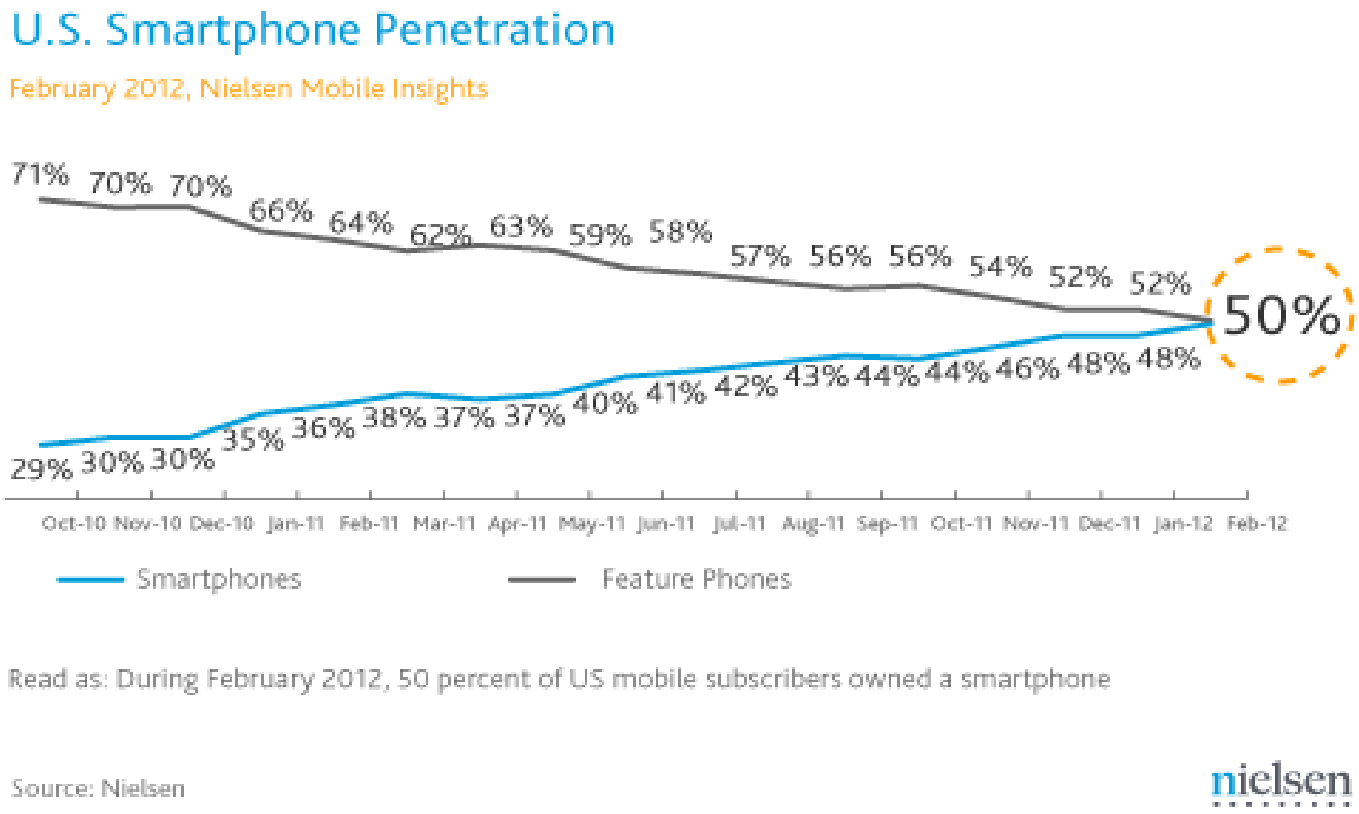
Machine Learning-based Techniques to Detect Malwares on Android-based Platform

Dr. Wei Yu
Department of Computer and Information Sciences
Towson University
http://www.towson.edu/~wyu
Email: wyu@towson.edu

Smart Mobile Device Market

- The sale of global smart mobile devices is expected to grow 25% from 472 million in 2011 to 630 million in 2012
- Smart mobile devices will reach nearly 1 billion by 2015

