Automated input type prediction in android user interfaces for dynamic vulnerability analysis

by

Md Shamsul Arifin

APPROVED BY SUPERVISORY COMMITTEE:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Latifur Khan, Chair

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Zhiqiang Lin

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Bhavani Thuraisingham

Copyright 2015

Md Shamsul Arifin

All Rights Reserved

Automated input type prediction in android user interfaces for dynamic vulnerability analysis

by

MD SHAMSUL ARIFIN, BS

THESIS

Presented to the Faculty of

The University of Texas at Dallas

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN

COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT DALLAS

December 2015

Acknowledgments

First of all I would like to thank my supervisor and advisor Dr. Latifur Khan for his comprehensive guidance and support during this research. His astute supervision was imperative in completion of this thesis.

I am grateful to everyone working at Data Mining Lab at Erik Jonsson School of Engineering and Computer Science for their constant support. Specifically, I am utmost grateful to David Sounthiraraj, whose prior work has paved the way for this research. In addition, I would like to thank Swarup Chandra for helping me understand various aspects of this research.

Finally, I would like to thank and acknowledge the support and inspiration of my parents who have supported me in this complete ordeal and every aspect of life.

December 2015

Automated input type prediction in android user interfaces for dynamic vulnerability analysis

Publication No. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Md Shamsul Arifin, MS

The University of Texas at Dallas, 2015

Abstract

Supervising Professor: Latifur Khan

Recent research showed a significant number of apps for Android smartphones have broken implementation of SSL, the de facto standard for secure communication over the internet. Due to the large number of available apps, manual analysis is infeasible and automated system is required. Automated analysis of apps requires automated interaction with the user interfaces. Automated interaction requires providing appropriate inputs in the input fields. This is hard due to the absence of annotations available in the user interface. In this thesis we develop a novel machine learning approach where we use supervised learning to construct a model which can predict the input type based on the textual content available in the user interface. The system attains an accuracy of 76.26% during 10 fold cross validation performed on 21601 input fields.

Table of Contents

acknowledgements iv

ABSTRACT v

LIST OF FIGURES vii

LIST OF TABLES ix

chapter 1 Introduction 1

CHAPTER 2 BACKGROUND 5

CHAPTER 3 MACHINE LEARNING BASED INPUT PREDICTION 24

CHAPTER 4 RESULTS 37

CHAPTER 5 FUTURE WORK 47

CHAPTER 6 CONCLUSION 48

REFERENCES 49

VITA

List of Figures

1.1 Smartphone OS market share 2

2.1 Man in the middle attack scenario 9

2.2 Android system architecture 11

2.3 SMV-Hunter system overview 16

2.4 Vulnerable entry point identification algorithm 17

2.5 Smart input generation algorithm 19

2.6 Sample decompiled android code displaying the type of an input 20

3.1 Sample decision tree to determine loan approval 28

3.2 Semi heuristic neighbor search algorithm 30

3.3 Android UI XML structure 31

3.4 Corpus construction 33

3.5 Sample from constructed corpus 34

3.6 Sample from stop words list 35

3.7 Input types 36

4.1 Data distribution for simple input types 38

4.2 Accuracy on small dataset 39

4.3 Data distribution on large dataset 40

4.4 Accuracy on different models on large dataset 41

4.5 Performance of RandomForest for varying number of decision trees 46

List of TABLES

4.1 Confusion matrix for RandomForest algorithm 43

4.2 Legend for confusion matrix 43

4.3 Precision, Recall and F-measure for RandomForest algorithm 45

# 

## Introduction

Smartphones has become an integral part of our daily life. One of the most significant features of smartphones is the ability to run third party apps, and the large number of them. These apps cover a wide range of categories from simple communication to games and entertainment to financial services. These apps often contain a server-side component, a web server running r internet and communicating with the app for additional functionality. Apps often transmit sensitive data such as email, passwords, location to web server, making secure communication a critical factor for them.

