System Security – Malware Defense II

TDDE62 – Information Security:
Privacy, System and Network Security

Ulf Kargén

Department of Computer and Information Science
Linköping University

LINKÖPING UNIVERSITY

Original slides by Alireza Mohammadinodooshan

What Has Been Covered …

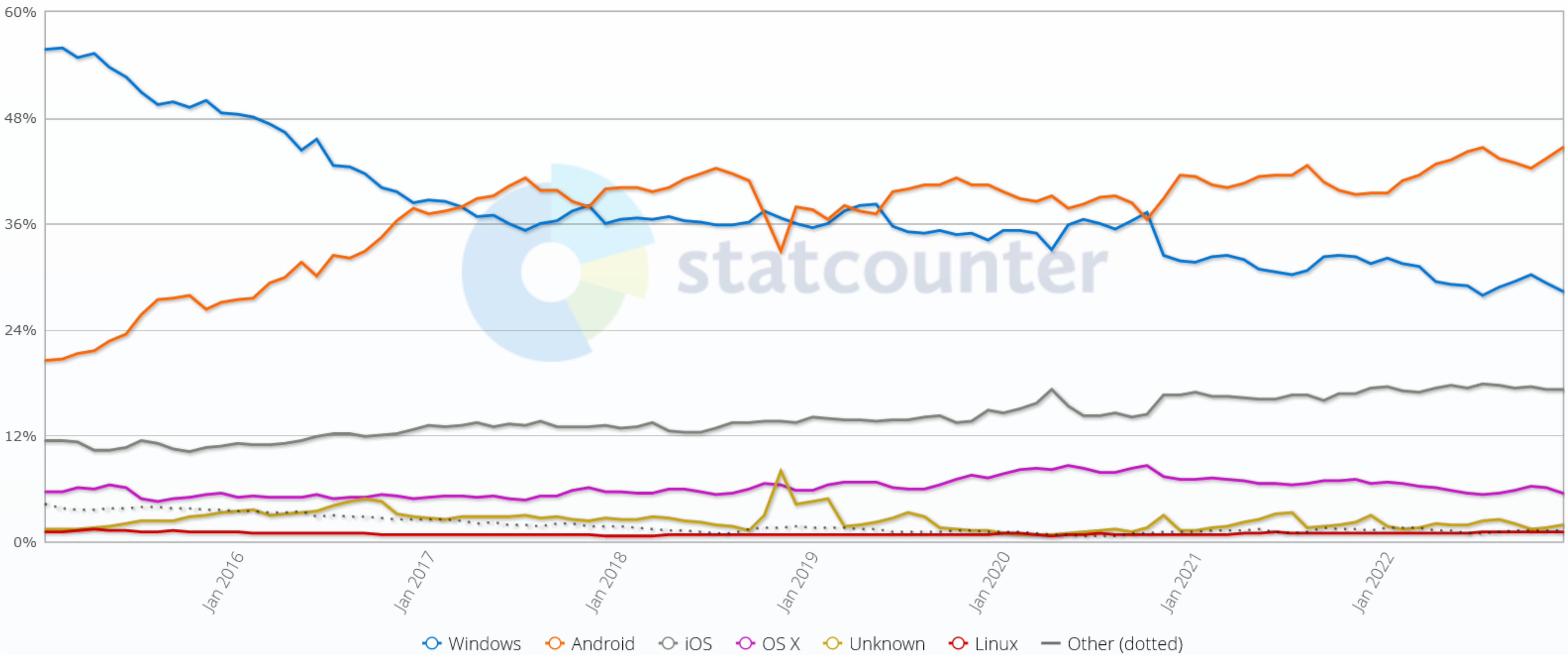
- Malware basics
  - Different types of functionality
  - Different infection methods
- AV cat and mouse game
  - Signatures based detection
  - More complex signatures and static heuristics
  - Static unpacking and emulation
  - Cloud-based detection
  - Machine learning detection

Agenda

- Mobile malware
  - Specific challenges
  - Specific risks
  - Security models and their effect on malware detection
    - iOS
    - Android
  - Detection countermeasures
- Machine learning for malware detection
  - Motivation
  - Terminology
  - Learning types
  - Machine learning-based malware detection challenges

Motivation

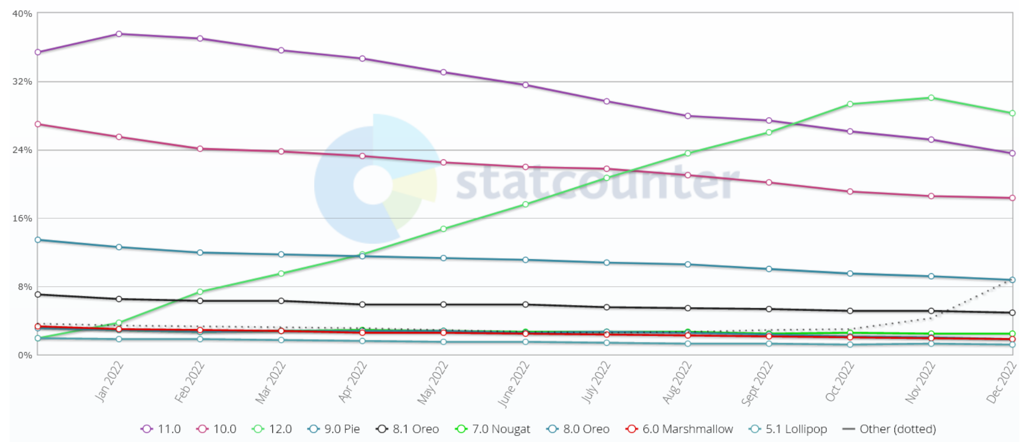
- 5.48 billion smartphone users in the world in 2022

