Machine Learning based malware detection for Android using static analysis

S. Yerima, G. McWilliams, S. Sezer (CSIT) and I. Muttik (McAfee)

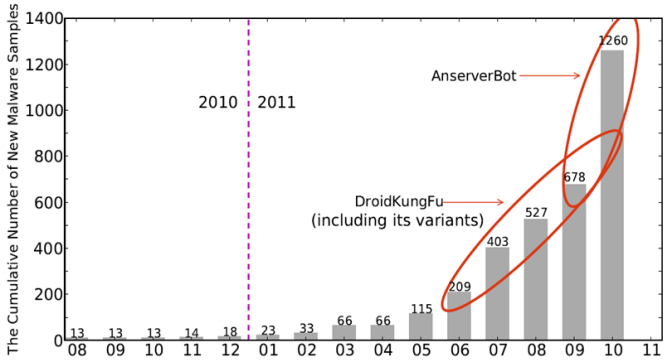
January 2012

Outline

- Background/Motivation
- App reverse engineering/static analysis
- Machine learning model
- Experimental results
- Conclusion

Background/Motivation

- Android mobile platform is becoming more popular with estimated 675,000 apps in the official Google's Android Market and downloads in excess of 25 billion.
- As the popularity of Android grows, so is malware targeting the platform, primarily spread via:
  - Repackaged apps to piggyback payload
  - Update attacks
  - Drive-by downloads
- Growing sophistication to circumvent detection by mobile antivirus software. (A recent study reveals 79.6 % best case to 20.2 % worst case detection)



Cumulative growth of collected Android mobile malware samples. Two major outbreaks AnserverBot (Starting Sept. 2011) and DroidKungFu (starting June 2011) are highlighted. Both are still actively evolving to evade detection from existing anti-virus software.

Background/Motivation

- A cumulative study of the 1260 Android malware samples from 49 of the 52 then discovered families revealed:
  - 86% are legitimate apps repackaged with malicious payload.
  - > 92% turned compromised phones into network or SMS controlled bots.
  - 36.7% leveraged root-exploits
  - 45.3% enable stealthy premium-rate SMS sending or phone calls
  - 51.1% harvest user information including user account and stored messages

Background/Motivation

- Android malware remain undetected for up to three months
- Google's Bouncer introduced in Feb. 2011 uses dynamic analysis and can be circumvented.
- Several third party Android Marketplaces exist, most without app scrutiny
- Mobile Antivirus engine detection rates are low
  - Ineffective against unknown malware
- The above motivates our machine learning based approach using static analysis. Main advantages:
  - Malware cannot modify its behaviour during analysis
  - Proactive and fast
  - Flexible and easily automated
  - Can complement signature-based and/or dynamic methods for unknown malware