At present more than 80% of running smartphones uses Android Operating System [1]. The open source operating system by Google has been preferred by most smartphone vendors because of its ease of modification and availability. Google has made the process of application development for android a very simple process, resulting in a wide variety and a huge number of third party apps available for the system. Google Play Store, the most used app store for android has over 1.43 million apps published and over 50 billion downloads [2]. A number of third-party app stores are also present, adding extra varieties of apps. As discussed earlier, these apps often require server-side communication over internet and uses SSL/TLS to secure the communication.

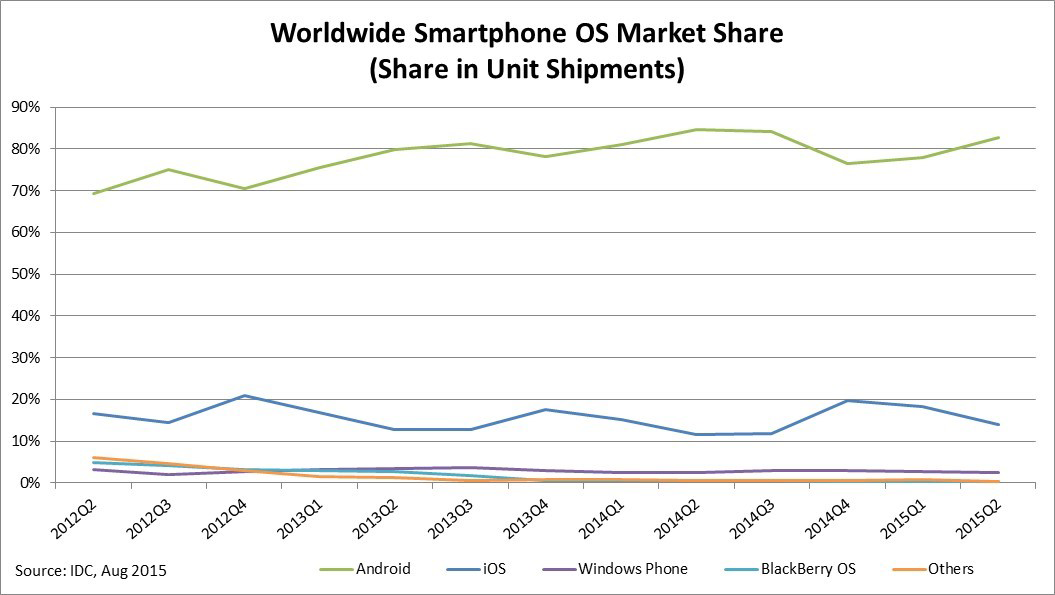


Figure1.1. Smartphone OS market share [1]

As of now, SSL/TLS is the de facto standard for secure communication over the internet. Most if not all sensitive information is communicated over the internet using SSL/TLS. SSL/TLS was implemented in order to provide end to end security in a communication channel. Public key encryption is used to secure the initial authentication phase. A trusted third party Certificate Authority is contacted to validate the authenticity of the server. The protocol specifications are well defined and considered to be secure for all practical purposes.

Initially SSL was only implemented by web browsers and email clients. They were checked for vulnerabilities on a regular basis and required modifications were performed to address them. Due to the arrival of android, a lot of new implementations came into being in order to integrate with the apps. Certain parts of the specifications are not implemented properly or not implemented at all in these implementations. This makes way for the potential eavesdropper to access the data being transmitted.

