A Host-based Anomaly Detection Approach by Representing System Calls as States of Kernel Modules

Shariyar Murtaza and Wahab Hamou-Lhadj

Software Behaviour Analysis (SBA) Research Lab
Concordia University, Canada

Dec 10, 2013 at AHLS Meeting

Intrusion Detection Systems (IDS)

- Monitor computer or network activity for signs of intrusions and alert administrator
- Provide information for forensics analysis
- Administrator confirm or refute IDS alerts

IDS Taxonomy: Detection Behaviour

- Signature based Detection
  - Look for events that match patterns of known attacks
  - Can only detect attacks for which a signature exists
- Anomaly Detection
  - Look for significant deviations from normal system behavior
  - Theoretically, it should be able to detect any attack

IDS Taxonomy: Protection Behaviour

- Network-based (NIDS) monitor network traffic for multiple hosts
- Host-based (HIDS) monitor activities of host systems
  - e.g., system calls, application logs and file systems

Host-based Anomaly Intrusion Detection Techniques

- Researchers applied different algorithms on logs or system calls to detect anomalies, such as:
  - Sliding window technique
  - HMM
  - Neural networks (two-class)
  - K-means
  - Varied length n-gram technique
  - Context Free Grammar

Limitations

- High false positive or false alarm rate
  - Any unknown sequence is considered anomaly by the sliding window technique

Sequence 1	fork	read	read	fork	read	read	fork	read	read	Normal
Sequence 2	fork	read	read	fork	read	fork	read	read	fork	Unknown

- High processing time
  - Time to train HMM

Kernel State Modeling (KSM)

- KSM is an anomaly detection technique
  - Transforms system calls into kernel modules, called states
  - Detect anomalies at the level of interaction of states

Transforming System Calls into States of Kernel Modules

State	Module in Linux Source Code	# of System Calls
AC	Architecture	10
FS	File System	131
IPC	Inter Process Communication	7
KL	Kernel	127
MM	Memory Management	21
NT	Networking	2
SC	Security	3
UN	Unknown	37

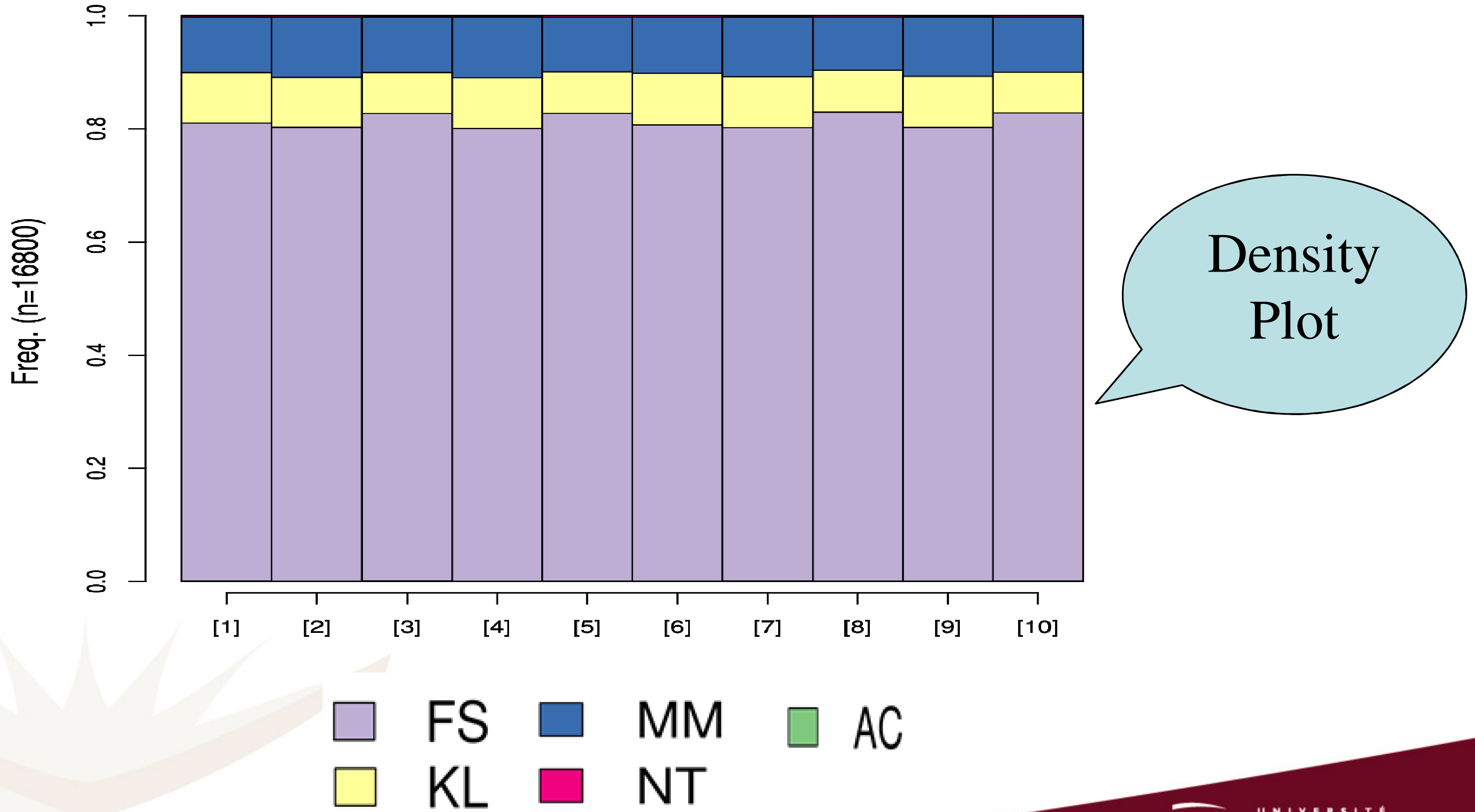
[Ref]: http://syscalls.kernelgork.com

Transforming System Calls into States of Kernel Modules

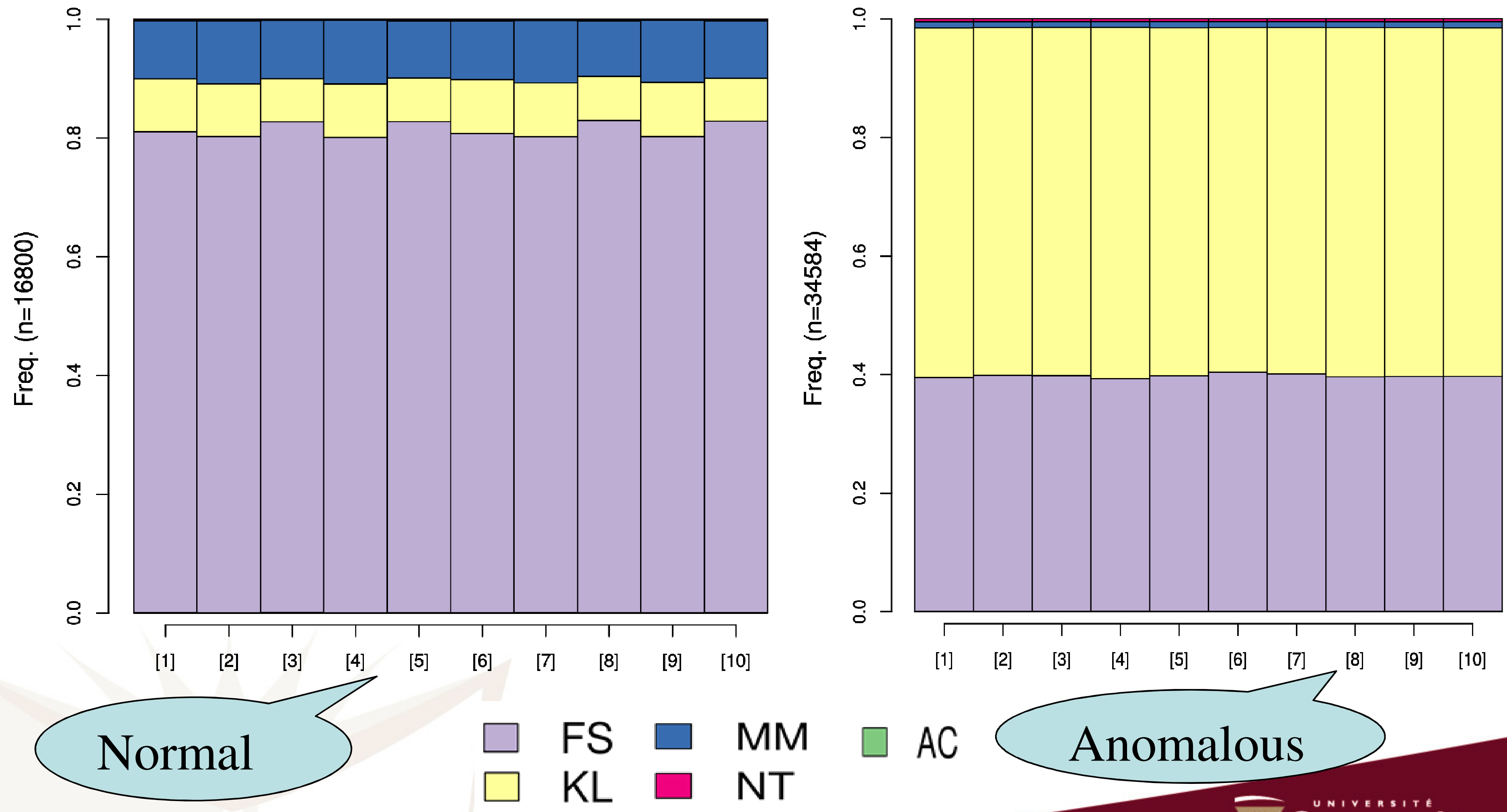
Sequence	fork	read	read	fork	read	read	fork	read	read
Sequence	fork	read	write	write	write	write	write	write	read
Sequence	read	read	read	fork	write	close	open	open	open
Sequence	read	close	write	write	close	write	close	read	read