CENTRE FOR SECURE INFORMATION TECHNOLOGIES

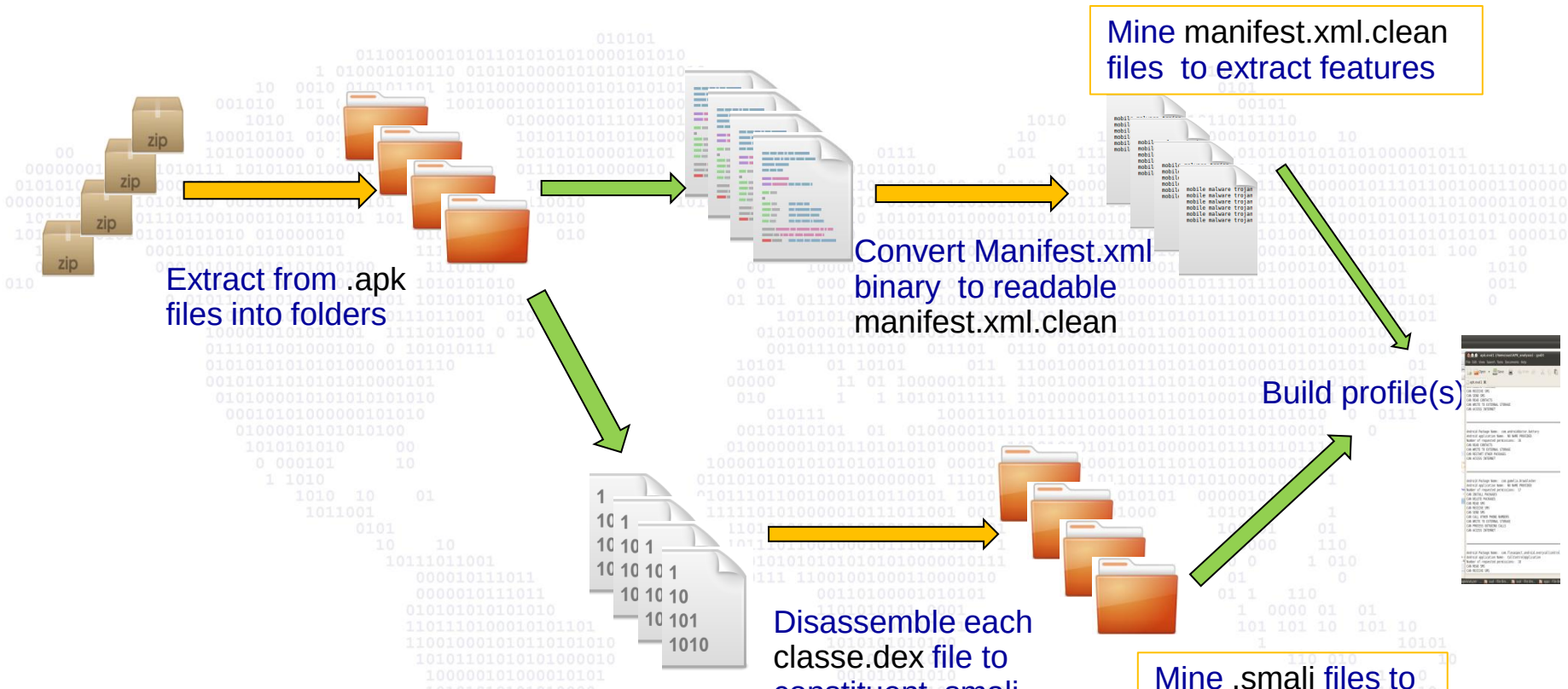
Machine learning model

- Our approach uses Bayesian Classification models obtained from automated static analysis of Android packages.
- Models are built from collection of app and code characteristics that provide indicators of potential malicious activities.
- The models are evaluated with 1000 real malware samples from 49 existing families including DroidKungFu, AnserverBot, Plankton, Pjapps, DroidDream, Geinimi etc.
- Malware samples are from both official and alternative Android markets.

Android app reverse engineering

- Android applications are written in Java and packaged along with data and resource files into a single compressed .apk package.
- The package contains:
  - Manifest (where permissions and components are declared)
  - Dalvik executable file (single .dex file with complete Dalvik bytecode)
  - /assets and /res folders to hold binary, XML-based resources, Libraries etc.
- Java-based apk analyzer reverse engineers these files and applies '*property detectors*' to extract features for training the Machine learning model.

Android package analyzer:
Functional steps



Functional steps of custom built Android package analyzer

App property detectors

- The *property detectors* include:
  - API calls detectors: used to detect use of API's e.g. Telephony Manager APIs for accessing IMSI, IMEI, sending/receiving SMS, listing/installing other packages etc.
  - Command detectors: used to detect references to system commands e.g. 'chmod', 'mount' '/system/bin/su' 'chown, etc.
  - Permission detectors: used to detect permissions requested at runtime as declared in the Manifest file.