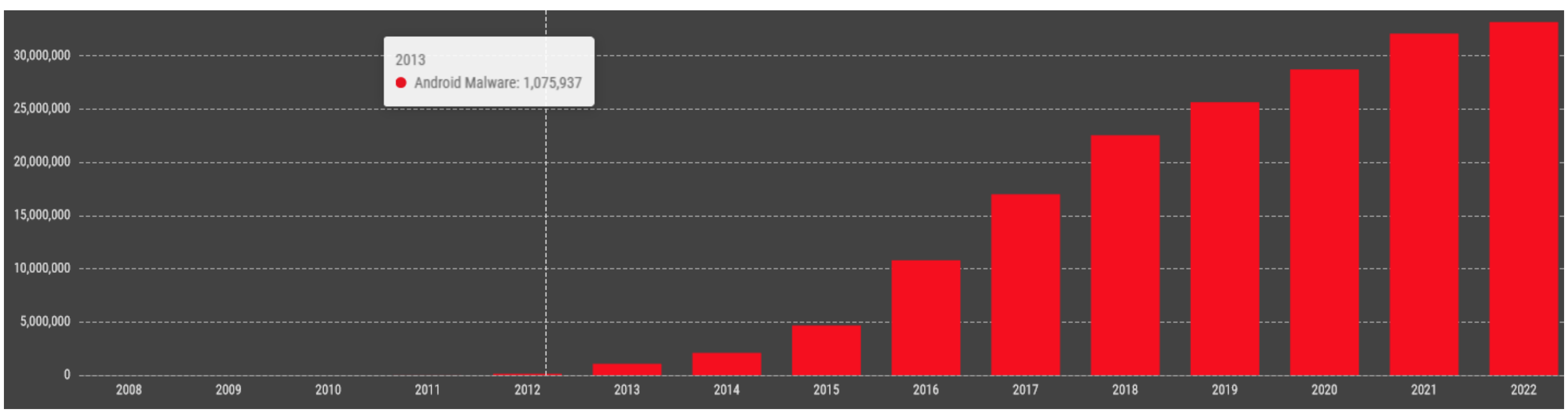


https://gs.statcounter.com/os-market-share
https://datareportal.com/global-digital-overview

Motivation

- Many phones with old versions of Android still around
- It is not surprising that the mobile platform became an appealing target for malware authors.





https://portal.av-atlas.org/malware/statistics

https://gs.statcounter.com/android-version-market-share/mobile-tablet/worldwide

Mobile Malware Definition

- Malicious software designed to attack mobile devices
  - Phone
  - Tablet
  - Watch
  - TV

Samples of Mobile Malware

- iOS stock
  - PawnStorm.A
    - Able to upload GPS location, contact list, photos to a remote server.
  - YiSpecter
    - Able to download, install and launch arbitrary apps
- Android
  - Android/Filecoder.C
    - Able to spread via text messages and contains a malicious link. Encrypts all of your local files in exchange for a ransom between $94 and $188.
  - Plankton
    - Communicates with a remote server, downloads and install other applications and sends premium SMS messages

Mobile Malware Specific Challenges

1. Personal-info and privacy concerns
   - Banking info
   - Personal photos
   - Contact info
2. Widespread access to networks
   - 4G
   - Wifi
   - Bluetooth

Mobile Malware Specific Challenges

3. Less computation power
   - Limited capabilities for on-device detection
4. Almost exclusively trojans
   - Repackaging
     - Add malicious functionality to a legitimate app, and re-release under own Android developer ID.
       - Much easier to reverse-engineer and modify Android apps than, e.g., PC software
     - A very simple technique is to replace the advertisement logic and re-bundle and publish the app
   - Fake apps also exist!

Mobile Malware Specific Challenges

5. Due to limited computation power, most of the trust in apps is moved to app stores to analyze the apps
   - While for the 3rd party stores and perhaps to a degree even for the Google Play store, this is a mistrust (we will elaborate on this ...)
   - Attackers also have the motivation to deliver their malware through stores (official or third party)

Mobile Malware Specific Challenges

6. Harder to detect with 3rd party AV on the device compared to PC malware due to stronger isolation between apps
   - Memory isolation
   - User isolation
     - Each app is treated as a separate user
     - Applications cannot interact with each other, and they have limited access to the system as well as other apps resources

Mobile Malware Risks

- System damage
  - Battery draining
  - Cryptocurrecy mining
  - Disabling system functions
    - Block calling functionality
    - Litter phone UI with ads
- Economic
  - Sending SMS or MMS messages to premium numbers
  - Dialing premium numbers
  - Deleting important data

Peng, S., Yu, S., & Yang, A. (2013). Smartphone malware and its propagation modeling: A survey. IEEE Communications Surveys & Tutorials, 16(2), 925-941

Mobile Malware Risks

- Information leakage
  - Privacy-sensitive data (personal photos, contacts, etc.)
  - Stealing bank account information

- Disturbing mobile networks
  - Denial-of-service (DoS)

