probability We Estimate

\[
\text{Prob}(\text{current keystroke} = K3 \quad \text{and} \quad \text{previous keystroke} = K2 \quad \text{and} \quad \text{two-ago keystroke} = K1 \quad \text{and} \quad \text{time between } K2 \text{ and } K3 = \text{Interval23} \quad \text{and} \quad \text{time between } K1 \text{ and } K2 = \text{Interval12} \quad \text{and} \quad \text{time } K3 \text{ was down} = \text{Downtime3})
\]
Visualizing Alarm #1

During *training* count how often each path taken (per user)
Testset Results – Alarm #1

“Intrusion” Detection Rates
(with < 1 false alarm per day per user)

Detection Rate on Testset

Window Width (W)

Absolute Prob
Using Relative Probabilities

Alarm #2: \[ \text{Prob}(\text{keystrokes} \mid \text{machine owner}) \]

\[ \text{Prob}(\text{keystrokes} \mid \text{population}) \]

Detection Rate on Testset

Window Width (W)

- Relative Prob
- Absolute Prob
Using *Two* Best Alarm Types (Chosen on Tuning Set)

We are also investigating other keystroke-related alarms (eg, length of words, sentences, etc)
Cascading Window Sizes

- Alarm in *Window Size* = $W$
  also if alarm in any *smaller* window

- (To Do: Re-choose thresholds for this scenario)
Cascading Window Sizes - Results

Can detect intrusions before window $W$ completely full

![Graph showing detection rate on test set vs. window width (W)]

Detection Rate on Testset

Window Width (W)

- Cascaded Alarm #2
- Uncascaded Alarm #2
- Cascaded False Alarms
- Uncascaded False Alarms
- One False Alarm per Day

Can detect intrusions before window $W$ completely full.
Tradeoff between False Alarms and Detected Intrusions (ROC Curve)

Note: left-most values result from ZERO tune-set false alarms
Current Work

- Extend to non-keystroke data
- Condition probabilities on other measurements
  - \( \text{Prob( keystrokes } \mid \text{ MS Office running }) \),
  - \( \text{Prob( keystrokes } \mid \text{ browser running }) \), ...
- Combine additional alarms
  - Approx full joint probability distribution (Bayes nets) on user’s measurements most divergent from general population
- Train standard machine learners to distinguish user \( X \) from general population
Some Related Work

- **Machine learning for intrusion detection**
  - Gosh et al. (1999)
  - Lane & Brodley (1998)
  - Lee et al. (1999)
  - Warrender et al. (1999)
  - Typically Unix-based; system calls &TCP analyzed

- **Analysis of keystroke dynamics**
  - Monrose & Rubin (1997)
  - For authenticating passwords
Conclusion

- Can accurately characterize *individual* user behavior using simple models
- Separate data into *train*, *tune*, and *test* sets
  - “Let the data decide” good parameter settings, on per-user basis
- Normalize prob’s by general-population prob’s
  - Separate *rare for this user/server* from *rare for everyone*