Feature ranking and selection

- Thus, *58 different feature attributes* (excluding permissions) are defined as matching criteria for the property detectors.
- These criteria are derived from commonly observed attributed used to detect suspicious activity manually by security analysts.
- A *feature ranking and selection function* ranks these according to relevance using Mutual Information calculation:

$$MI(R_i, C) = \sum_{r \in \{0,1\}} \sum_{c \in \{true, false\}} P(Ri=r, C=c) \frac{P(Ri=r, C=c)}{P(Ri=r) P(C=c)}$$

CENTRE FOR SECURE INFORMATION TECHNOLOGIES

Top 25 Ranked Features

Top 25 MI ranked features based on 1000 malware and 1000 benign apps corpus

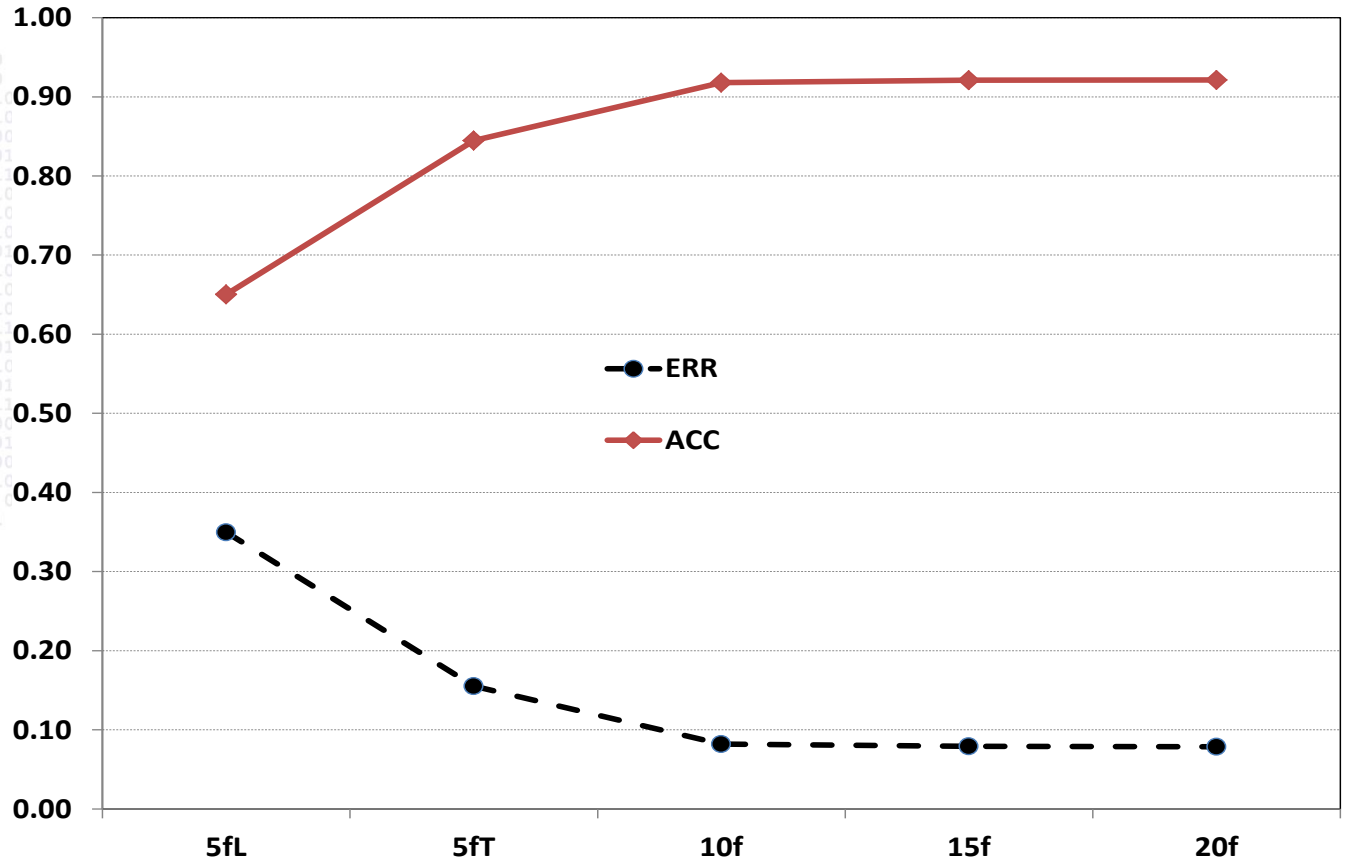
Features	Benign	Malware
getSubscriberId (TelephonyManager)	42	742
getDeviceId (TelephonyManager)	316	854
getSimSerialNumber (TelephonyManager)	35	455
.apk (secondary payload)	89	537
intent.action.BOOT_COMPLETED	69	482
chmod (system command)	19	389
Runtime.exec( ) (Executing process)	62	458
abortBroadcast (intercepting broadcast notifications)	4	328
getLine1Number (TelephonyManager)	111	491
/system/app	4	292
/system/bin	45	368
createSubprocess (creating child process)	0	169
getSimOperator (TelephonyManager)	37	196
remount (system command)	3	122
DexClassLoader (stealthily loading a class)	16	152
pm install (installing additional packages)	0	98
getCallState (TelephonyManager)	10	119
chown (system command)	5	107
.jar (secondary payload)	87	252
mount (system command)	29	152
KeySpec (code encryption)	99	254
/system/bin/sh	4	90
SMSReceiver	3	66
getNetworkOperator (TelephonyManager)	202	353
SecretKey (code encryption)	119	248