iOS Security Model

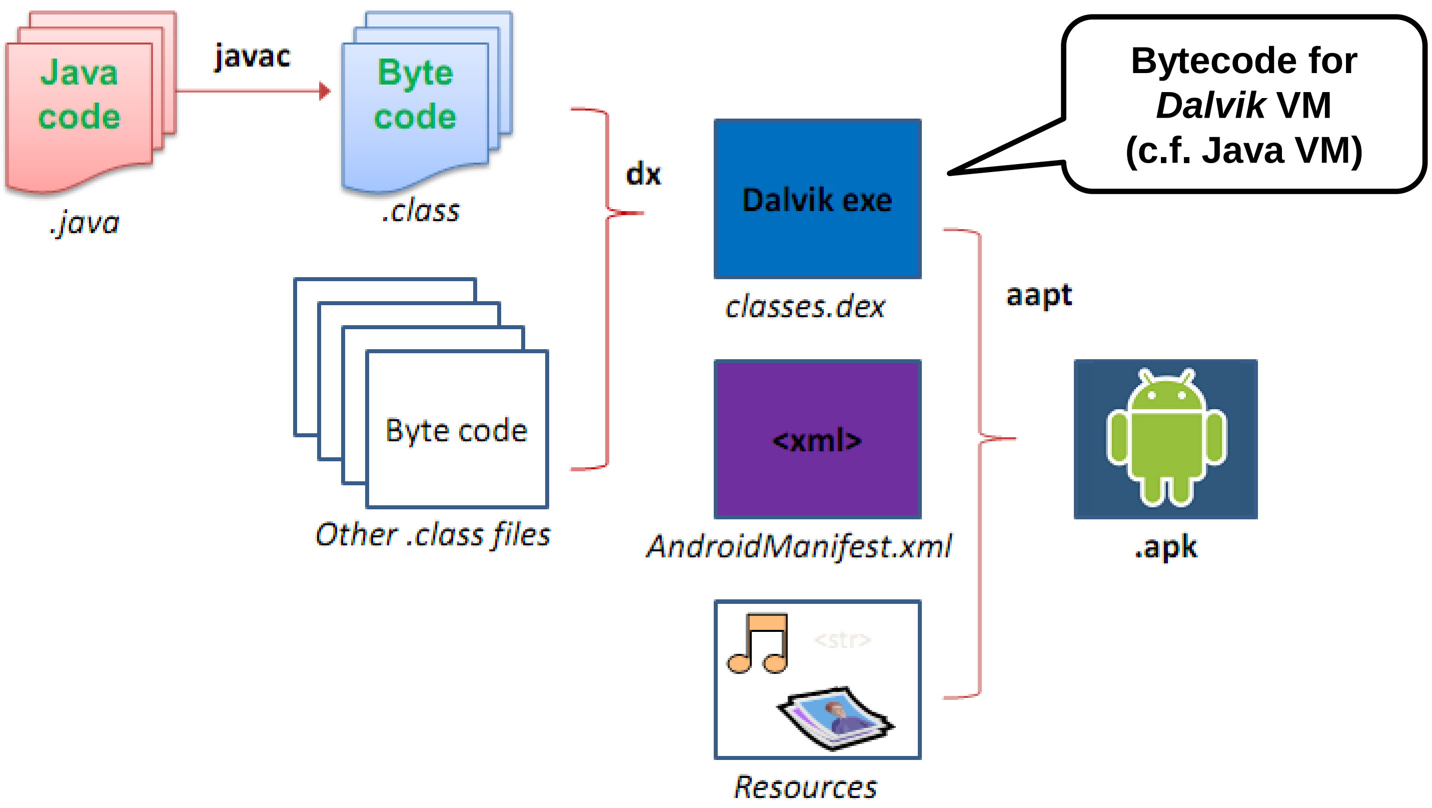
- System Security
  - Startup and updates are authorized
- Data security
  - File-level data protection uses strong encryption keys derived from the user's unique passcode.
- App security
  - Application run in their sandboxes.
  - More important than this …

https://developer.apple.com/app-store/review/

iOS Security Model

- Before releasing on store, they go through a strict vetting process
  - Manual testing
  - Static analysis
  - Apps can not do actions outside of what they claim

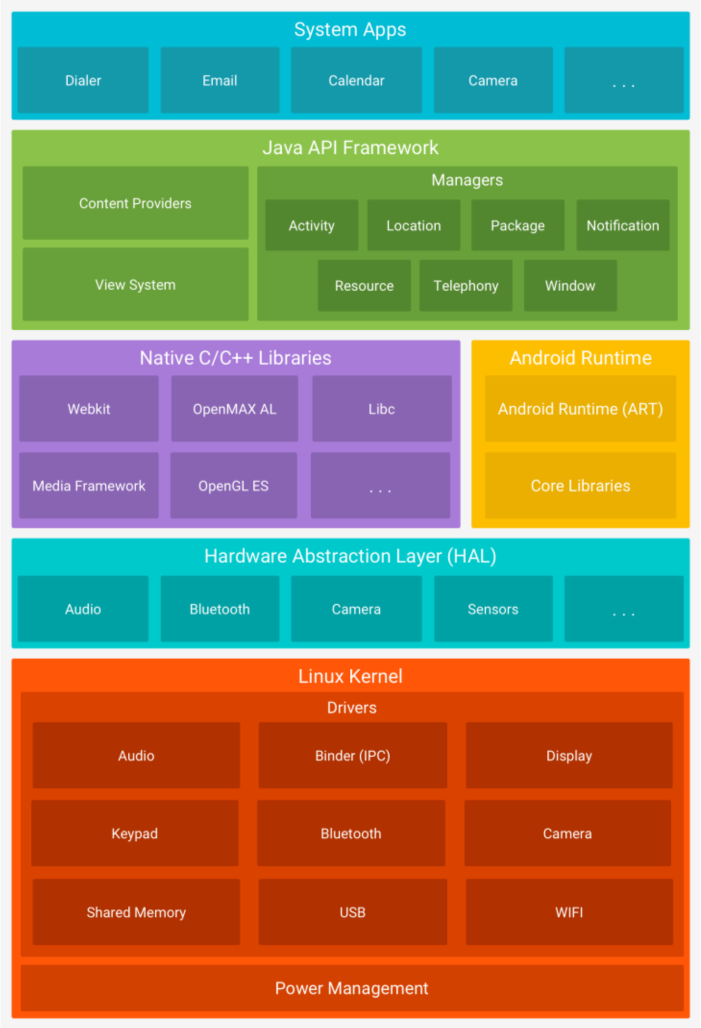
Android Application Compiling



https://justamomentgoose.wordpress.com/2013/06/04/android-started-note-2-android-file-apk-decompile/