Sequence	Fork	read	read	fork	read	read	fork	read	read

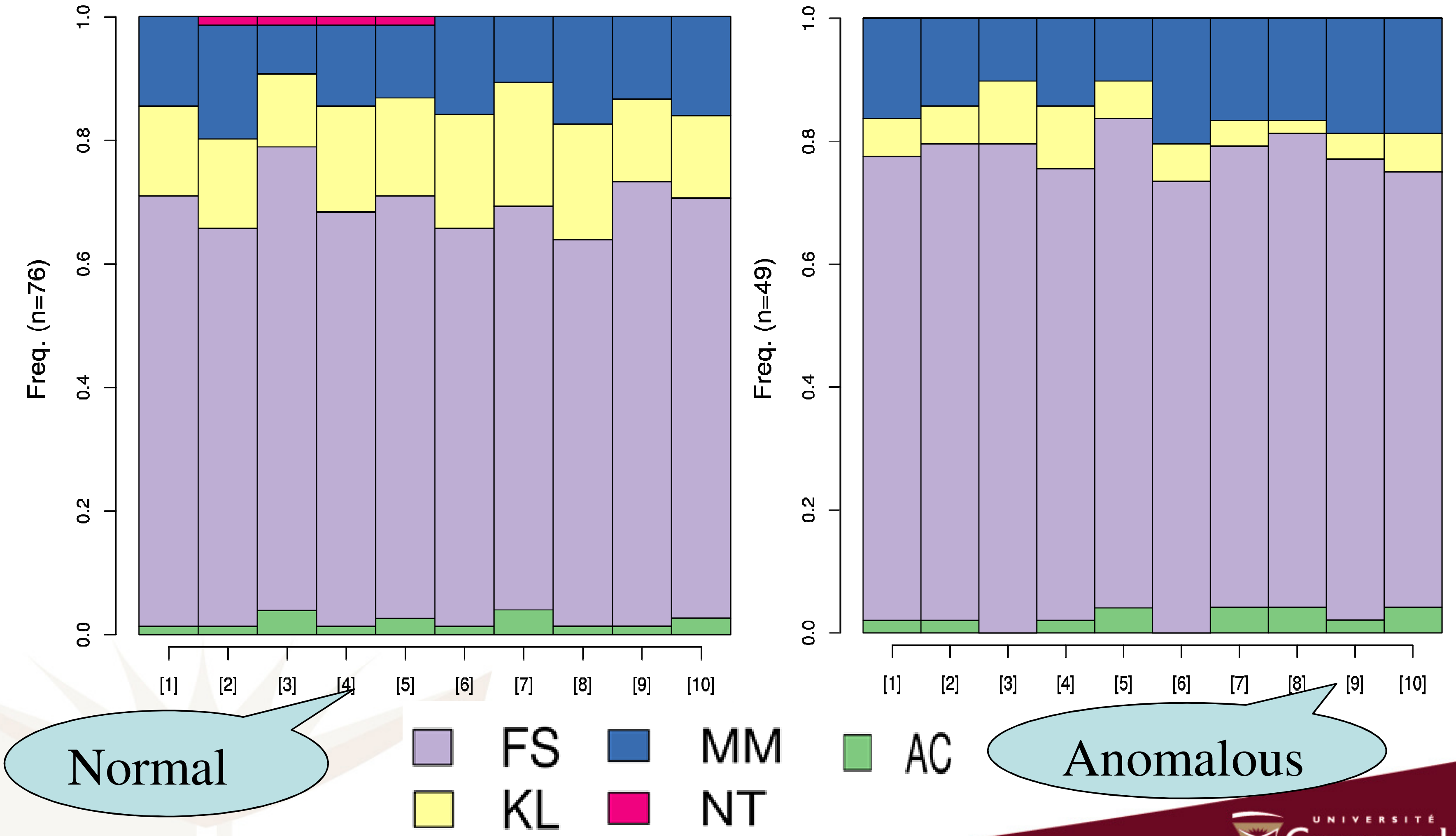
Transforming System Calls into States of Kernel Modules



Anomaly Detection in Firefox



Anomaly Detection in Login Utility



UNIVERSITÉ Concordia UNIVERSITY

Automatically Detecting Anomalies

Trace #	FS	KL	MM	Type
1	0.60	0.20	0.00	Normal
2	0.54	0.06	0.40	Normal
3	0.73	0.04	0.23	Normal
4	0.74	0.05	0.03	Normal
5	0.82	0.01	0.03	Normal
6	0.82	0.03	0.11	Normal
7	0.55	0.15	0.19	Anomalous
8	0.53	0.16	0.20	Anomalous

Xlock Program

Automatically Detecting Anomalies

- To determine significant deviation threshold (alpha):
  - Divide normal dataset into training set, validation set, and testing set
  - Extract probabilities from training set
  - Evaluate on validation set and adjust alpha till no false alarms
  - Measure accuracy on testing set

Case Study 1: Dataset

Program	# Normal Traces			#Attack Types	#Attack Traces
	Training	Validation	Testing		
Login	4	3	5	1	4
PS	10	4	10	1	15
Stide	400	200	13126	1	105
Xlock	91	30	1610	1	2
Firefox	125	75	500	5	19

Case Study 1: Results

Program	Technique	TP rate	FP rate
Login	KSM (alpha=0.00)	100%	0.00%
	Stide (win=6)	100%	40.00%
	Stide (win=10)	100%	40.00%
	HMM (states=10)	100%	40.00%
PS	KSM (alpha=0.02)	100%	10.00%
	Stide (win=6)	100%	10.00%
	Stide (win=10)	100%	10.00%
	HMM (states=5)	100%	30.00%
Xlock	KSM (alpha=0.04)	100%	0.00%
	Stide (win=6)	100%	1.50%
	Stide (win=10)	100%	1.50%
	HMM (states=5)	100%	0.00%

Case Study 1: Results

Program	Technique	TP rate	FP rate
Stide	KSM (alpha=0.06)	100%	0.25%
	Stide (win=6)	100%	4.97%
	Stide (win=10)	100%	5.25%
	HMM (states=5)	100%	0.25%
Firefox	KSM (alpha=0.08)	100%	0.60%
	Stide (win=6)	100%	44.60%
	Stide (win=10)	100%	49.20%
	HMM (states=5)	100%	1.40%

Case Study 1: Execution Time

	Size of All Traces	KSM	Stide	HMM
Login	26.2KB	4.46 secs	0.03 secs	56.43 mins
PS	29.6KB	5.14 secs	0.11 secs	46.24 mins
Xlock	47.4MB	1.51 mins	12.3 mins	13.37 hrs
Stide	36.2MB	5.85 mins	8.53 mins	2.3 days
Firefox	270.6MB	9.35 mins	4.17 hrs	4.03 days