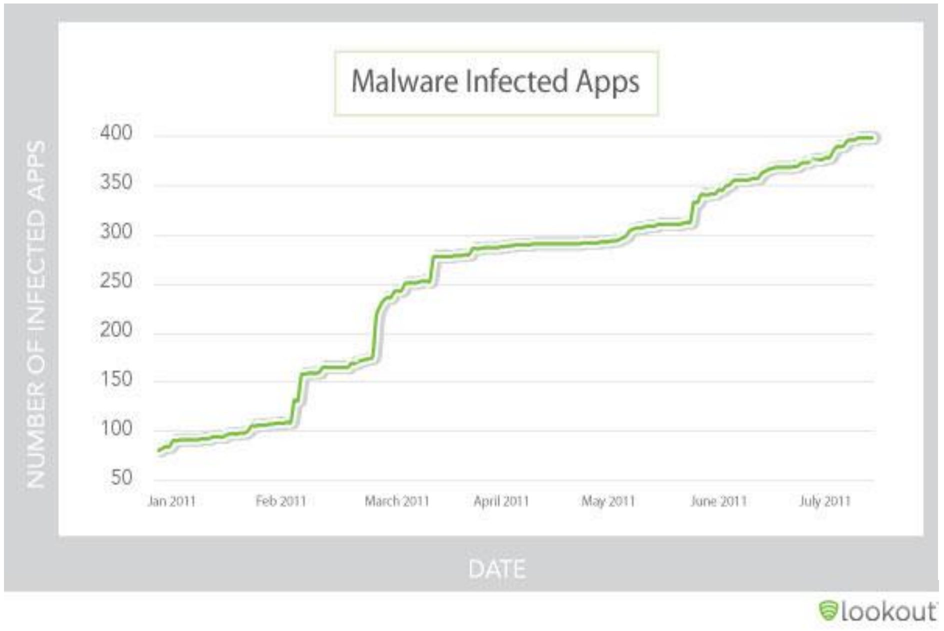


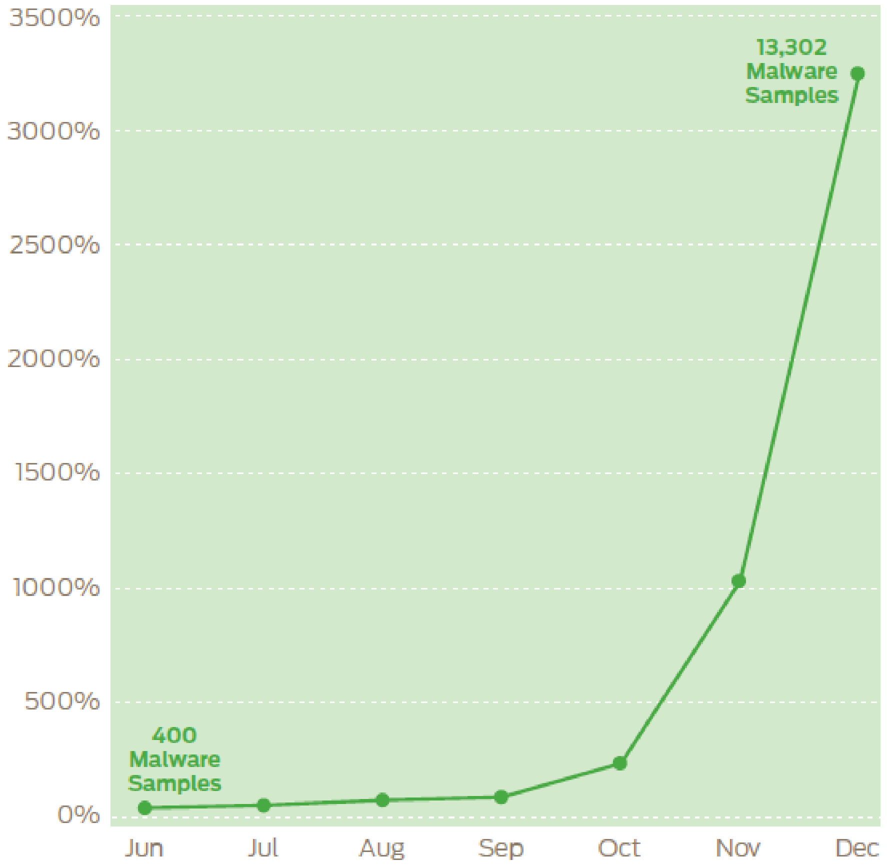
Android Platform Based Attacks

- Market leadership - attention of cybercriminals
- Android’s open application marketplace model
  - “Zero-day” application publishing
  - Unofficial third-party application store
- The availability, deployment time and updates to the operating system
  - Android’s open source model relies on mobile device manufacturers and service providers to push security patches through the devices
  - Device manufactures delay to push security updates