Android Architecture

- Android Runtime
  - Each app runs in its own process and with its own instance of the Android Runtime (ART).
  - It is possible to have compiled C/C++ code packaged with an Apk which can be called through Java Native Interface (JNI)
  - Apps are pre-compiled from Dalvik bytecode to native code during installation
    - Old Android versions ran Dalvik bytecode directly in a VM



https://developer.android.com/guide/platform

Androidmanifest.xml

- Provides the essential information to the Android system regarding this app
  - Minimum Android API
  - Linked libraries
  - Components, activities, services, …
  - Required permissions

Android Security Model

- Application Sandboxing
  - Android automatically assigns a unique Linux user ID to each app at installation
    - Each app runs as a unique “user” on Android
  - App is allowed to access:
    - Own files
    - World-accessible resources
  - More access:
    - Managed through defining in the *androidmanifest.xml*

    E.g.: `<uses-permission android:name="android.permission.READ_PHONE_STATE"/>`

Android Vetting Process

Android does not require an exhaustive app vetting process

- More lenient compared to iOS
- Apps are dynamically tested with a Google security service known as *Bouncer*
  - Attempts to exercise different code paths by interacting with app in simulator while checking for malicious behavior
  - The results are combined with the output coming from the Google reputation system
- Researchers have shown the feasibility of fingerprinting Bouncer*
  - Android ID, phone number, etc.
  - Malware may be able to bypass Bouncer by not displaying malicious behavior within Bouncer

* https://jon.oberheide.org/files/summercon12-bouncer.pdf

Mobile Malware Detection

- Static Code Analysis
  - Signature-based techniques
    - Specific strings or patterns in the byte code
    - Extracting the strings is straightforward
  - Permission-based techniques
    - Analyzing the requested permissions to identify the potential malware samples – useful for heuristic flagging of potential malware
  - Dalvik bytecode-based techniques
    - Analyzing the byte code to identify malicious Android samples (API calls, data flows, …)

Mobile Malware Detection

- Dynamic Behavior Analysis
  - Sequence of system calls
  - Accessed files

- Hybrid Analysis

Malware Detection Countermeasures

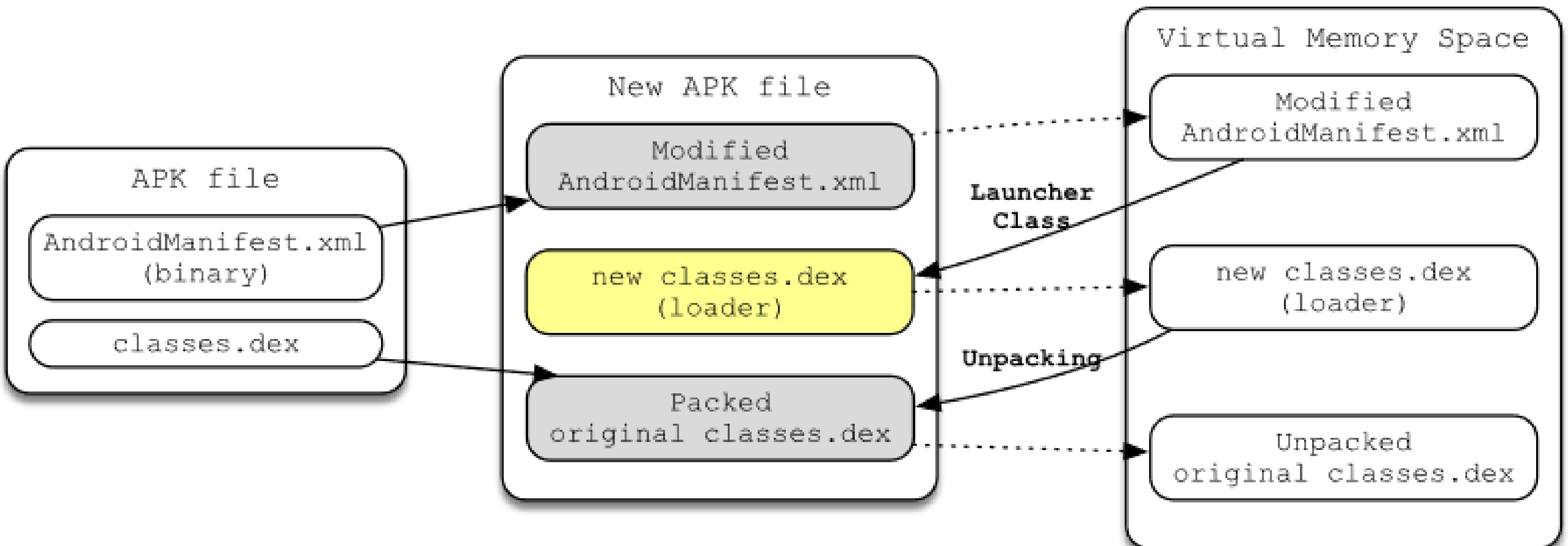
- Static
  - Obfuscation
    - Making the byte code hard to understand
    - Making signature or even some static heuristics-based analysis harder
  - Packing
- Dynamic
  - Sandbox detection
    - Many of the sandboxes still do not have real device behaviors
      - E.g. do not support GPS or do not have a real GPS accuracy

Obfuscation

- Identifier renaming
  - Replace identifiers (e.g., variable or method names) used in the source code with meaningless names, e.g., ’a’, ‘b’, ‘aa’, ‘ab’, ‘ac’
  - Mostly used to prevent humans from reverse-engineering apps
- String encryption
  - Replacing constant strings with their encrypted form and adding the code to decrypt them on the fly
- Control-flow obfuscation: changing the logical flow of the program
  - Injecting dead code
  - Re-ordering statements
  - Inserting *opaque predicates*

```
obj = benign()
var1 = 10
var2 = [var1 for i in range(10)]
if var1 == var2[0]:
  obj = malware()
obj.load()
```

Packing


APK file
AndroidManifest.xml (binary)
classes.dex
New APK file
Modified AndroidManifest.xml
new classes.dex (loader)
Packed original classes.dex
Launcher Class
Unpacking
Virtual Memory Space
Modified AndroidManifest.xml
new classes.dex (loader)
Unpacked original classes.dex

Machine Learning for Malware Analysis

LINKÖPING UNIVERSITY

Why?