CENTRE FOR SECURE INFORMATION TECHNOLOGIES

Experimental scenario

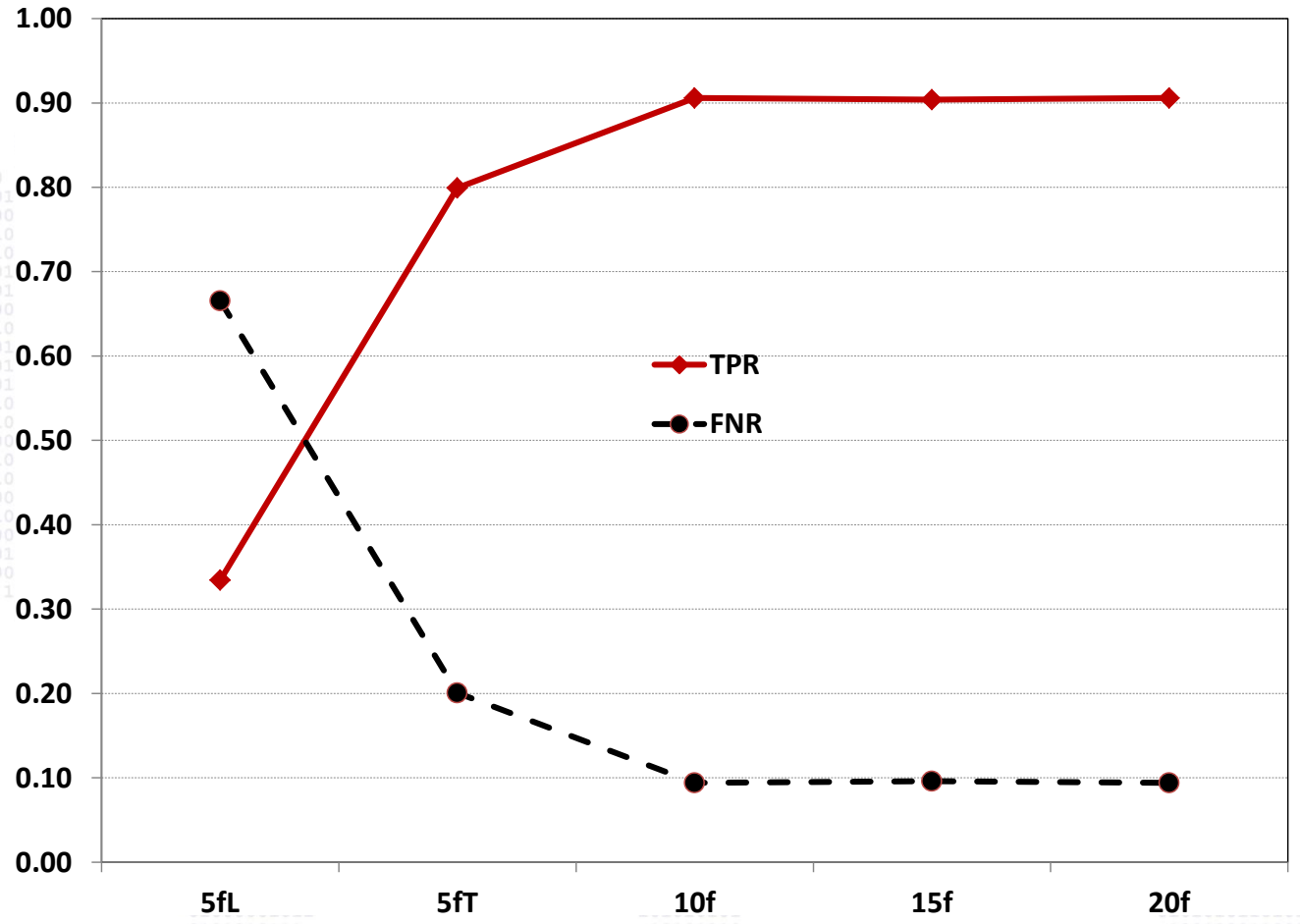
- With feature vectors built from the top 25 MI- ranked features, the Bayesian classifier model is trained to classify apps into ‘benign’ or ‘suspicious’.
- The model is evaluated using N-fold validation technique by varying number of features and number of training samples.
- True positive rate, True negative rate, accuracy, error rate, false positive rate etc. are determined.