Growth of Android Platform-based Malwares

In the end of 2011, Lookout measured **4%** yearly likelihood of an Android user encountering malware, jumped from **1%** in the beginning of 2011





Android Malware Distribution

- Repackaged apps - Shotgun Distribution
  - Attackers will publish a large number of apps across multiple developer accounts and multiple markets
- Malicious advertising
  - Buy mobile ads, directing users to download malware on the Android Market or from a fake site designed to imitate the Android Market
- Update Attacks (DroidKungFu)
  - Release a legitimate application, updates the application with a malicious version
- Browser attacks - Drive-by-downloads
  - A web page automatically starts downloading an application when a user visits it, then encourage users to open the download.
  - 2/2012, malware on Facebook

Malware Capabilities

- Dial premium numbers or sends premium-rate SMS messages (Zsone, GGTracker, RuFraud)
  - Intercept any SMS messages from the SMS service to prevent a user from becoming aware of the charge
- Bots (ADRD, Geimini, Pjapps, Bgserv, jSMSHider, TigerBot, UpdtKiller) gives the malware writer remote control over all infected devices
  - Data exfiltration – device information, GPS location
  - Targets custom firmware devices
  - Install/remove apps, open web pages
  - Record phone calls
  - Capture and upload the image
  - Kill other running processes

System call on Android

- A system call is a mechanism used by an application for the requesting a service from the operating system’s kernel.
- For Android, we collected 56 calls in our library. The main types of system calls are as follows:
  - Process Control
  - File Management
  - Device Management
  - Information Maintenance
  - Communications

N-gram

- N-gram is a contiguous sequence of n items from a given sequence of text or speech
  - N is the length of the sequence, i.e., the number of system calls in one segment
- N-gram can map to relatively independent and meaningful actions taken during the program execution or the program blocks
- Examples
  - 1-gram: 2 3 4 6 3 5 7
  - 2-gram: 23 34 46 63 35 57
  - 3-gram: 234 346 463 635 357

Our Approach

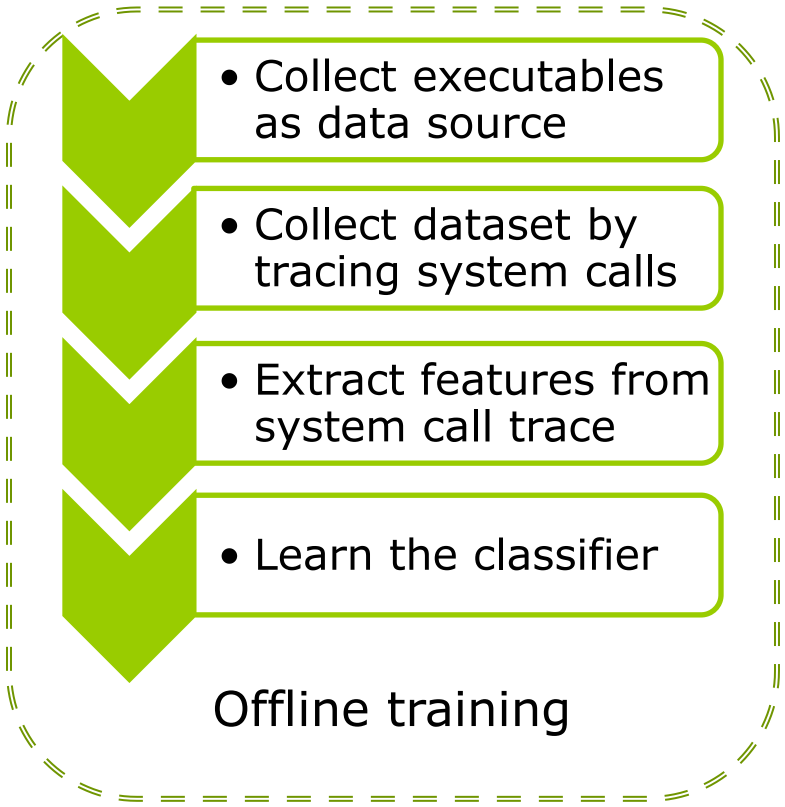
- Traditional static analysis is common used approach to detect malware on smart mobile devices
  - This approach is limited to detect applications that use a relatively small number of permissions and API calls.
  - Malware authors have developed various obfuscation techniques against static analysis.
- What programs do, i.e., their run-time behaviors or dynamic properties is the key
- We adopt the dynamic program analysis to profile the run-time behavior of program
  - Efficiently and accurately detect new malware executables