- Creating detection rules (signatures) manually couldn’t keep up with the emerging flow of malware.
  - Zero-day malware
- Need a more reliable method when we know that the relation between the sample features is hard to find for the human
- Sometimes we need a triage method
  - A procedure we use to prioritize the samples that should be examined

Machine Learning

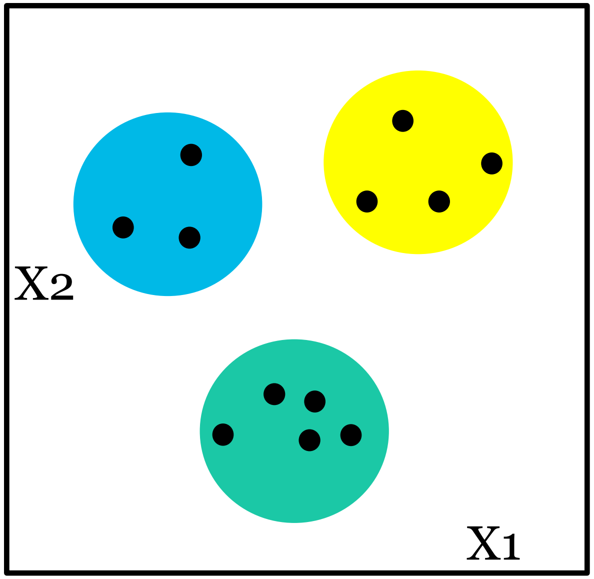
- Machine learning is a set of methods that gives computers the ability to learn without being explicitly programmed
  - Learning from the data
  - It is used when we want to (explicitly or implicitly) learn the relation using some available data (known as training data)

Terminology

- (Predictive) Model: The hidden relation
- Training data: Data based on which we make the model
- Testing data: Data based on which we evaluate the model
- (Hidden relation) Learning types:
  - Unsupervised
  - Supervised

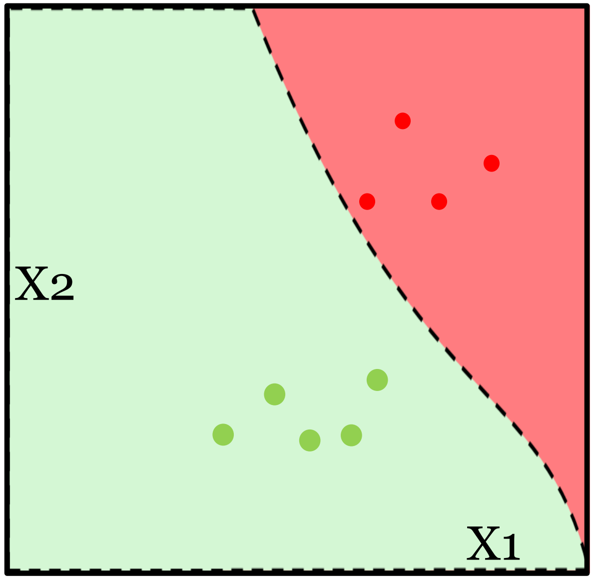
Unsupervised learning

- Given features **X** (X1 and X2 in the following figure)
- The goal is to *discover* the structure of the data
  - Clustering: splitting a data set into groups of similar objects
    - Application example
      - Grouping malware into potential families



Supervised Learning

- Having *both* **X** (X1 and X2 in the following figure) and **y** (the colors in the figure) we try to learn the relation between them (**X** and **y**)
- For example, malware detection:
  - **X**: features of malware and benign apps
  - **y**: "*malware*" or "*benign*" label



Classification vs. Regression

Classification

- When **y** is a *categorical* variable
  - For example, “malware” or “benign” (binary classification)
- Also: So called one-class classification or **anomaly detection**
  - Train classifier to learn distribution of expected (or “normal”) data
  - Detect samples that *deviate* too much from the training data