Recent research has shown that a significant number of apps available in the Google play store use poor implementations of SSL/TLS, often resulting in a compromise of privacy sensitive user data. To address this problem, a large number of apps were analyzed in order to find SSL/TLS vulnerability. However, most tools developed to analyze SSL vulnerabilities required manual analysis, making them hard to scale for a large number of apps. SMV-Hunter is a tool to check android apps for SSL/TLS vulnerabilities which requires no manual analysis, making it easily scalable for analyzing a large number of apps. [8]

Initially apps are decompiled and analyzed to identify potential vulnerable code. A method call graph is generated to identify entry points to the vulnerable code. The user interface of the app is analyzed and inputs are generated for input fields of the user interface. A man-in-the-middle attack environment is configured using a proxy which listens to the communication between the emulator and the internet. Later the app is run on an android emulator. The system interacts with the app, providing inputs as necessary in order to execute vulnerable code. The time of the execution of vulnerable code is monitored and the data from proxy is obtained from that timeframe to detect SSL Vulnerability. [8]

Providing valid inputs to the user interfaces in the apps is essential to be able to interact with the app. Even after analyzing the decompiled source of an app, it is hard to automatically generate a proper user input because of lack of annotation provided by the developer. Hence we propose a natural language processing and machine learning based solution to the problem. In our solution, we take a number of decompiled apps and extract relevant textual features from its input fields. Features from the neighboring text field are also extracted as it is often used to provide input information to the user. A supervised learning model is constructed using these features as input. During analysis, we extract the same features from the input field and feed it to the model to predict the type of input required. Several models are constructed using various supervised learning algorithm and the best performing one was chosen. In our analysis, the model generated using RandomForest; an ensemble learning technique using several decision trees had the best performance, achieving 75% accuracy on performing 10.fold cross validation over 5600 input fields extracted from 2000 apps.

# 

## Background

**2.1 Secure Socket Layer/Transport Layer Security**

Transport Layer Security (TLS), formerly known as Secure Socket Layer (SSL), are a set of cryptographic protocols designed to ensure privacy and data integrity during a communication over a network. Most major web service providers use TLS to secure communication between the client and the server. Symmetric-key cryptography is used for communication and the keys are generated uniquely for each session. The keys are generated by the server and sent over to the client using a handshake protocol. In a TLS handshake, the client and server initially agrees upon a cipher and hash function. The server sends a certificate message, containing the name of the server, the trusted certificate authority and a public encryption key. The client then contacts the certificate authority to validate the authenticity of the certificate. Upon validation, communication is initiated between the client and server. [3]

There are four protocol layers of SSL. The record layer formats Alert, ChangeCipherSpec, Handshake and application messages. Each message is provided a header and a hash generated from a Message Authentication Code. The ChangeCipherSpec layer signals the start of secure communication. The Alert protocol is responsible for sending error, problem or warning messages. It has two fields, severity level and alert description. The handshake protocol consists of a set of messages passed between the client and server, initiating a secure connection. The key components of SSL/TLS are as follows:

2.1.1 Symmetric Key Cryptography

Symmetric-key cryptography refers to a set of cryptographic algorithms that uses the same cryptographic keys for both encryption and decryption. In practice the keys represent a shared secret between the client and the server. The algorithms do not guarantee the integrity of the message. As a result a Message Authentication Code is added to the ciphertext. In SSL/TLS, the negotiation of the shared secret is considered secure and cannot be modified by an attacker who is placed in the middle of the connection. [4]

2.1.2 Asymmetric Key Cryptography (Public Key Cryptography)

Public key cryptography refers to cryptographic algorithms where the keys for encryption and decryption are different. In these schemes, a sender wishing to communicate obtains the public key of the recipient and encrypts the message. The recipient, upon receiving the message decrypts the message using its own private key. The public key can be shared without compromising security. The algorithms are approximations and as of the day the problem is considered unsolved. The key idea of these approximation algorithms is that even if a private key can be generated from a public key, it will take a very long time to compute it, making the generation impractical in practice. The algorithms are computationally complex and hence used for short messages e.g.: transfer of a shared secret. [3][4]

2.1.3 Certificate Authority

To ensure the security of SSL/TLS, a trusted third party is involved in the communication that validates the certificate message and the public key sent by the server. The certificate authority (CA) issues digital certificates and validates them upon receiving. The CA typically uses domain validation which involves sending an email containing an authentication token where the email addresses in known to be managed by the legitimate owner of the domain. [5]