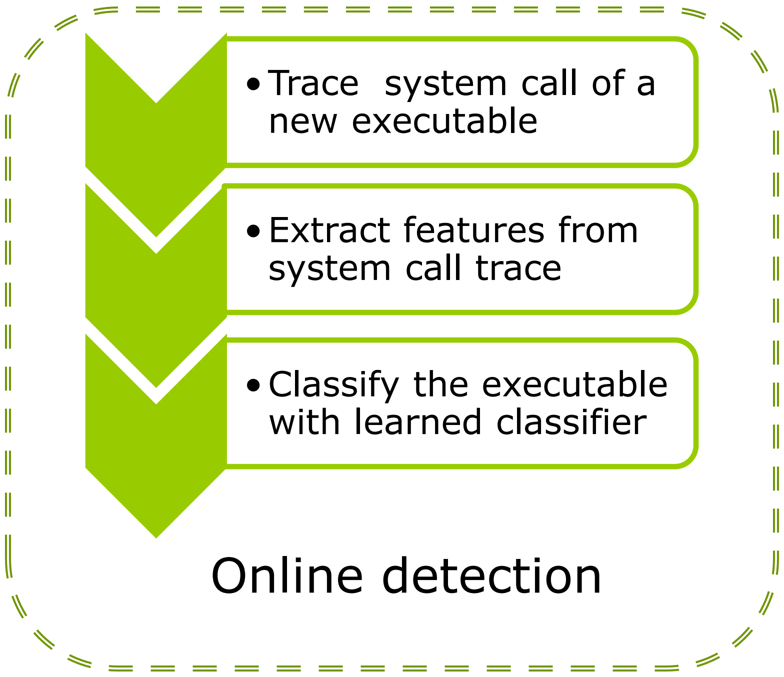
Challenges

- Dynamic program analysis poses three challenges
  - To capture the runtime behavior of executables (both malware and benign ones), we have to execute a large number of malwares
  - Manually executing and analyzing them is infeasible in practice.
    - Find an efficient way to automatically capture programs’ run-time behavior from their execution.
  - Find some constant and fundamental behavior differences between malwares and the benign executables

Workflow

- We dynamic execute the program and capture its dynamic behavior based on system call traces
- The overall workflow includes offline learning and online detection


Collect executables as data source
Collect dataset by tracing system calls
Extract features from system call trace
Learn the classifier
Offline training


Trace system call of a new executable
Extract features from system call trace
Classify the executable with learned classifier
Online detection

Offline Training -- Data Set Collection

- Experiments on two popular android devices:
  - Samsung Galaxy nexus
  - Google Nexus 7
- Collect around over hundred real-word android malware and benign programs

  - Root mobile devices
  - Install Strace
  - Install executables
  - Run executables and find process ID
  - Dump system calls
  - Pull output files from device to local PC

Machine Learning Algorithms

- Support Vector Machine (SVM)
- A Support Vector Machine (SVM) performs classification by constructing an N-dimensional hyperplane that optimally separates the data into two categories.
- The SVM is a machine learning algorithm which
  - solves classification problems
  - uses a flexible representation of the class boundaries
  - implements automatic complexity control to reduce overfitting
  - has a single global minimum which can be found in polynomial time

Machine Learning Algorithms (cont.)