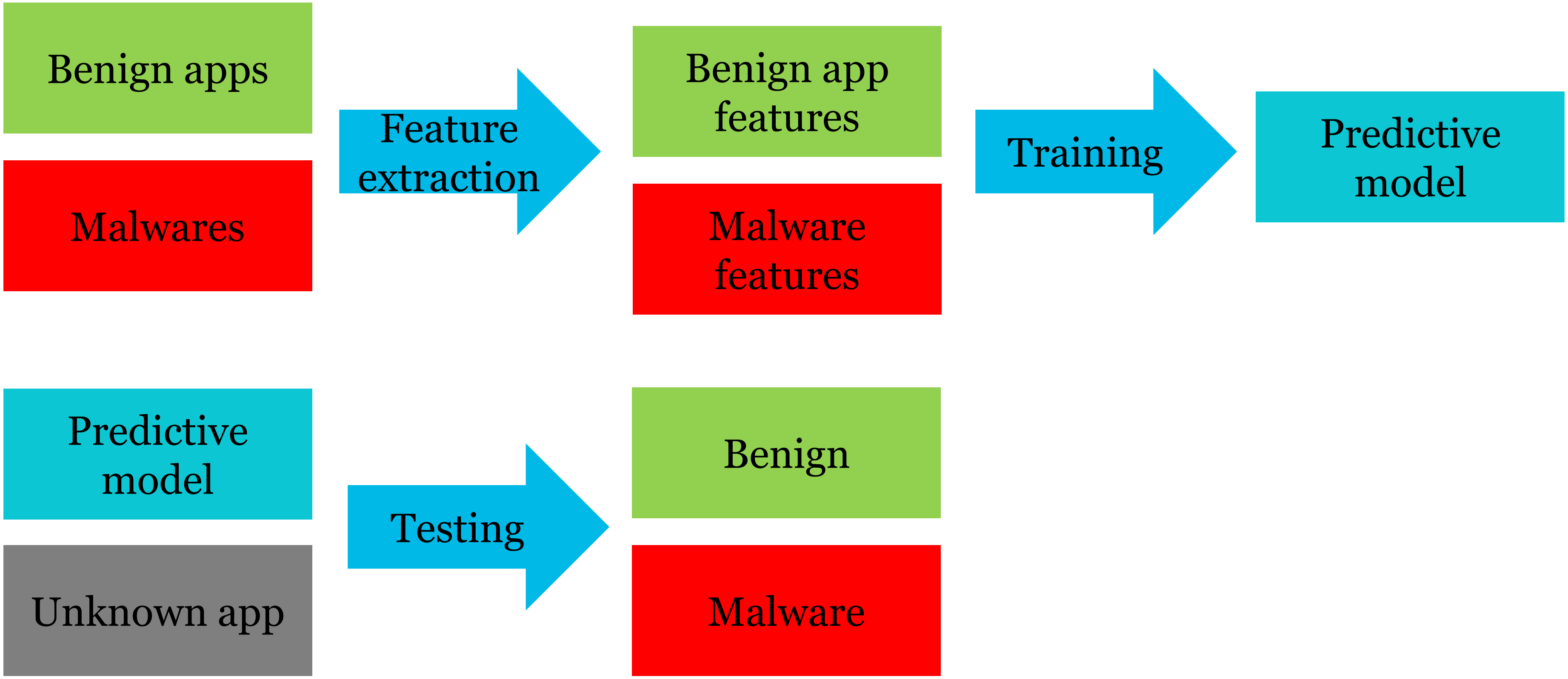
Regression

- When **y** is a *continuous* variable
  - For example, probability of belonging to a specific malware family (e.g., can be used for triaging the app)

ML-based Malware Detection procedure

- Collecting training data
- Extracting features from training data
- Training the model: finding the model
- Testing (Evaluating) the model

ML-based Malware Detection Workflow



Collect Training data

Dataset should be representative of real-world malware

- Example of bad practice
  - Suppose that we collected some benign and malware samples, but
    - all benign apps happen to have sizes > 1 MB, and
    - all malware samples happen to be < 100 kB
  - Not representative of all malware/benign apps …
  - The model overfits to this unrealistic pattern
    - For example, model might classify *all* small apps as malware!

Extracting Features

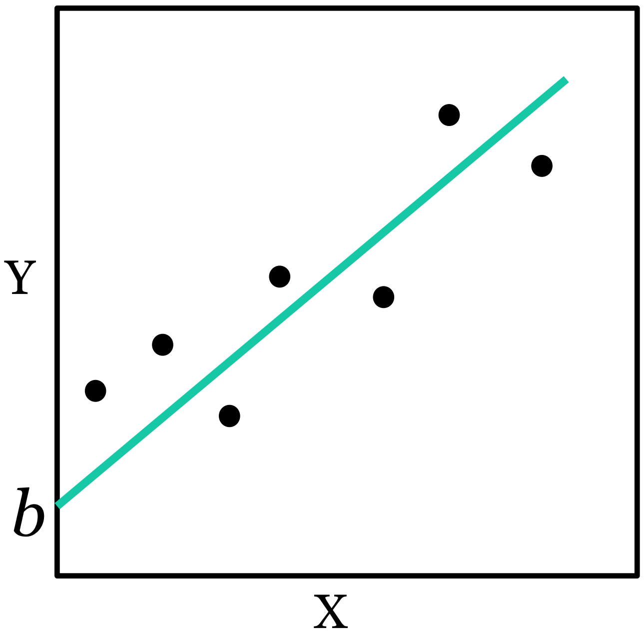
- The extracted features should be relevant.
- Usually, domain knowledge helps a lot here
  - Examples
    - PC
      - The header values of executables
    - Mobile
      - Set of privileges (in androidmanifest.xml)
    - Both
      - Obfuscation status
  - Feature selection methods can be used to limit the number of features
    - For example, low-variance features can be removed (i.e., having similar values for both benign and malcious apps)

Training

- Models have some *parameters* which during the training phase are optimized using the training data
  - This optimization happens based on a particular metric.
  - This particular metric is usually the classification or regression error

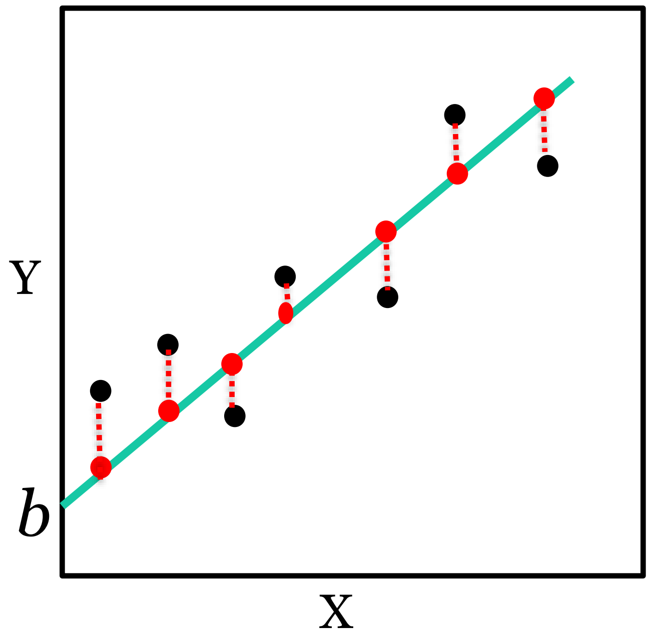
Training (Example)