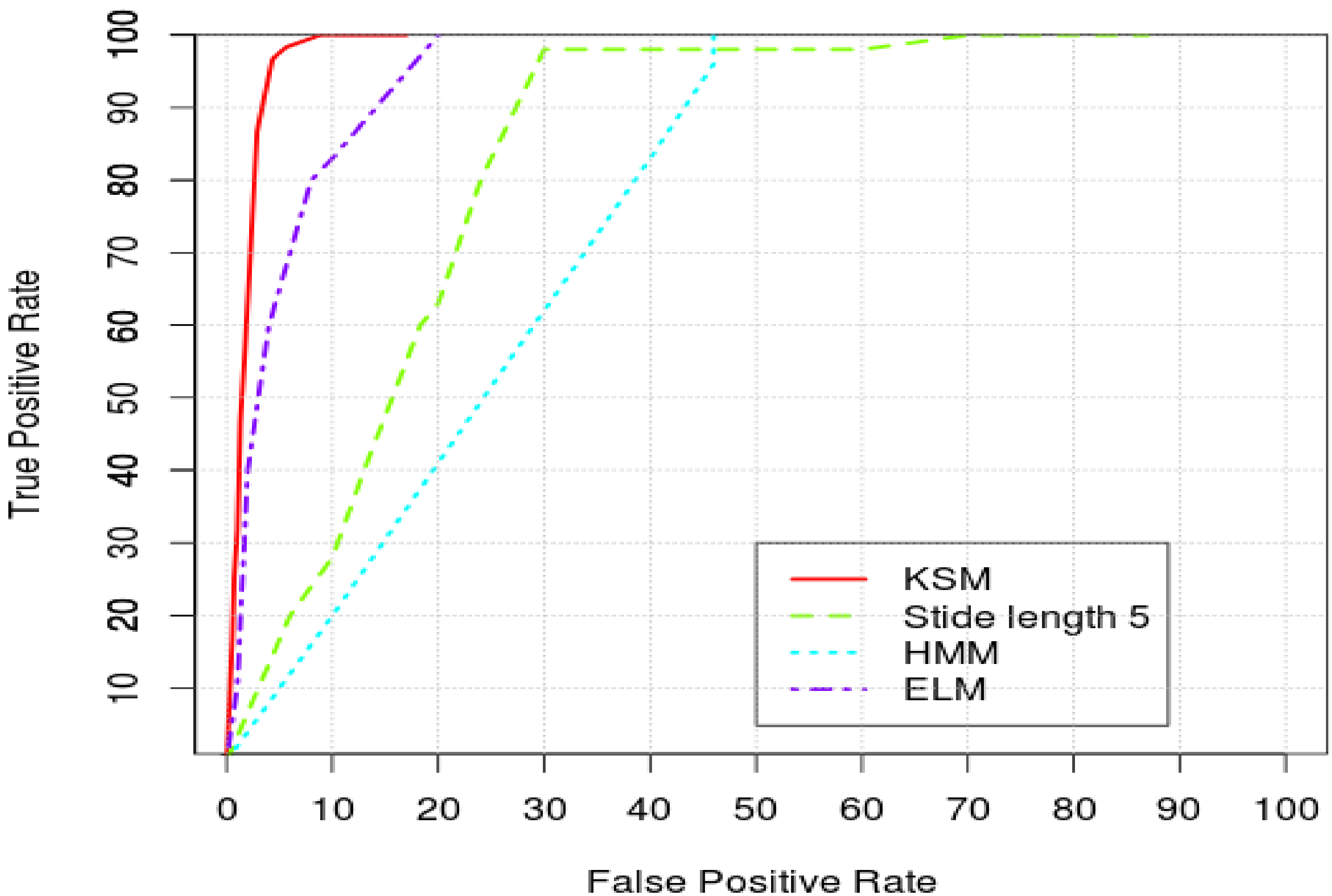
Case Study 2: ADFA Linux Dataset

- A host with Ubuntu 11.04, Apache 2.2.17, PHP 5.3.5, TikiWiki 8.1, FTP server, MySQL 14.14 and an SSH server
  - web-based exploitation
  - simulated social engineering
  - poisoned executable,
  - remotely triggered vulnerabilities,
  - remote password brute force attacks
  - system manipulation
- No per process separation of traces

Case Study 2: ADFA Linux Dataset

Normal	
# of Training traces	833
# of testing traces	4373
Total attacks	
# of attacks	60
# of attacks traces	686