Classification results



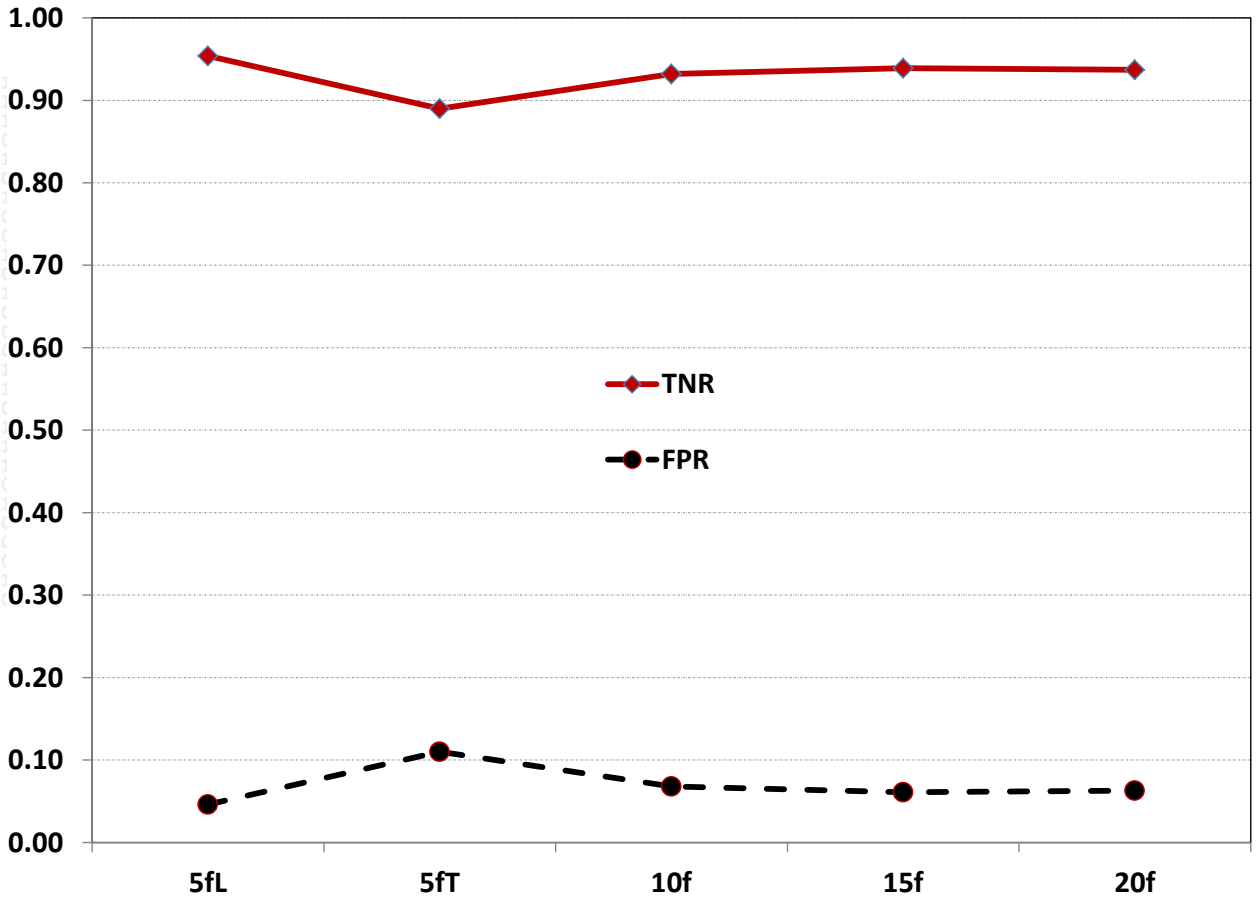
Error and Accuracy based on Bayesian classification