- Linear regression
  - We have a set of (X_i, Y_i) *training points*
  - We want to find the regression line
    - Which with the least error estimates the points
    - $F = aX + b$
      - $a_{opt\,?}$
      - $b_{opt\,?}$



Training(Example)

- Learning workflow
  - For each point X_i compute the response F_i
    - $F_i = aX_i + b$
  - Compute $ERR_{tot} = SUM((F_i - Y_i)^2)$
  - Now we can compute a_{opt} and b_{opt}
    - Which minimizes ERR_{tot}
      - Closed form
      - Optimization
- This was a regression example
  - For the classification, for example
    - We can find the discriminative line or hyperplane between the points



Testing

- After finding the optimal values of parameters (in this case $\boldsymbol{a}$ and $\boldsymbol{b}$) we test it on *testing data*.
  - To see whether it can generalize to unseen data
  - Or it has just memorized the training data
- In this case (testing) we will also have some error
  - We train a model by minimizing its error on the *training data*
  - The training error is different from the testing error
  - This testing error value is computed on test data
    - Must not overlap with training data – otherwise testing results will be biased

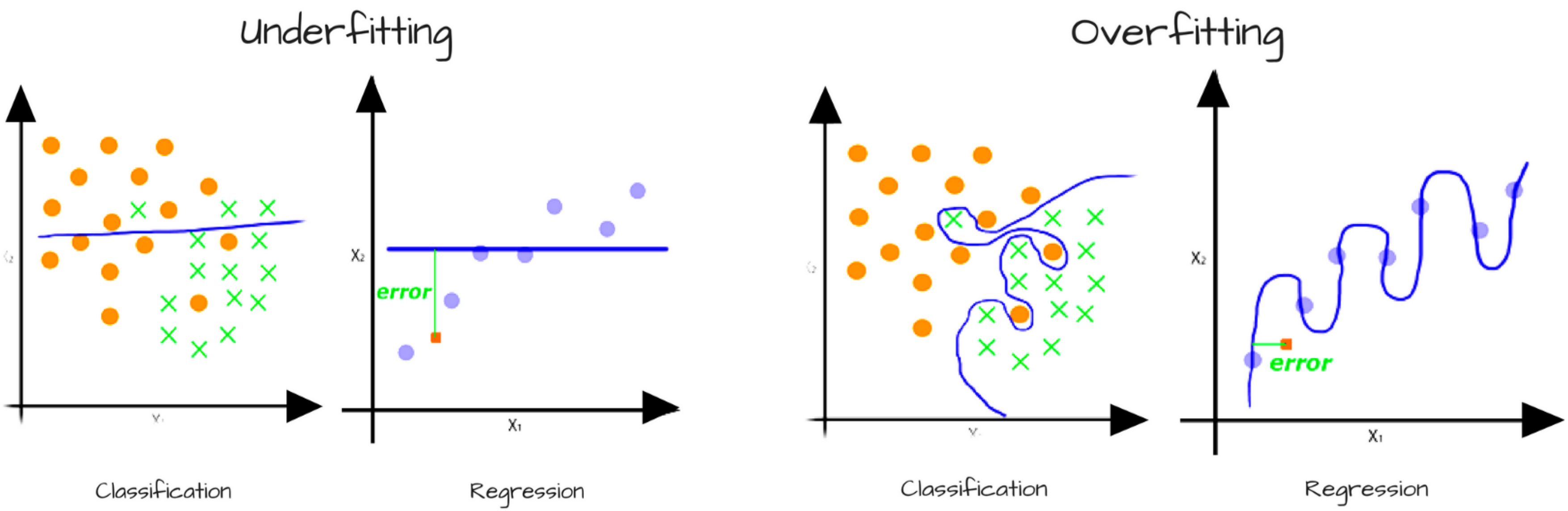
Machine Learning-based Malware Detection Challenges

- Under- and Over-fitting
- Imbalanced datasets
- Performance evaluation measures
- Dataset quality

Underfitting and Overfitting

- Underfitting
  - The model is unable to obtain a low error even on the training set
    - Model might be too simple (too few parameters) to accurately reflect training data – too low *learning capacity*
- Overfitting (Memorization)
  - The training error is small, but not the testing error
    - Model might have *too many parameters* compared to the volume of training data – *too high* learning capacity

Underfitting and Overfitting

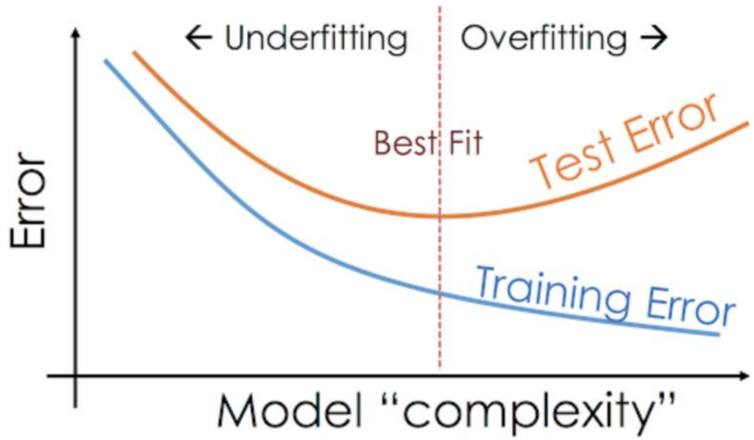


https://blog.booleanhunter.com/using-machine-learning-to-predict-the-quality-of-wines/