2.1.4 TLS Handshaking

In a TLS handshake, the server and client agrees on a protocol version, selects encryption and decryption algorithms and generate shared secrets. Often both authenticate each other, but the step is mentioned optional in the protocol specification. The server sends a certificate message to validate its authenticity. Public key cryptography is used during this communication to ensure security. [4]

TLS Handshake is initiated by the client. The first message sent is ClientHello, where the client specifies a set of options. They are the SSL Version, the CipherSuits used by the client and CompressionMetheds used by the client. It also consist a 32 byte random number that helps the client in establishing a secure connection. In response, the server sends ServerHello. In ServerHello, the server specifies the SSL Version, CipherSuite to be used and the Compression Method. Later, the server initiates ServerKeyExchange, where it sends its public key and a digital certificate. The client validates the certificate through a trusted third party known as certificate authority. Then a ServerHelloDone message is sent by the server to indicate its part of handshake is done. The client sends a ClientKeyExchange message specifying the shared secret that both client and server will use to communicate with each other. The message is encrypted with the server’s public key in order to subvert man-in-the-middle attacks; as the middle man requires to know the server’s private key in order to decrypt the message. Then both client and server send a ChangeCipherSpec indicating the communication is entering a secure state. [9]

2.1.5 Man-in-the-middle attack

A man-in-the-middle attack (MITM) refers to an attack where the attacker secretly places himself in the midpoint of a communication and observes and alters the messages that are being sent. This kind of attack can only be successful if an attacker can impersonate each endpoint to

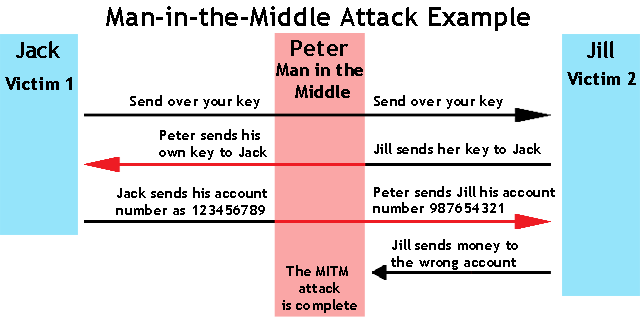


Figure 2.1. Man in the middle attack scenario [11]

the satisfaction of the legitimate other end. Most, if not all cryptographic protocols contains some form of authentication specifically to prevent this sort of attacks. SSL/TLS performs this authentication using a CA which is trusted by both ends [6].

**2.2 Structure of an Android application**

Android Operating System is a multi-user Linux platform. Each app is a different user, and the OS assigns appropriate permissions for the files in order for the app to access them. Apps are written in Java, each app has its own virtual machine, is order to run in isolation from other apps. Internally, every app runs in its own Linux process, the startup and shutdown of these processes are managed by the operating system [7].

The following are the key components of an android application.

2.2.1 Activity

Android activities are the most typical entry point of an android app. It usually contains a single user interface. An app can have multiple activities; each can work independent from the others in order to provide a cohesive user experience. Android maintains an activity stack for each application. When a new activity starts, it is pushed into the stack. If the user is done and presses the ‘back’ button, the activity is popped from the stack and destroyed and the older activity comes into focus. [7].

2.2.2 Service

In brief, a Service is an activity without a user interface. A service typically runs in the background. It can perform long running operations in the background. A service requires an activity to start, and can run in the background even after the activity is closed. On the other hand, activities can bind to service in order to communicate with them. A service usually kills itself when its work is done, but if memory is low and the OS requires resources for the activity that has user focus [7].