- Bayesian classifiers are statistical classifiers. They can predict class membership probabilities, such as the probability that a given sample belongs to a particular class.
- Bayesian classifier is based on Bayesi theorem. Naive Bayesian classifiers assume that the effect of an attribute value on a given class is independent of the values of the other attributes

Offline Training -- Feature Extraction Support Vector Machine (SVM)

- The original output file consists of detailed system call information (e.g., function name, parameters, and others)

```
clock_gettime(CLOCK_MONOTONIC,{3277,54199221})=0
clock_gettime(CLOCK_MONOTONIC,{3277,54595950})=0
clock_gettime(CLOCK_MONOTONIC,{3277,54992678})=0recvfrom(37,
0xbead2950,24,64,0,0) = -1 EAGAIN (Try again) ioctl(10,
0xc0186201,0xbead3130) = 0 clock_gettime(CLOCK_MONOTONIC,
{3277,55969241})= 0 futex(0x408096a4,FUTEX_WAKE_PRIVATE,1)=1
futex(0x4080969c, FUTEX_WAKE_PRIVATE, 1) = 1 futex(0x408096a4,
FUTEX_WAKE_PRIVATE,1) = 1 futex(0x4080969c,
FUTEX_WAKE_PRIVATE,1)
= 1 recvfrom(38, 0x512597f8, 1240,64,0,0)=-1 EAGAIN
clock_gettime(CLOCK_MONOTONIC,{3277,59265139}) = 0
epoll_wait(0x1b,0xbead35f8, 0x10,0xffffffff) = 1
```

Offline Training -- Feature Extraction Support Vector Machine (SVM)

- Step1: We extract system call names as detection features:

```
clock_gettime  clock_gettime clock_gettime
recvfrom  ioctl clock_gettime futex futex
futex futex sendto recvfrom clock_gettime
epoll_wait
```

- Step2: The SVMLight only allows integer as input, we map the system call names to integers:

```
1 1 1 5 7 1 17 17 17 17 6 5 1 2
```

Offline Training -- Feature Extraction Support Vector Machine (SVM)

- Step2(for 2 gram): For the 2 gram case, instead of using one single system call as one SVM feature, we use the contiguous 2 system calls sequence as one SVM feature.
- To conduct the 2-gram mapping, we first zero fill every single digit integer, then combine every 2 contiguous integer together and got the output as follow:

```
0101 0101 0105 0507 0701 0117 1717 1717 1717 1706 0605
0501 0102
```

Offline Training -- Feature Extraction Support Vector Machine (SVM)

- Step3: After we have the integer sequence as feature, the next step is to get the value for the each feature.
- We calculated the frequency of each integer as the values of the features and get the feature value pair as below:

```
1:0.228305 2:0.0369626 3:0.0386701 4:0.0267176
5:0.084773
```

Offline Training -- Feature Extraction Support Vector Machine (SVM)

- Step4: We add target to each file, we add 1 to benign software and -1 to malware. Then put all files together and got the following output which can be used as SVMLight input.

```
-1 1:0.138216 2:0.0230937 3:0.0234893 4:0.0169617 5:0.0427257
-1 1:0.215698 2:0.0473563 3:0.0398686 4:0.0305419 5:0.0801314
-1 1:0.163538 2:0.0341676 3:0.0355171 4:0.0244755 5:0.0412219
-1 1:0.298989 2:0.0584028 3:0.0422061 4:0.0392505 5:0.0998995
 1 1:0.215374 2:0.040832  3:0.0406608 4:0.029233  5:0.0729327
-1 1:0.177037 2:0.031519  3:0.0328282 4:0.0238274 5:0.0615651
```

Offline Training -- Feature Extraction Naïve Bayes Classifier

- Naive Bayes classifier requires the input of table format. The first row is the column names, in our case is the integers represent the feature names and type.
- Use the feature value pairs we got in step3, we wrote Python scripts to parse them into table format. Then we assign types to each line. Type A means it is benign software while type B means it is malware.

1	2	3	4	5	type
0.228305	0.0369626	0.0386701	0.0267176	0.084773	A
0.298989	0.00697523	0.0359993	0.0289058	0.000118224	A
0.257303	0.0046592	0.0116742	0.00701497	0.00303633	A
0.120521	0.0117298	0.0123743	0.00554267	0.0260376	B
0.242534	0.00647457	0.010266	0.00612459	0.00974102	A

Offline Training – Classifier learning(SVMLight)

- We use the SVMlight learning module to learn the training data.
- We apply the linear model as kernel function.
- The learning module(*svm_learn*) is called with the following command:

```
svm_learn training_file model_file
```

- The training_file is the input of the function, contains the training data which was generated in step 3. The model_file is the output file, it contains the SVM analysis of the training file.