Underfitting and Overfitting

Solution: Adjust model complexity to minimize error

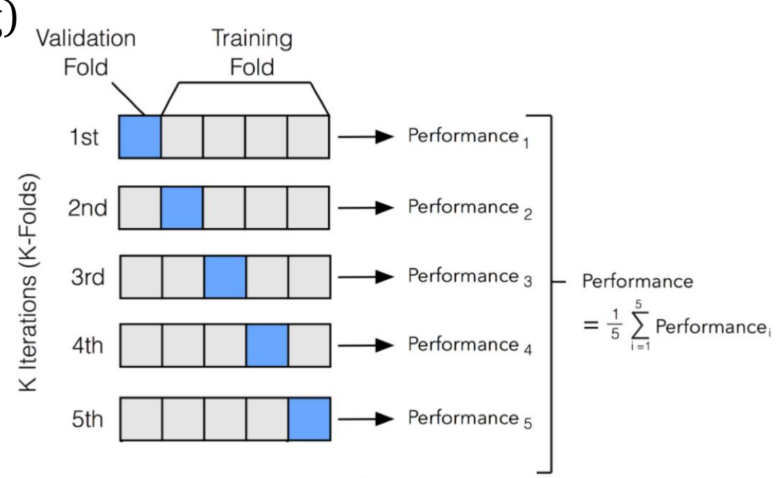
- Most ML algorithms have several tunable *hyperparameters* (= parameters **not** learned directly from training data)
  - Number of hidden layers in neural networks, maximum depth of decision trees in random forest, etc.
- Hyperparameter tuning: Test different combinations of hyperparameters until we get the best generalization on testing/validation data
- New problem: What if we don’t have enough data to “spare” for a separate test set?



https://vitalflux.com/overfitting-underfitting-concepts-interview-questions

Cross-Validation

- Basic idea
  - Each observation in our dataset has the opportunity of being tested
- Procedure for *k-fold cross validation*
  - We divide the dataset into *k* sets
  - For k rounds, we go over the dataset, and in each round (or *fold*):
    - One part is used for validation (testing)
    - Remaining parts used for training
    - Based on the average performance value across all *k* folds, we can select the optimal hyperparameters



http://ethen8181.github.io/machine-learning/model_selection/model_selection.html

The problem of imbalanced datasets

- Malware datasets are usually imbalanced
- Suppose that we have a dataset in which 99 percent of samples are benign
  - Now a naïve malware detection classifier which classifies all the samples as being benign reaches an accuracy of 99 percent
  - Probably no other model can reach this optimal accuracy
  - But is accuracy a good metric to train the model on?
  - Evidently not. This model cannot detect any malware!
    - Accuracy only meaningful when we have a 50/50 distribution of malware and benign samples
- We need to focus on some other performance measures!

Performance Measures

- Accuracy

$$\frac{TP+TN}{TP+FP+TN+FN}$$

- Recall (Sensitivity)

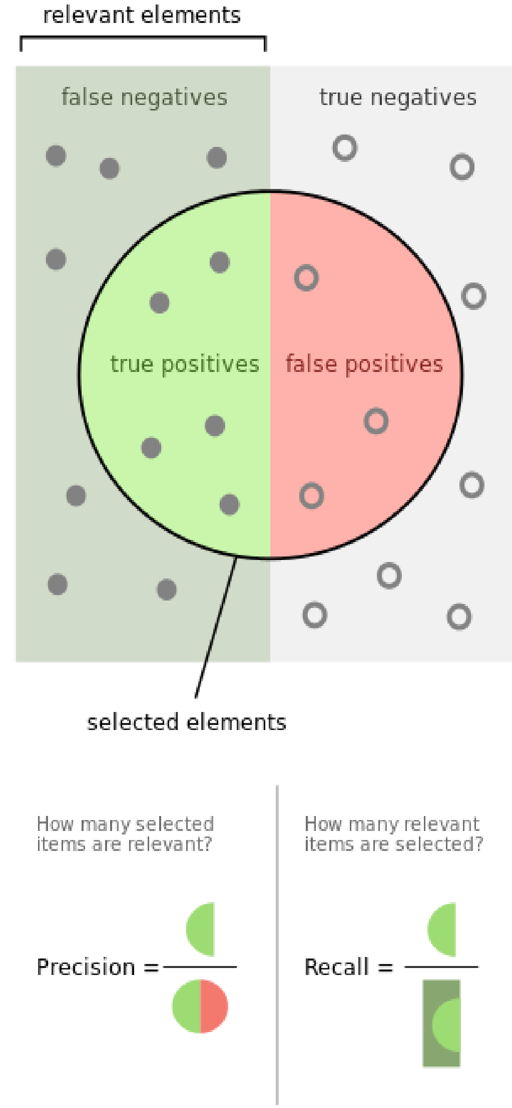
$$\frac{TP}{TP+FN}$$

- Precision

$$\frac{TP}{TP+FP}$$

- F-score : F-Score is the weighted average of Precision and Recall.

$$\frac{2*precision*recall}{precision+recall}$$



https://en.wikipedia.org/wiki/Precision_and_recall

Dataset quality

- Having a representative dataset is crucial for machine learning methods.
  - Recall the bad practice for data collection
- It is not possible to train the models on the end points
  - We cannot collect representative data there!
- The training is done on the cloud

Summary

- We motivated the need for mobile malware detection
- We discussed mobile malware specific challenges
  - Low-powered devices, app isolation, …
- Mobile malware risks were reviewed
  - System damage, economic risks, privacy risk, …
- We reviewed the security model of iOS and Android
  - We discussed the differences between iOS and Android vetting processes
- We have reviewed different techniques for mobile malware detection
  - Static, dynamic, hybrid
- Obfuscation techniques were reviewed

Summary

- The role of machine learning in malware detection
- Different learning types:
  - Supervised
    - Classification
      - Binary classification vs anomaly detection
    - Regression
  - Unsupervised
    - Clustering

Summary

- ML-based Malware Detection procedure
  - Collecting training data
  - Extracting features from training data
  - Training the model
  - Validating the model
- Machine Learning-based Malware Detection Challenges
  - Under- and Overfitting
  - Imbalanced datasets
  - Performance evaluation measures
  - Dataset quality