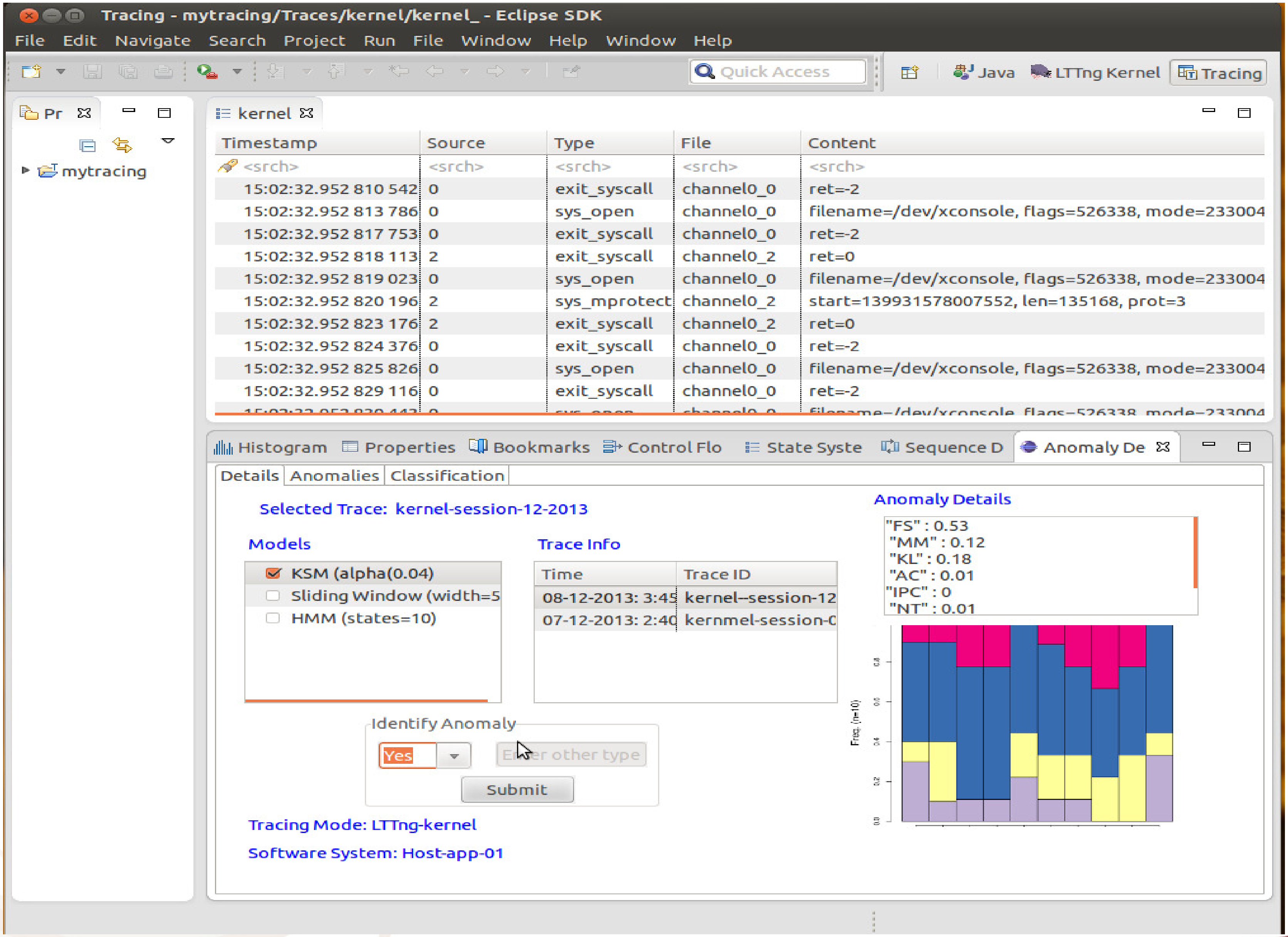
Case Study 2: ADFA Linux Dataset



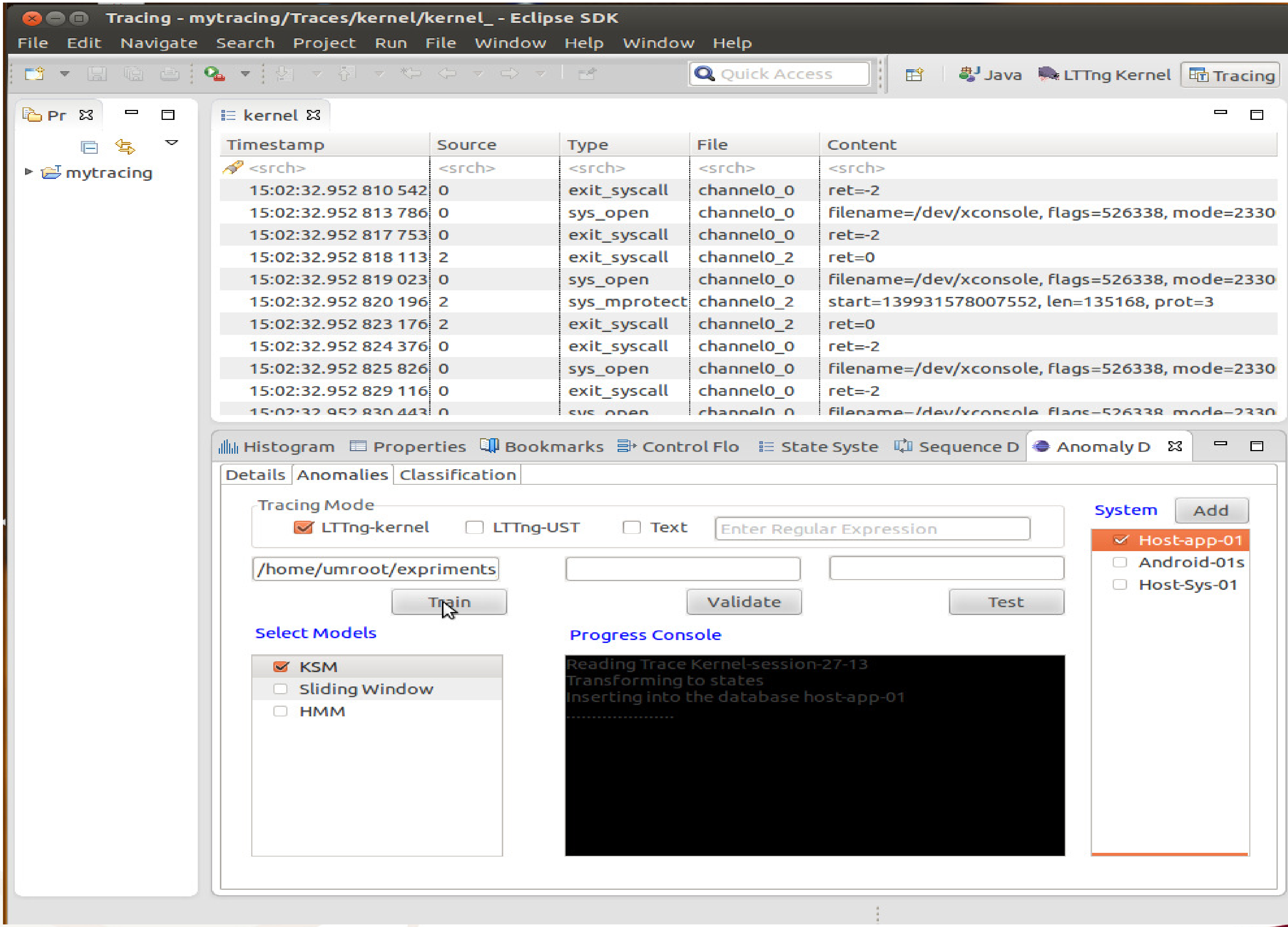
Conclusions

- KSM is efficient in processing time, has low FP rate and provides visual feedback
- Visual feedback allows an analyst to make judgment about false positives and true positives
- These attributes are lacking simultaneously in HMM and Stide

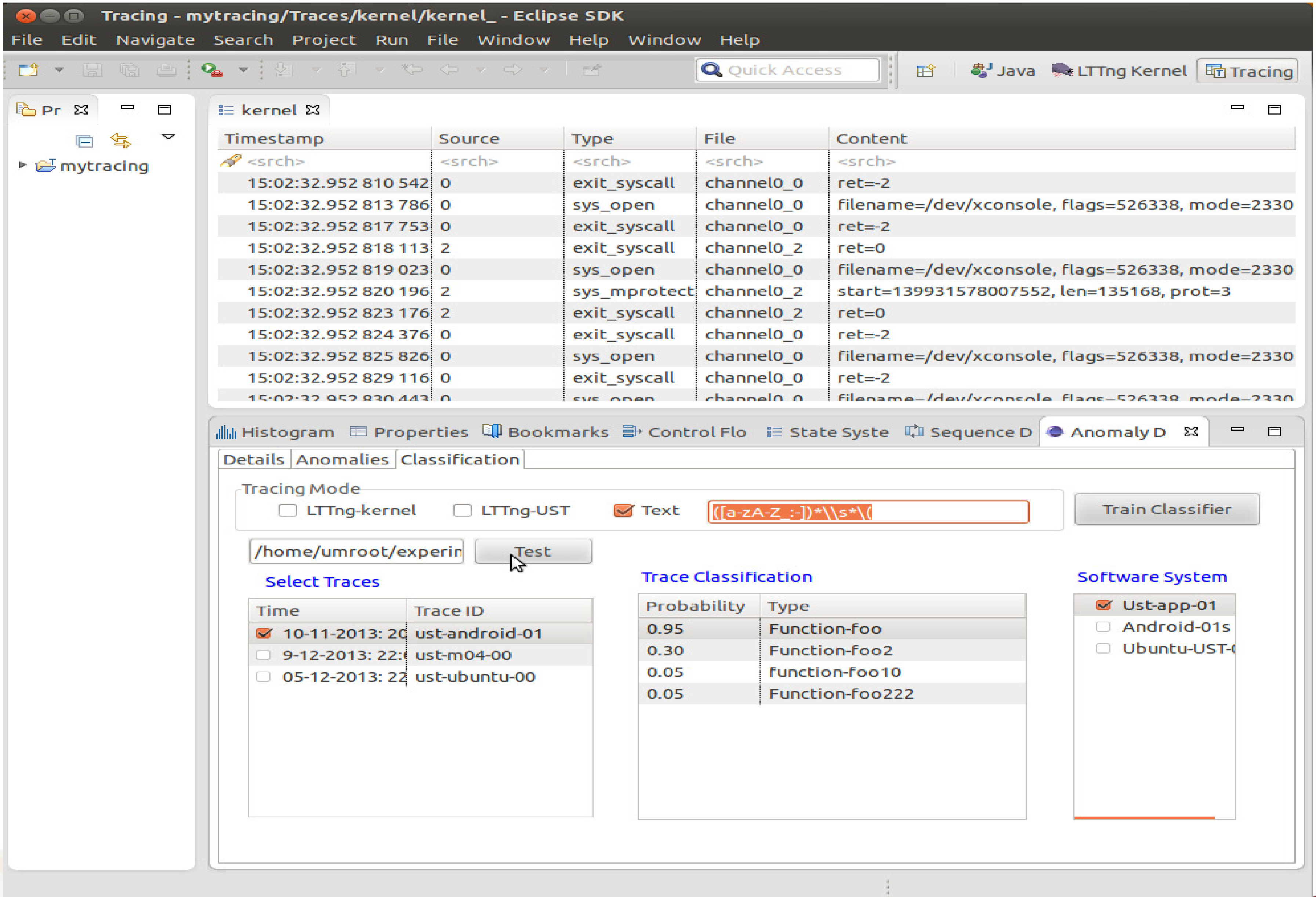
Screenshots of TMF Plugin: Detective of Anomalies in Software Systems (DASS)



Université Concordia University



Tracing - mytracing/Traces/kernel/kernel_ - Eclipse SDK
File Edit Navigate Search Project Run File Window Help Window Help
Quick Access
Java
LTTng Kernel
Tracing
Pr
mytracing
kernel
Timestamp
Source
Type
File
Content
<srch>
15:02:32.952 810 542 0 exit_syscall channel0_0 ret=-2
15:02:32.952 813 786 0 sys_open channel0_0 filename=/dev/xconsole, flags=526338, mode=2330
15:02:32.952 817 753 0 exit_syscall channel0_0 ret=-2
15:02:32.952 818 113 2 exit_syscall channel0_2 ret=0
15:02:32.952 819 023 0 sys_open channel0_0 filename=/dev/xconsole, flags=526338, mode=2330
15:02:32.952 820 196 2 sys_mprotect channel0_2 start=139931578007552, len=135168, prot=3
15:02:32.952 823 176 2 exit_syscall channel0_2 ret=0
15:02:32.952 824 376 0 exit_syscall channel0_0 ret=-2
15:02:32.952 825 826 0 sys_open channel0_0 filename=/dev/xconsole, flags=526338, mode=2330
15:02:32.952 829 116 0 exit_syscall channel0_0 ret=-2
Histogram
Properties
Bookmarks
Control Flo
State Syste
Sequence D
Anomaly D
Details
Anomalies
Classification
Tracing Mode
LTTng-kernel
LTTng-UST
Text
Enter Regular Expression
System
Add
Host-app-01
Android-01s
Host-Sys-01
/home/umroot/expriments
Train
Validate
Test
Select Models
KSM
Sliding Window
HMM
Progress Console
Reading Trace Kernel-session-27-13
Transforming to states
Inserting into the database host-app-01



Université Concordia University

Thank you!