Offline Training-Classifier Learning(Naïve Bayes)

- We use the program *dom* to determine the feature domains. The *dom* is called by the following command:

```
dom -a all.tab all.dom
```

- The *all.tab* file is the input of this function, it contains all of our data. The -a option tells the program to determine automatically the column data types. The *all.dom* is the output file which illustrates the domain of features.

Offline Training-Classifier Learning(SVMLight)

- We use the *bci* program to use the training data to train the classifier. The *bci* program is called with the following command:

```
bci all.dom training_file model_file
```

- The *bci* program take the *all.dom* and the *training_file* as input. The *model_file* is the output file, it contains the Naive Bayes analysis of the training file.

Online Detection

- The first two steps of online detection are similar with the offline training steps.
  - First step is to run it and dump the system call.
  - Then process the system call to meet the format of machine learning tools input.
- In the end we use the machine learning tools to classify whether it is a benign software or a malware.

Online Detection-- SVMLight

- We use the classification module(*svm_classify*) to classify our test data. The *svm_classify* is called with the following command:

```
svm_learn test_file model_file result_file
```

- The classifier takes the *model_file* and the *test_file* as input files. Generate the *result_file* as output shown as below:

```
0.47027026 1.3260829 1.132413 1.1246804 0.4504972 -0.58676456
0.41940827
```

- Each number presents the classify result on one software. When the number is positive, the SVM classify it as benign software, and the negative number means the SVM classify it as malware.

Online Detection– Naïve Bayes

- We use the *bcx* program to classify our test data. The *bcx* program is called with the following command:

  ```
  bcx  model_file test_file result_file
  ```

- The classifier takes the *model_file* and the *test_file* as input files. Generate the *result_file* as output shown as below. The *result_file* is in table format, the 6, 7, 8, 9 represents the feature names. The TYPE represents the accrual software type, and the bc represents the classification result. we use type B to stand for the malware and use A to stand for the benign software.

```
6           7         8          9          TYPE    bc
0.00294442  0.0866151 0.0127592  0.0103055  B       B
0.00294442  0.0866151 0.0127592  0.0        B       B
0.00294442  0.0866151 0.0127592  0.0        B       B
0.0135927   0.11639   0.025689   0.0185809  B       A
0.0415218   0.0985558 0.0177186  0.0235357  B       B
```

Experiment Environment

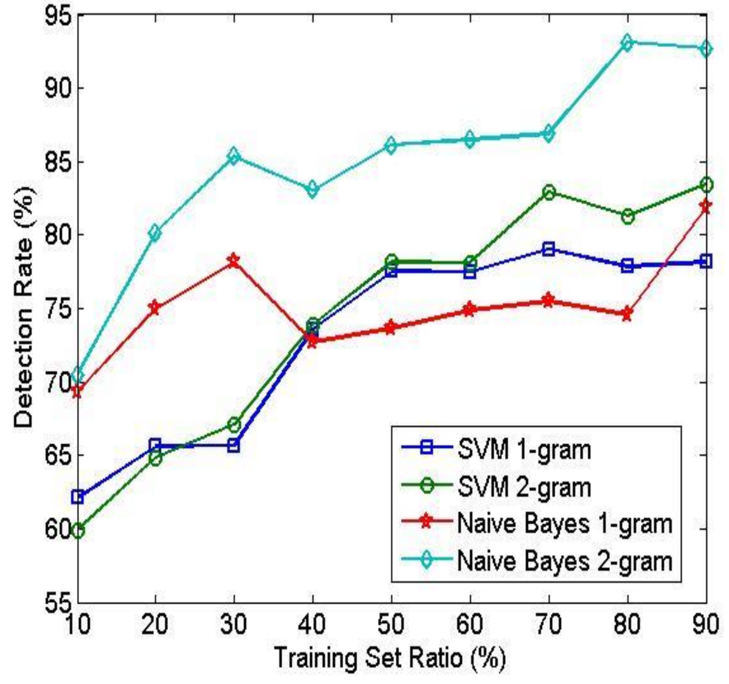
- We used Sumsang Galaxy Nexus and Google Nexus 7 as our experimental mobile platforms.
- The PC hardware platform is DELL Optiplex 980 equipped with Intel core 3.20GHZ i5 processor rated at 8GB RAM and 320GB hard drive.
- The Mobile OS is Android 4.2 and On PC, we use ubuntu 12.04.
- We collected 96 normal applications form Google play 92 digital book malware samples from the Android Malware Genome Project at North Carolina State University.