Classification results

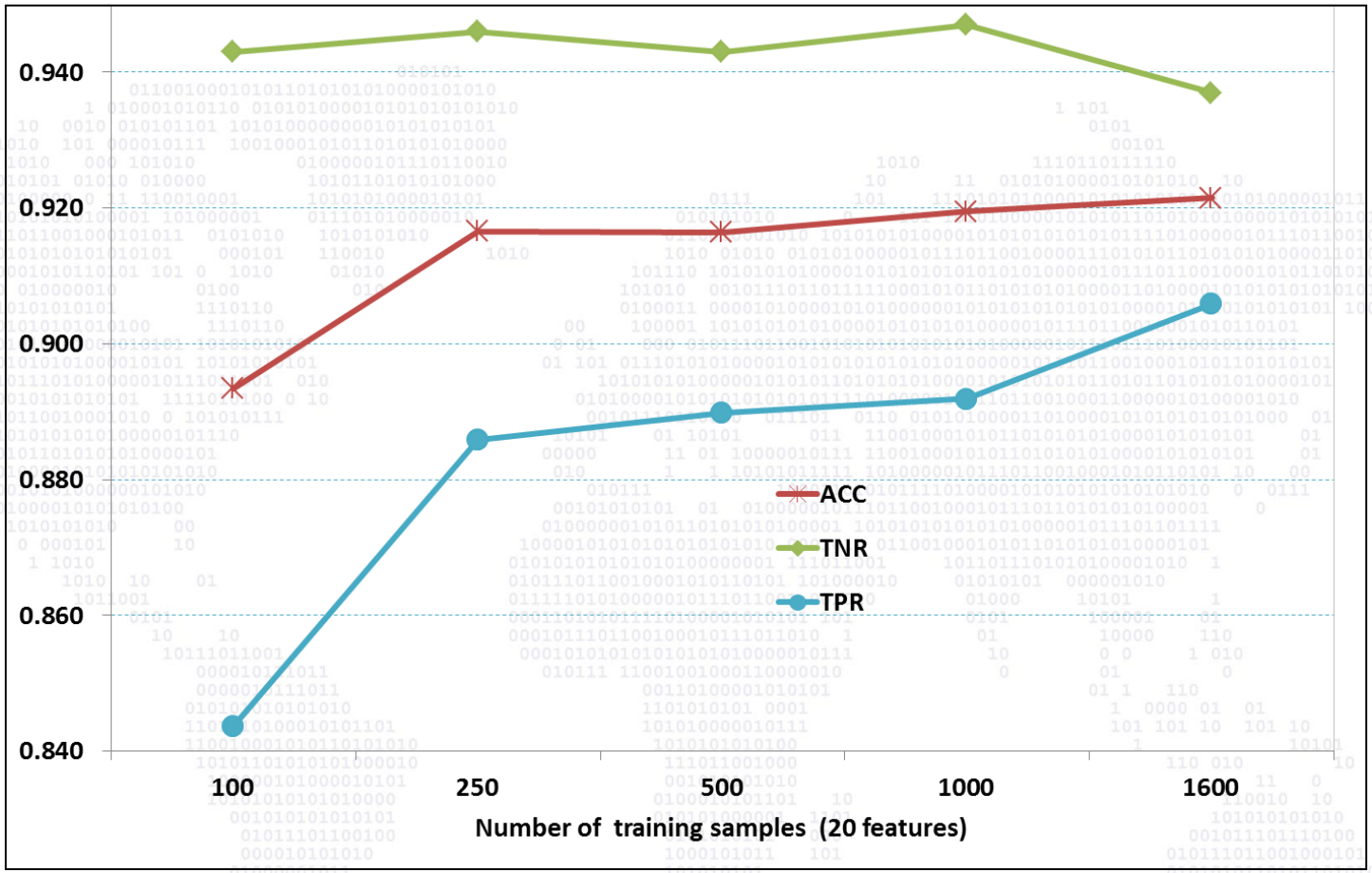


Detection rates (TPR/FNR) based on Bayesian classification

Classification results



TNR/FPR positive rates based on Bayesian classification



ACC/TNR/TPR positive rates with increasing training samples

CENTRE FOR SECURE INFORMATION TECHNOLOGIES

Tabulated results

	ERR	ACC	TNR	FPR	TPR	FNR	Prec.	AUC
5fL	0.350	0.650	0.954	0.046	0.335	0.665	0.860	0.61709
5fT	0.155	0.845	0.890	0.110	0.799	0.201	0.880	0.94437
10f	0.082	0.918	0.932	0.068	0.906	0.094	0.931	0.97428
15f	0.079	0.921	0.939	0.061	0.904	0.096	0.937	0.97232
20f	0.079	0.921	0.937	0.063	0.906	0.094	0.935	0.97223

Samples	ERR	ACC	TNR	FPR	TPR	FNR	Prec.	AUC
100	0.107	0.893	0.943	0.057	0.844	0.156	0.937	0.95794
250	0.083	0.917	0.946	0.054	0.886	0.114	0.943	0.96877
500	0.084	0.916	0.943	0.057	0.890	0.110	0.940	0.97119
1000	0.081	0.919	0.947	0.053	0.892	0.108	0.944	0.97177
1600	0.079	0.921	0.937	0.063	0.906	0.094	0.935	0.97223

Conclusions/summary

- Android malware is growing in scale and complexity
- Static analysis coupled with machine learning is an effective tool for filtering apps to detect unknown Android malware
- Based on experiments with real malware samples
  - > 90% detection rate obtainable with low false positives using Bayesian classification.
- Higher detection rates are possible with the ML + static analysis method.
- Viable approach for filtering large amounts of apps added to Android market on daily basis (estimated 1200 daily).

CENTRE FOR SECURE INFORMATION TECHNOLOGIES

Thank you