Online surveillance of critical computer systems through advanced host-based detection

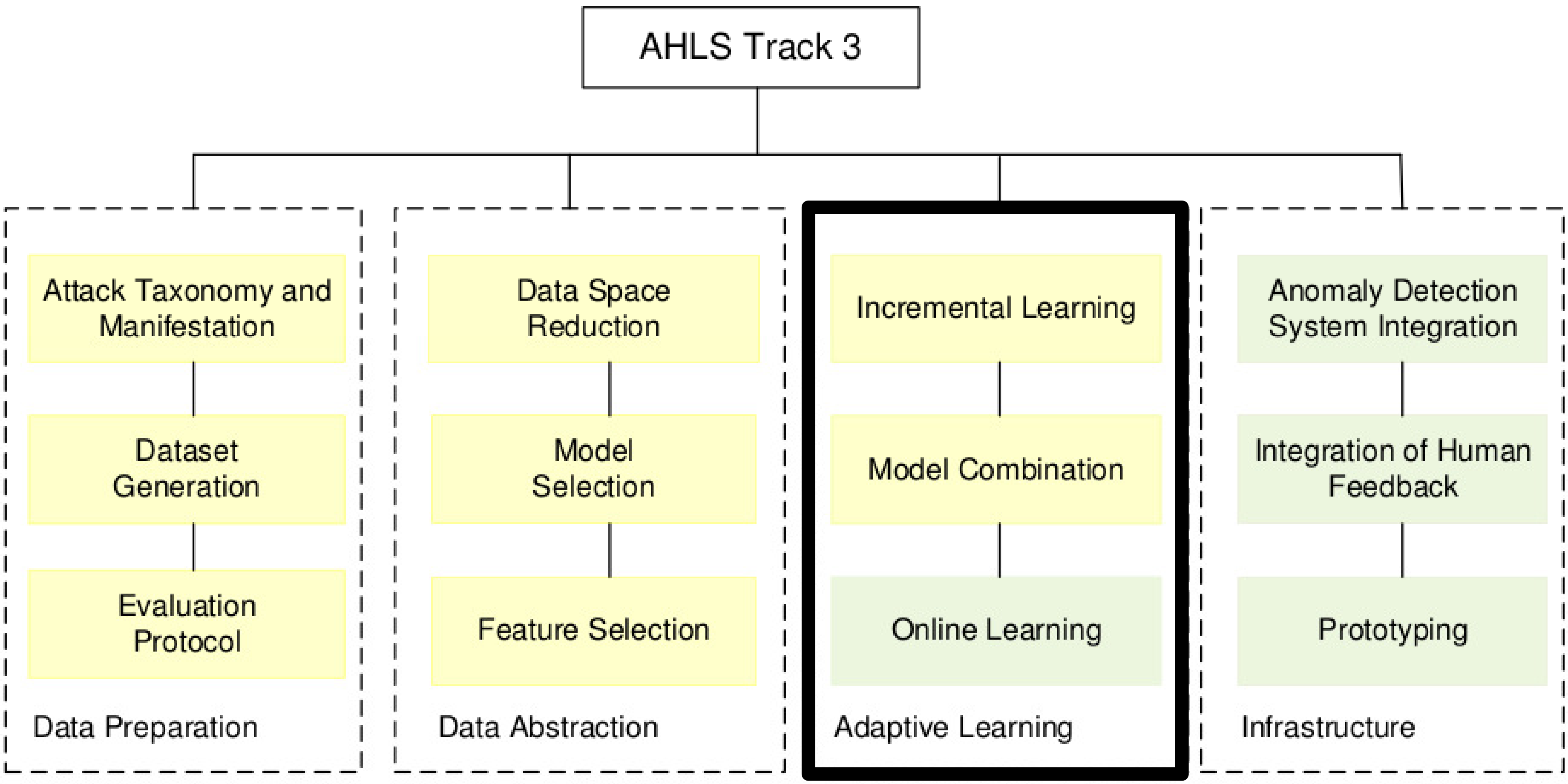
Harmonized Anomaly Detection Techniques – Project Track 3

Wael Khreich and Wahab Hamou-Lhadj

Software Behaviour Analysis (SBA) Research Lab
Concordia University
Montreal, QC, Canada

Dec 10, 2013

Research Threads



Model Combination

- Multiple Classifier Systems, Ensemble of Classifiers, Ensemble Methods, etc.
- A single classifier or model may not provide a good approximation to the underlying data structure or distribution
  - No dominant classifier for all data distributions (“no free lunch” theorem)
  - True data distribution is usually unknown
  - Limited amount of (labeled) data is typically provided training

Model Combination - Advantages

- Can improve overall system accuracy because different models may:
  - Have different domains of expertise
  - Converge to different local optima
  - Provide complementary information
  - Commit different type of errors
- Can improve system adaptability, modularity and scalability

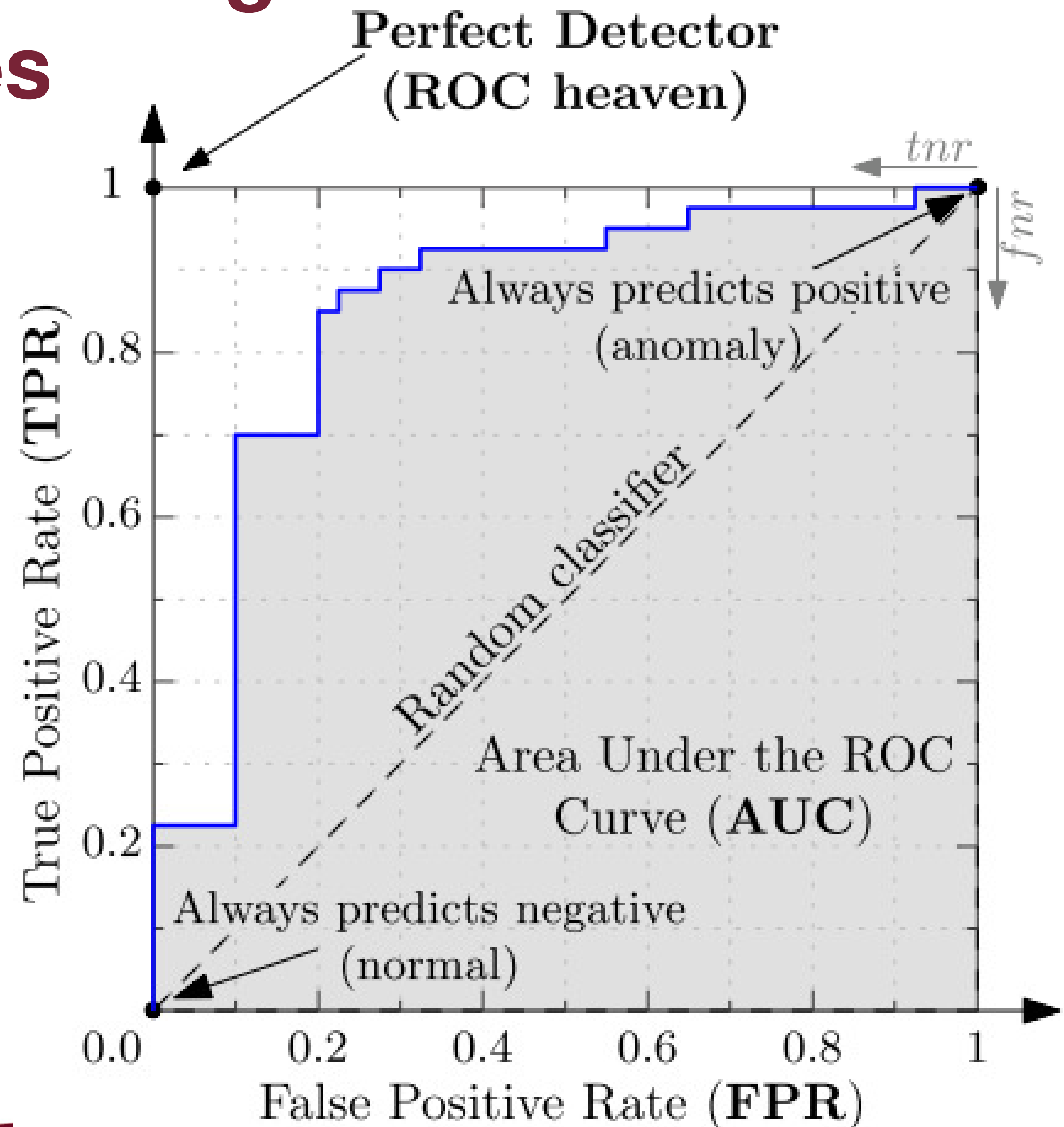
Model Combination - Challenges

- Level of combination?
  - data, feature, score, decision
- Combination method (or function)?
  - static (voting), adaptive (weighted voting), trainable
- Selection of “best” models for combination?
  - complementary, diverse, heterogeneous...
- Choosing the number of models?
  - accuracy vs. complexity, design constraints
- Managing models overtime?
  - changing environment