Evaluation Metrics

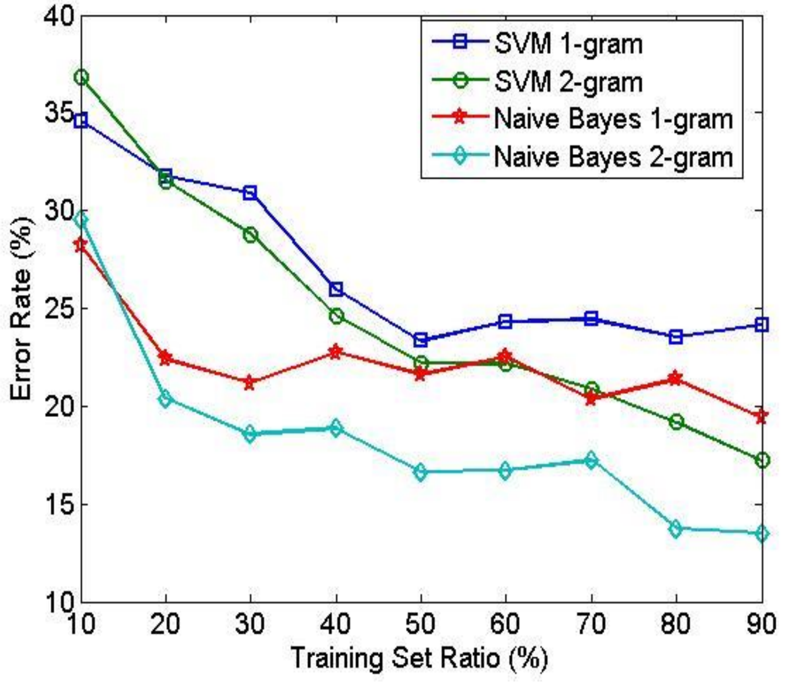
- We use the following 4 metrics to evaluate our detection approach.
  - *Detection rate*: Defined as the probability of correctly classifying the malware, which is the ratio that correctly classified malware vs. total malware samples.
  - *Error rate*: Defined as the probability of falsely classifying the class of applications, which is the ratio that applications falsely classified as other class vs. total applications.
  - *Training Time*: Defined as the duration of machine learning algorithms executes offline training process.
  - *Detection Time*: Defined as the duration of online detection process.

Evaluation Results

- Detection rate increases while error rate decreases as the training set ratio grows.