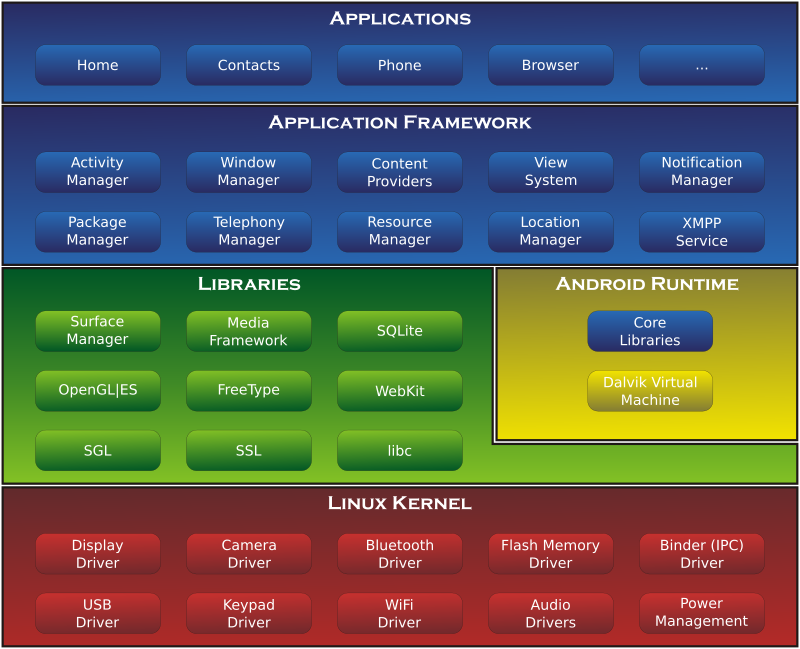


Figure 2.2. Android system architecture [10]

2.2.3 Broadcast receiver

A broadcast receiver works as an event listener for system events. An app requires registering broadcast listeners if it wants notification of system events such as SMS receive, Wi.Fi connected etc. [7].

2.2.4 Manifest

The manifest is an xml file, containing basic information regarding the app. It incorporates the minimum API level required for the app, permissions the application require to operate, such as access to hardware or private user data as location, contact list or SMS. The manifest also contains details about each Activity present in the app, which are entry points to an application. It also contains information about Services and Broadcast Listeners [7].

2.2.5 Source code

The source code of a typical android app is developed using android sdk. Android sdk is based on java, though native support is provided through Java Native Interface wrapper. Certain built.in classes are provided in the sdk which a developer extends during development. For instance, every app must contain a subclass of Activity class, which determines the entry point of the application. The name of the activity is provided in the manifest. Typically, each window in the application UI has its own activity. Background daemons are developed by extending Service class, and System event listeners are developed by extending the Receiver class. [7]

2.2.6 UI Resources

Though the user interface can be generated dynamically in the source, android system provides a well-designed xml based system for developers to design the applications user interface so that they can be maintained independently. In this approach, every UI component has its own xml file which defines their size, orientation and position. The developer provides an id for each component which is required during runtime, and can retrieve it in the code using the id. The UI constant assets (static text, image) are also given an id. The id values are generated during runtime and a hexadecimal constant value is generated in compile time for each id by android system. This approach helps provide better localization support and support for providing alternate resource with varying device configurations [7].

**2.3 SSL/TLS in Android**

Most if not all android apps require communication over internet. The communication needs to be secure as the data sent are often sensitive private information. As a result, the apps use SSL/TLS to ensure secure communication of such information. As explained earlier, SSL/TLS requires validating server information using a Certificate Authority. Failure in such validations makes the system vulnerable to a Man-in-the-middle (MITM) attack, where an attacker capable of intercepting and altering the communication can pose as the server and retrieve sensitive data.