Receiver Operating Characteristics (ROC) Curves

- True Positive: anomaly detected as anomaly

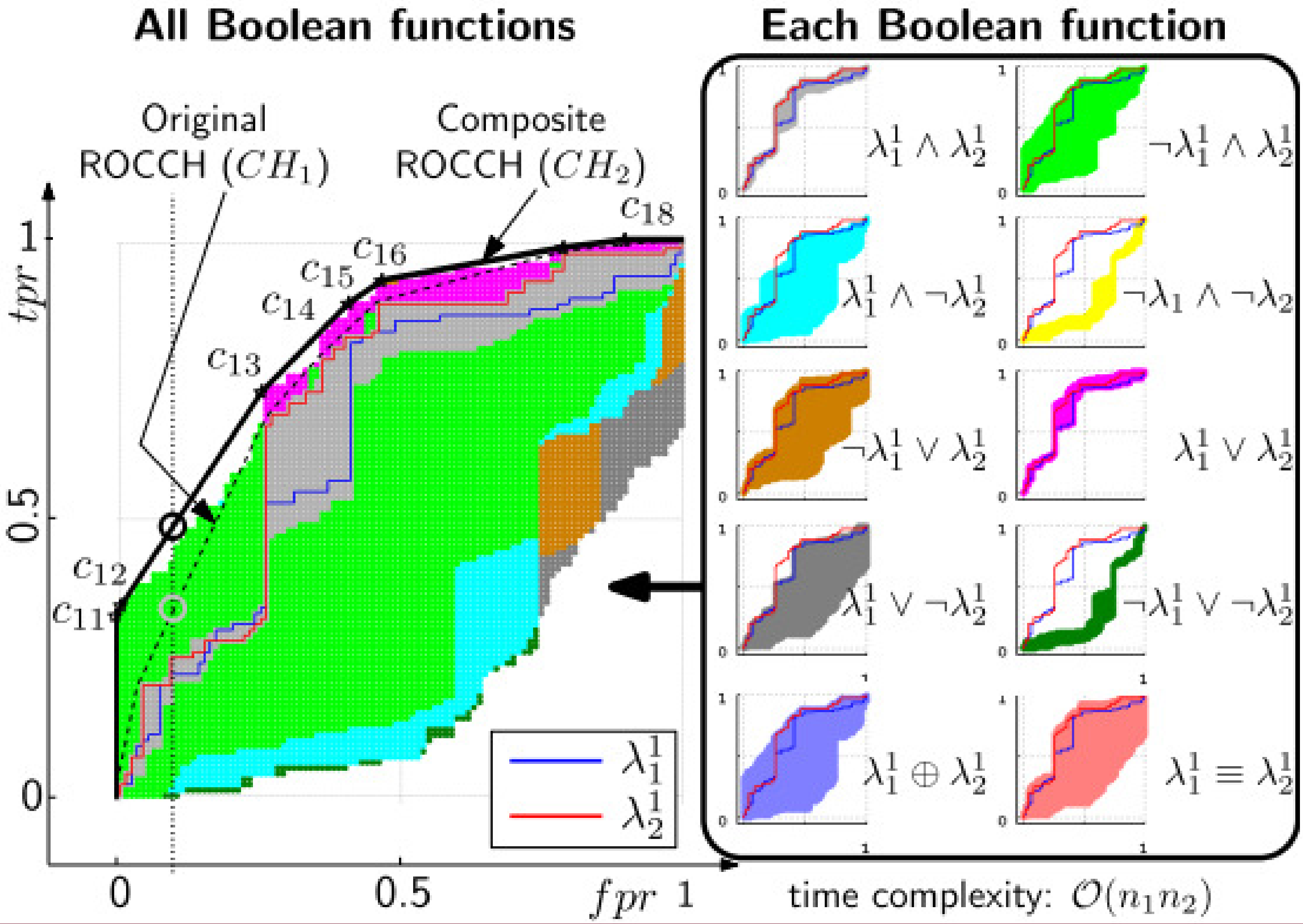
- False Positive: normal detected as anomaly



IBC: Iterative Boolean Combination in the ROC Space

- For each threshold from the first detector and each threshold from the second detector:
  - Combine the responses using all Boolean functions
  - Select thresholds and Boolean functions that improve the ROCCH

IBC - Example



All Boolean functions
Each Boolean function
Original
ROCCH (CH_1)
Composite
ROCCH (CH_2)
c_{11}
c_{12}
c_{13}
c_{14}
c_{15}
c_{16}
c_{18}
tpr
fpr
0
0.5
1
λ_1^1
λ_2^1
$\lambda_1^1 \wedge \lambda_2^1$
$\neg\lambda_1^1 \wedge \lambda_2^1$
$\lambda_1^1 \wedge \neg\lambda_2^1$
$\neg\lambda_1 \wedge \neg\lambda_2$
$\neg\lambda_1^1 \vee \lambda_2^1$
$\lambda_1^1 \vee \lambda_2^1$
$\lambda_1^1 \vee \neg\lambda_2^1$
$\neg\lambda_1^1 \vee \neg\lambda_2^1$
$\lambda_1^1 \oplus \lambda_2^1$
$\lambda_1^1 \equiv \lambda_2^1$
time complexity: $\mathcal{O}(n_1 n_2)$

IBC - Advantages

- Optimize the ROCCH
  - Minimize FPR and Maximize TPR
  - Implicitly the AUC
- Allow to change the operating point during operation (w/o re-training)
- Inherit the properties of the ROC curves
  - Independent of cost of errors
  - Independent of class imbalance
- But require a representative validation set

Case Study: ADFA System Call Datasets (Linux)

Normal	
# of training traces	833
# of testing traces	4373
Attacks	
# of attacks	60
# of attacks traces	686

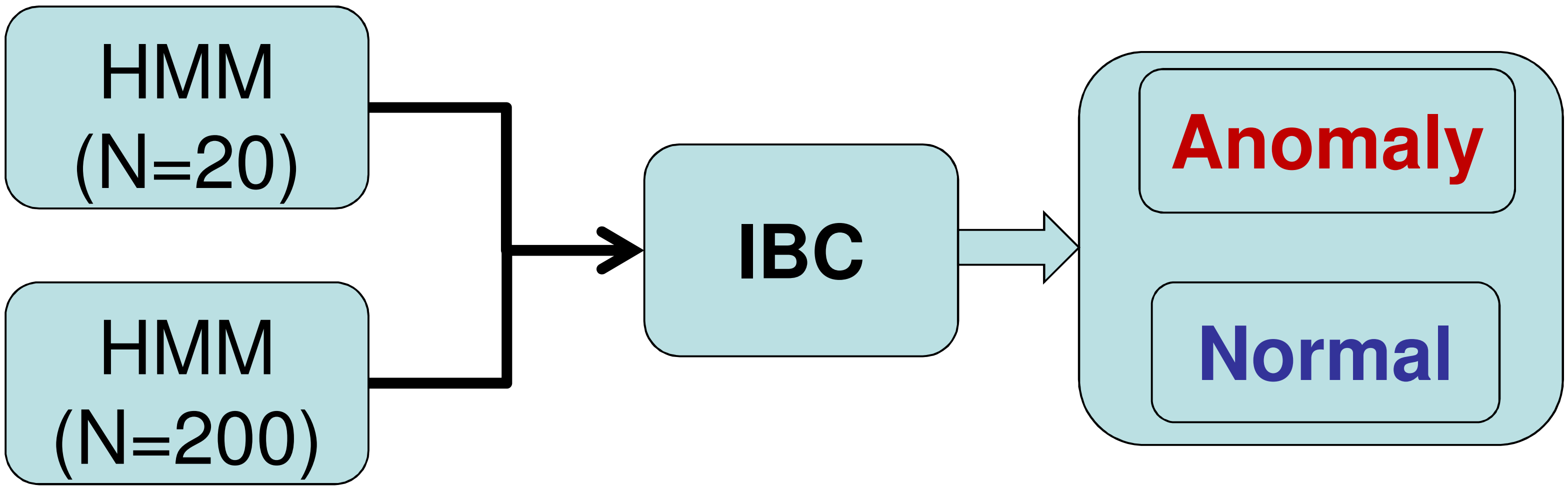
ADFA - Attacks

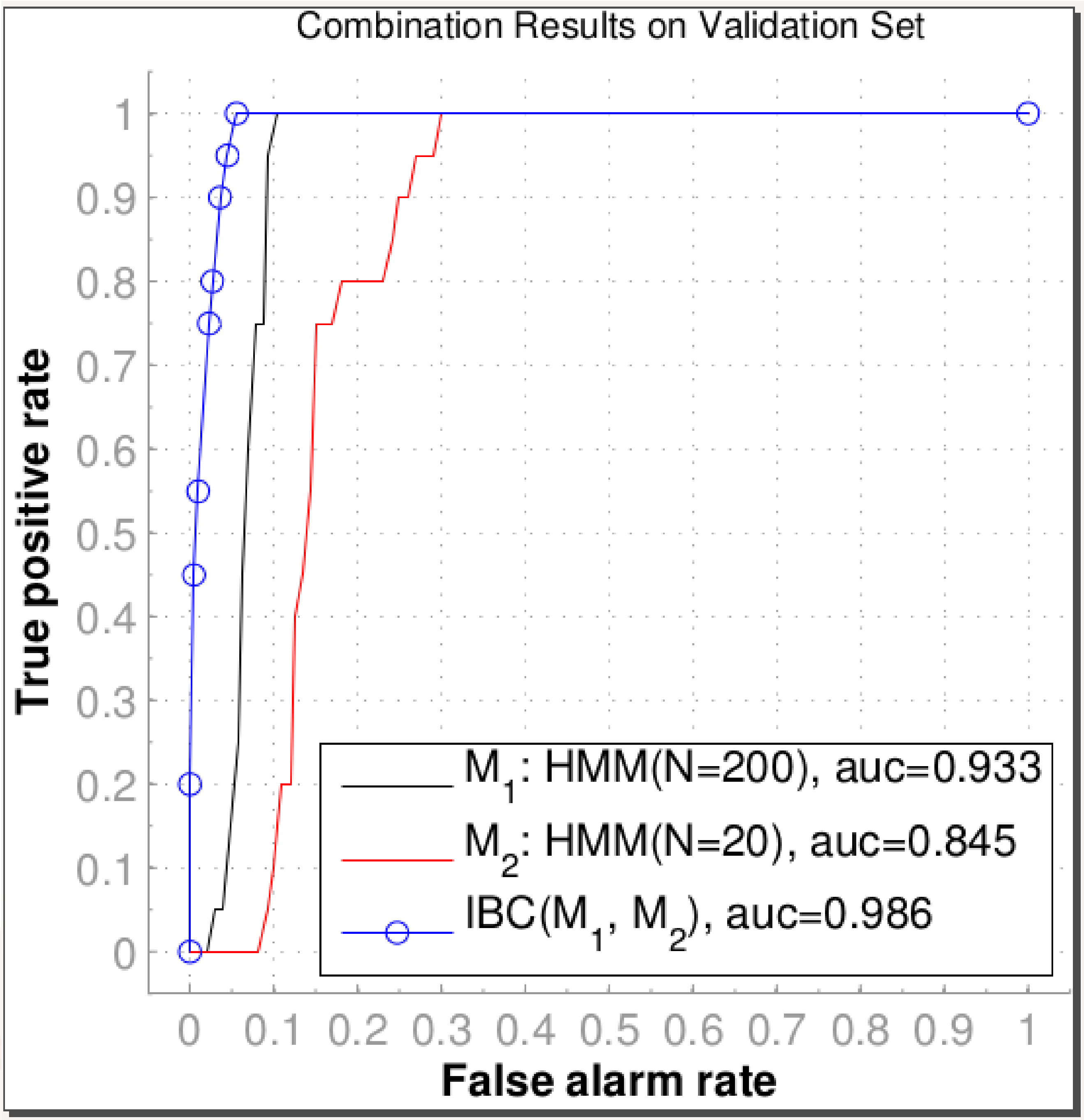
- Ubuntu 11.04, Apache 2.2.17, PHP 5.3.5, TikiWiki 8.1, FTP server, MySQL 14.14 and an SSH server
- Web-based exploitation
- Simulated social engineering
- Poisoned executable
- Remotely triggered vulnerabilities
- Remote password brute force attacks
- System manipulation