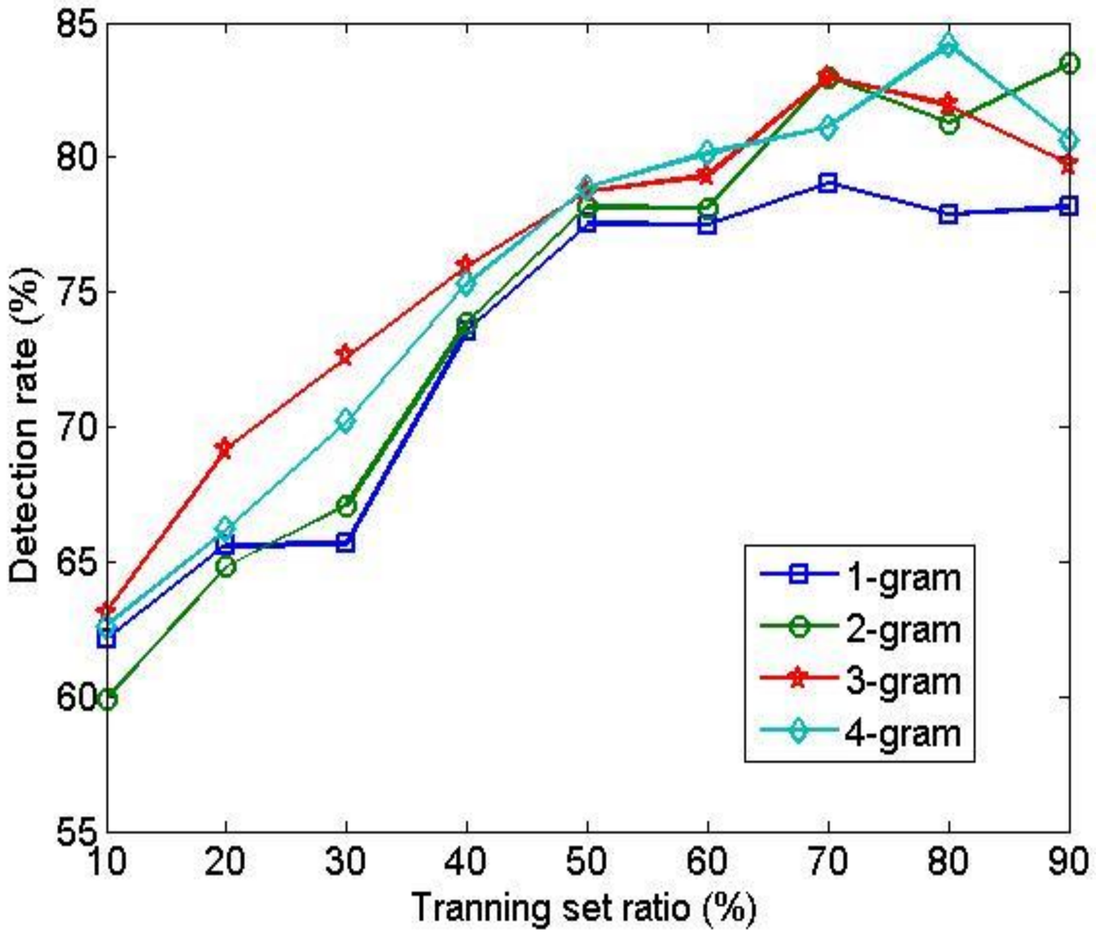
Detection rate vs. Training set ratio



Error rate vs. Training set ratio

Evaluation Results

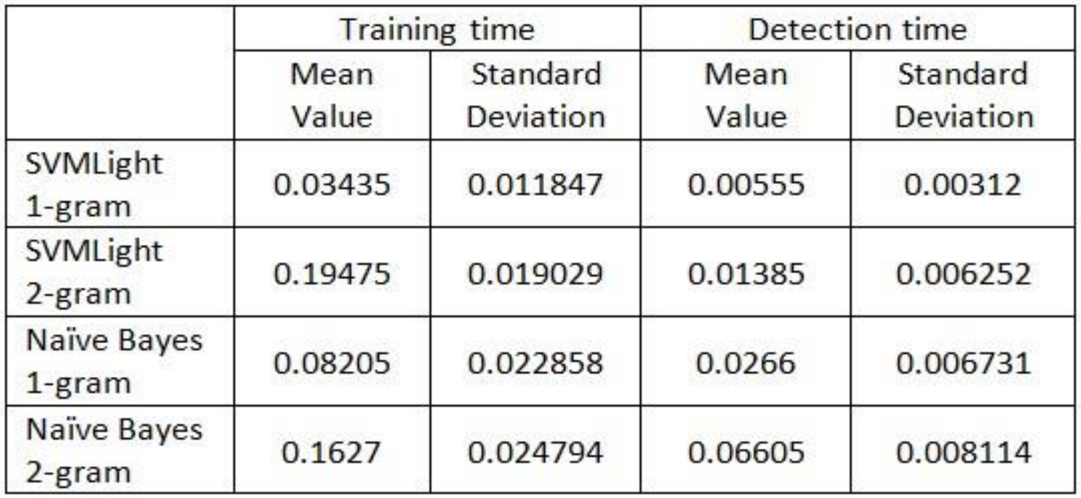
- Higher degree of gram achieves better performance.



Detection rate vs. Training set ratio

Evaluation Results

- Repeat each experiment 20 times and calculated the mean value and standard deviation.
  - Training time measures training duration for 100 software.
  - Detection time measures testing duration for 1 software



	Training time		Detection time	
	Mean Value	Standard Deviation	Mean Value	Standard Deviation
SVMLight 1-gram	0.03435	0.011847	0.00555	0.00312
SVMLight 2-gram	0.19475	0.019029	0.01385	0.006252
Naïve Bayes 1-gram	0.08205	0.022858	0.0266	0.006731
Naïve Bayes 2-gram	0.1627	0.024794	0.06605	0.008114

Training and Detection Time Efficiency