The rules to establish a secure SSL/TLS connection is described in RFC 2818, 2246 and 3280. Clients requiring a connection need to check if the certificate chain and the hostname are valid. A certificate chain is valid if none of the certificate in the chain has expired, the root certificate is verified from a trusted certificate authority which is present in the client’s keystore, and each of the certificates in the chain was validated by the CA just after they were added in the chain. A hostname is valid if the name in the certificate matches the domain name of the server. [8]

Android OS provides SSL certificate validation and hostname verification as a built.in support. However, developers are allowed to incorporate their own implementations using X509TrustManager and HostNameVerifier interfaces. A developer might choose to use a custom implementation as there were errors in certificate validation in the earlier versions of Android. Moreover, if the certificate's root CA is not present in the Android keystore, a secure connection was impossible to establish. Additionally, developers use self-signed certificates. Some third party libraries also provide SSL certificate validation. Such custom implementations often have errors, some intentionally accept all self-signed certificates, some accepts all without any validation. [8]

Recent research has demonstrated the SSL/TLS vulnerabilities in Android system. Georgiev et. al. established that the SSL certificate validation is dysfunctional in many popular security critical apps and libraries. They used both white and black box techniques to discover vulnerabilities in SSL validation logic. Egele et al. demonstrated that the primitive cryptographic functionalities used by apps like block ciphers or Message Authentication Code(MAC) are not properly used. In their findings, they analyzed 11748 apps and found out that 88% of them make at least one mistake. Fahl et al. statically analyzed 13500 apps in order to find potential vulnerabilities. From the potentially vulnerable apps, they manually inspected to find true vulnerable apps. They also conduct a user survey and found that 50% of the users taking the survey failed to judge whether their browser session was protected by SSL/TLS or not. In another study they pointed out that that the root cause of the problem is not simply careless developers; there are flaws in SSL implementation paradigm as well. They suggested using a centralized SSL implementation instead of third party implementations for all apps. [12] [13] [14] [15]

**2.4 SMV-Hunter**

SMV-hunter was designed to enable automatic, large-scale identification of SSL Man-in-the-middle vulnerability (SMV). Observation of earlier research has established that a static analysis is not powerful enough to detect SMV. Static analysis is not sufficient to determine the flow of information, data dependency and user interaction, which results in inaccuracy. SMV-hunter incorporates a dynamic analysis component, in which a target app is monitored during its execution, detecting the triggering of vulnerable behavior by performing a simulation of user interaction expected by the app. [8]

A dynamic analysis would go through all possible user interface present in the app. This makes the approach slow. Moreover, the apps often validate user inputs before accepting it, which makes it hard for a dynamic mechanism to provide valid user input. To address this problem, a static analysis component is integrated in SMV-Hunter. The static analysis inspects the app, prunes the UI search space and generates valid user input. [8]

The major design goal of SMV-hunter was to establish a tool which is capable of analyzing a large number of apps within considerable time. As a result, the pruning of the UI search space is critical. Moreover, as the approach is entirely automated, there should be no false positives in SMV detection. [8]



Figure 2.3. SMV-Hunter system overview [8]

2.4.1 Static Analysis

The static analysis component decompiles an android app, looks for vulnerable code entry points and attempts to generate valid user inputs. Apktool was used for decompilation process. It decompiles the resources accordingly, and the code is disassembled into smali, a human readable format. Disassembly was chosen over decompilation as it is a much faster process. [8]

In order to detect vulnerable entry points, it looks for implementation of X509TrustManager and HostNameVerifier interfaces. A few common pattern has been identified which results in SMV. X509TrustManager can be implemented as no-op which asserts all certificates as valid. It can be implemented so that it looks for a single self-signed certificate in the certificate chain, checks for its expiry but does not checks its signature or notifies the user that a self-signed certificate is being used; or it can be implemented so that it just checks whether the certificate is expired or not. HostNameVerifier is most commonly implemented as a no-op. [8]



Figure 2.4. Vulnerable entry point identification algorithm [8]