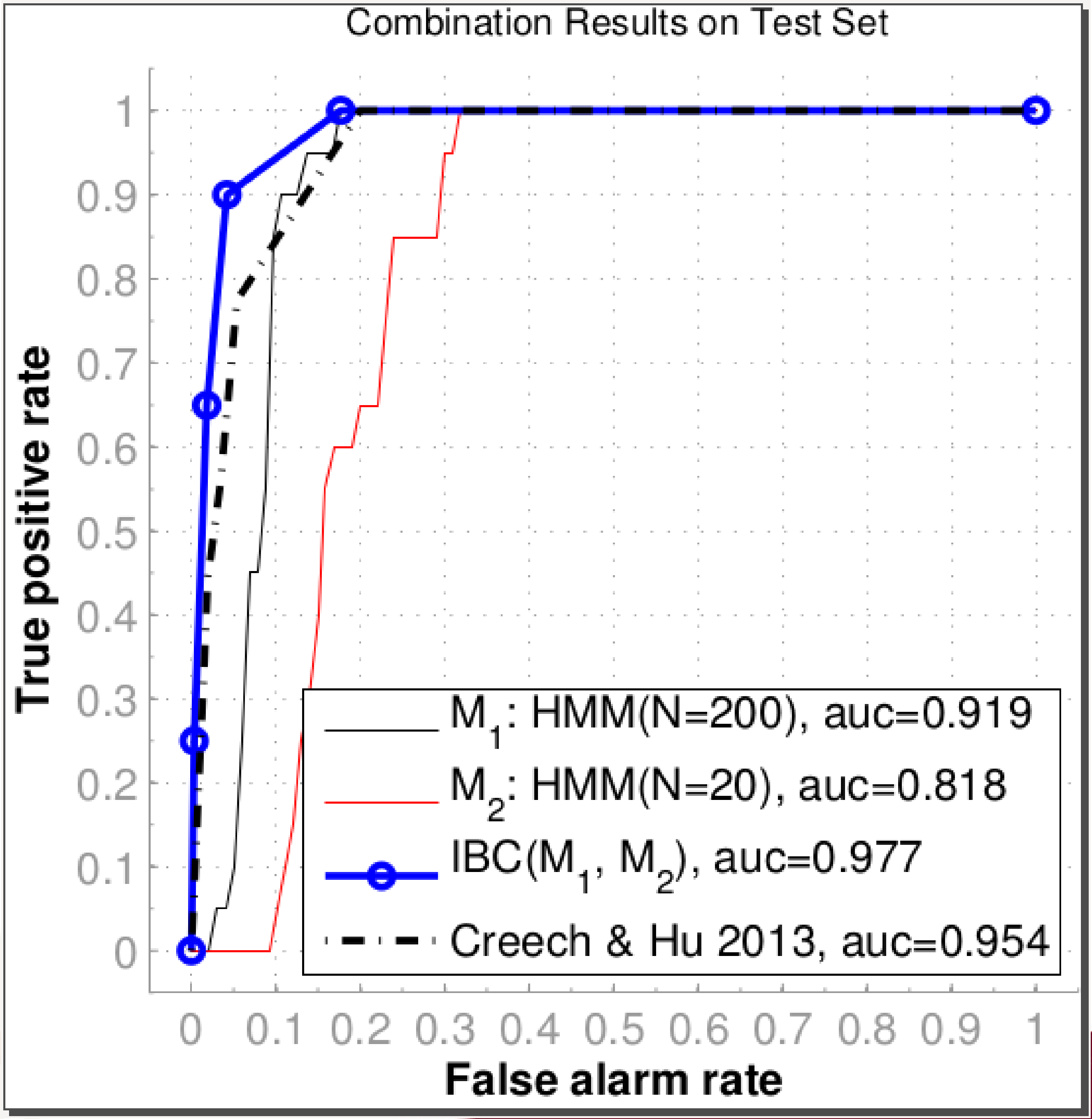
Experimental Methodology

Training Set	
# of training traces	**833**
Validation Set	
# of attacks	**20**
# of normal traces	**1000**
Testing Set	
# of attacks	**40**
# of normal traces	**3373**

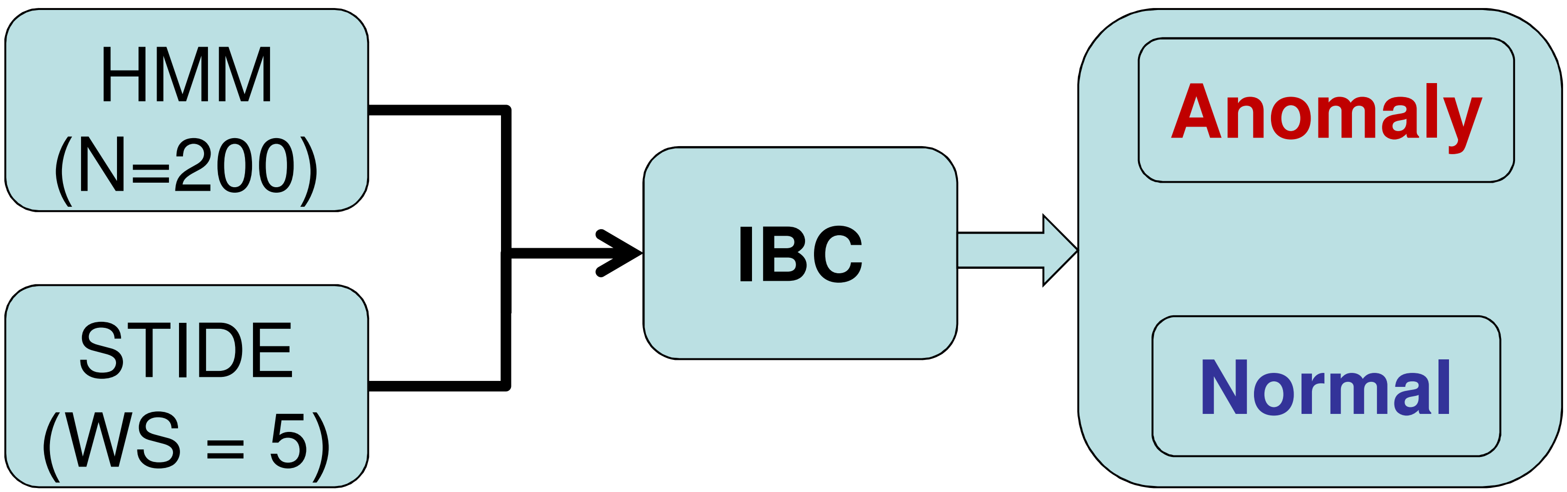
Combination of Responses from Different HMMs

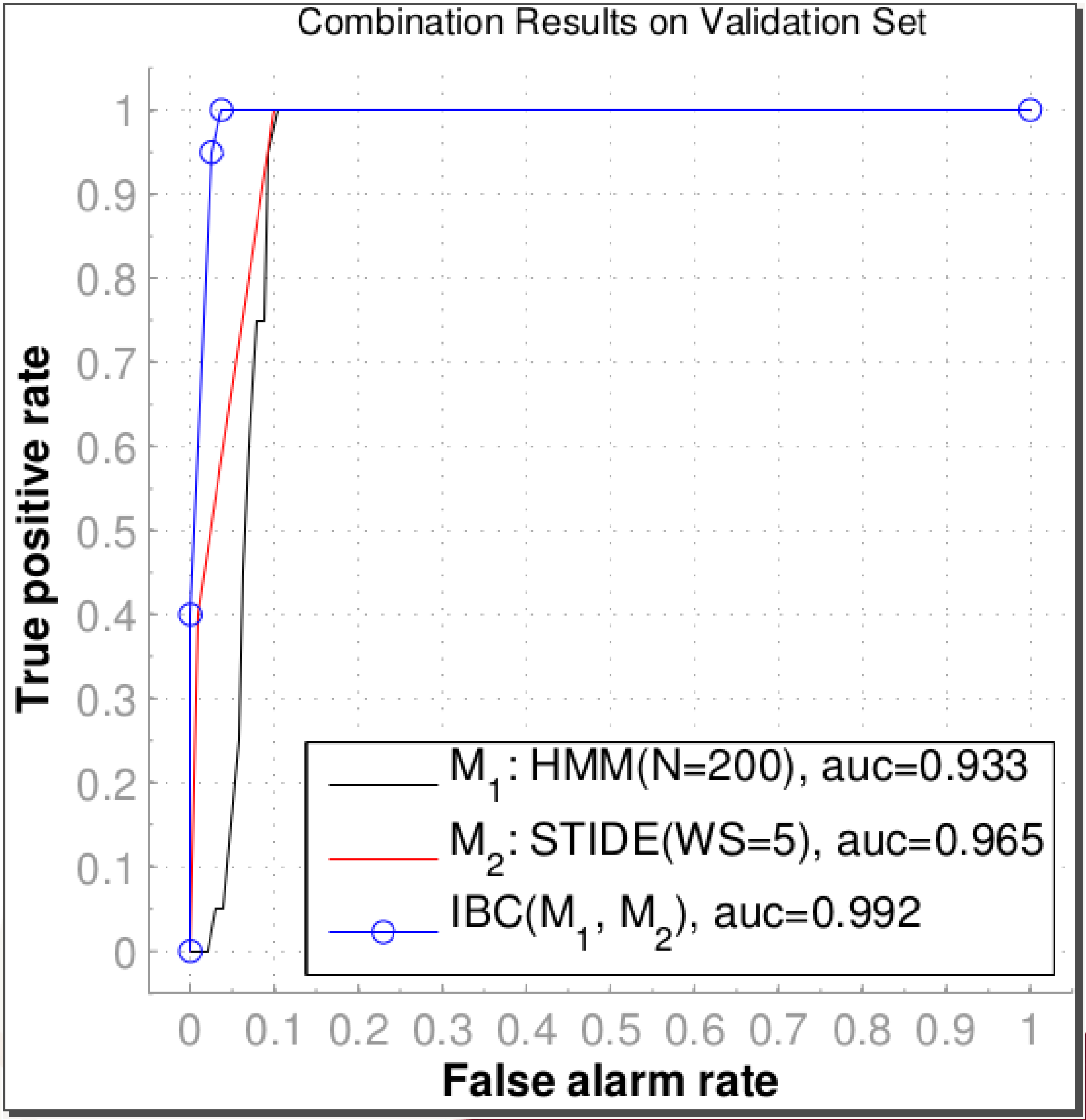



Combination Results on Validation Set
True positive rate
False alarm rate
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
M1: HMM(N=200), auc=0.933
M2: HMM(N=20), auc=0.845
IBC(M1, M2), auc=0.986

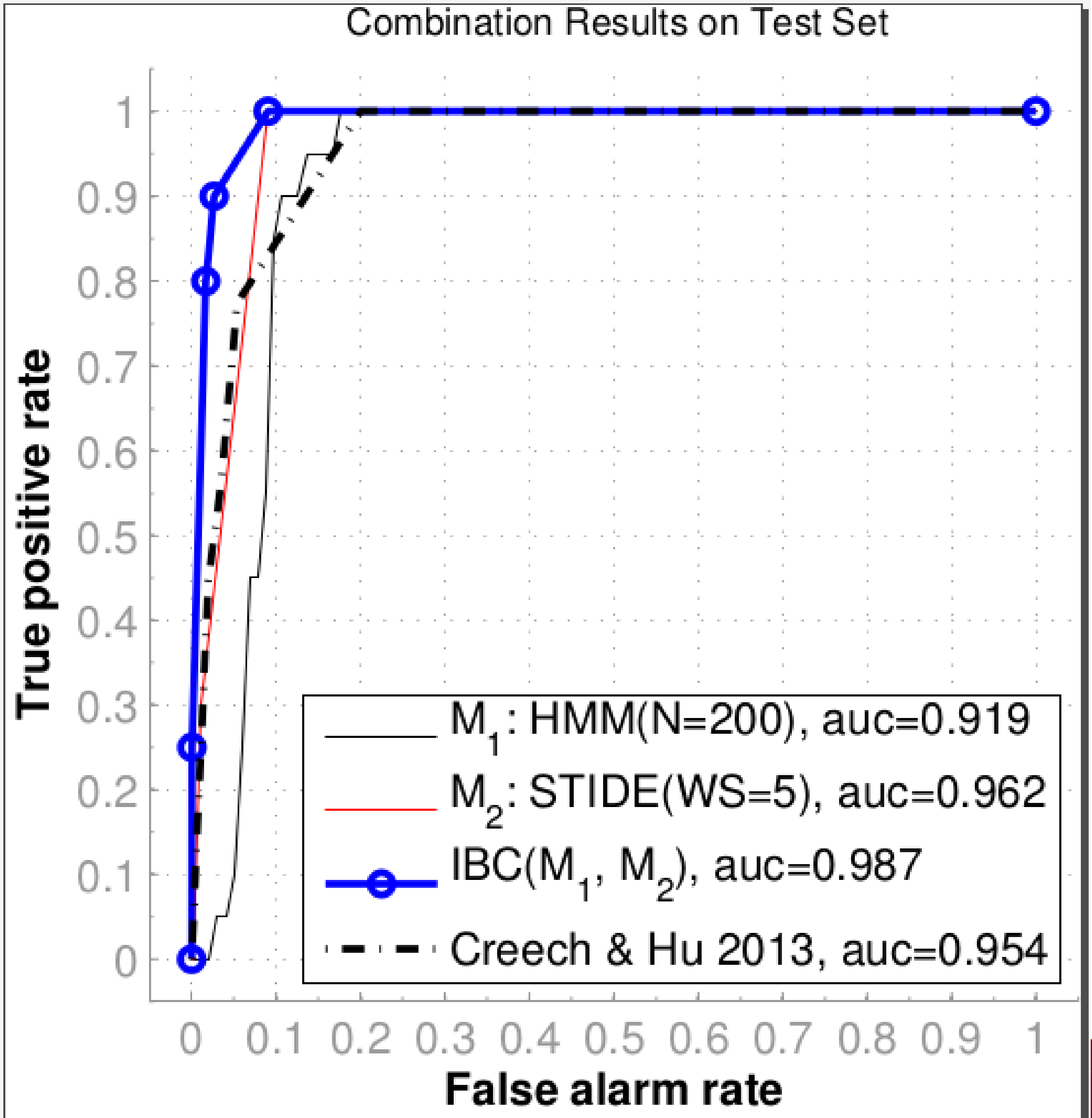

Combination Results on Test Set
True positive rate
False alarm rate
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
M1: HMM(N=200), auc=0.919
M2: HMM(N=20), auc=0.818
IBC(M1, M2), auc=0.977
Creech & Hu 2013, auc=0.954

Combination of HMM and STIDE Responses





UNIVERSITÉ Concordia UNIVERSITY



Combination Results on Test Set
True positive rate
False alarm rate
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
M_1: HMM(N=200), auc=0.919
M_2: STIDE(WS=5), auc=0.962
IBC(M_1, M_2), auc=0.987
Creech & Hu 2013, auc=0.954

Conclusion

- The iterative Boolean Combination (IBC) is shown to significantly improve the detection accuracy while reducing the false alarms
- Combining heterogeneous detectors (HMM & STIDE) seems to improve detection accuracy over homogeneous ensembles (HMMs)
- The detection accuracy of IBC outperformed that of the state-of-art achieved by ELM with semantic features (Creech & Hu; 2013) using their ADFA datasets

Future Work

- Use IBC to select the best combinations among large number of homogenous and heterogeneous models
  - KSM, HMM, STIDE, SVM, Markov Models, etc.
- Apply the IBC of heterogeneous models to incremental learning scenarios
  - Blocks of data come over time, after putting system into operation

Thank You

Wael Khreich
Postdoctoral Fellow
wkhreich@ece.concordia.ca

Software Behaviour Analysis (SBA) Research Lab
Concordia University
Montreal, QC, Canada