2.4.2 Method Call Graph

An android app can have various entry points through its activities and services. Nevertheless, some entry points does not involve in establishing a HTTPS connection, making them redundant for our analysis. Smv-hunter detects vulnerable entry points by constructing a method call graph(MCG). In MCG, the UI is represented as a directed graph, whose nodes corresponds to windows and edges correspond to actions/UI events. It finds the methods using the X509TrustManager or HostNameVerifier interfaces and traces them back to their entry points. It is possible that a vulnerable method is never called by the app itself, rather than called by Android OS. In such cases a good assumption is that an object was constructed using the methods class and was passed on to the OS. As a result, MCG looks for instantiation of the class, and continues the search. [8]

2.4.3 Smart Input Generation

Apps often require user input, and validation is performed on the input upon receiving. An app cannot be interacted with if a valid input is not provided during dynamic analysis. SMV hunter looks in the decompiled android code to check for potential type casts to the input. Furthermore, it looks at the input type annotations provided in the xml layout of the input and tries to provide a typical input for the field. This approach is insufficient as most inputs are further validated after type casting and developers often do not provide an annotation to the field in the xml layout. [8]



Figure 2.5. Smart Input generation algorithm [8]

In our analysis of 5300 android apps, it has been found that among 14,500 input fields, only 5600 contained input type annotation.



Figure 2.6. Sample decompiled android code displaying type information of an input [8]

To address this problem, a machine learning based mechanism is used to predict the desired input type. In this scheme, a model is trained with features of annotated input fields. When a field with no annotation is found, the features are fed into the model to predict its potential input type.

2.4.4 Dynamic Analysis

In dynamic analysis each app is run in an android emulator by a device management component, which in turn triggers UI automation component and collects logging information. The UI automation component interacts with the app and a proxy monitors network traffic in order to launch a MITM attack. [8]

2.4.5 Device Management

The device management component runs android emulators, installs apps and runs them. It dynamically manages a set of emulators, increasing its number or decreasing it as per the available resource. It monitors the internal states of each emulator. It triggers the UI automation component to interact with the app. It logs information from android OS and the app and monitors network traffic. It also handles emulator crashes and other errors and takes corrective actions. [8]

The component manages two sets of emulators: the running pool and the free pool. It registers a DeviceChangeListener callback with the ADB. When an emulator starts, it is added to both the sets. If the emulator crashes or enters an offline state, it removes it from both sets and starts a new emulator in place of it. When the app scheduler requests an emulator, it is removed from the free set and added back when the app is analyzed. [8]

The component contains an app scheduling thread in order to manage the list of apps being tested. It iterates over each app, retrieves an emulator to run it in, install the app in the emulator and runs UI automation on each vulnerable code section. Then the emulator is returned to the previously mentioned sets of emulators. It also handles errors like emulator crash or failed installation. [8]

2.4.6 UI Automation

UI automation attempts to emulate user interaction with the app, executing code paths found in static analysis to lead to the execution of vulnerable code. It has three goals: understanding the user interface, providing appropriate inputs and observe and manage the state of the app. [8]

Initially the module decomposes the UI into its component elements, determining its coordinates in the screen and the type of input required. It uses ViewServer, an internal component of Android app tooling framework to query the WindowManager which is responsible for displaying the UI. After the correct input is determined, WindowManager is used to emulate the appropriate input event. The app state is monitored from ViewServer, and the component waits between two state changes so that the event can be processed. [8]

2.4.7 Man In the Middle Proxy

For executing SSL MITM attack, all HTTPS network traffic from the emulator to the internet needs to be monitored. In order to perform that, a proxy is set in the workstation. The emulator connects to the internet through that proxy, and proxy traffic is monitored and logged. [8]

## Machine Learning Based input prediction

For SMV-Hunter to accurately detect vulnerabilities, it needs to interact with the User interface of the android app in order to execute vulnerable code. In order to achieve that, proper inputs need to be provided in the input fields that appear in the user interface.

During static analysis step, decompiled code is searched for potential type casts in order to determine the type of input that needs to be provided. Also, xml layouts are searched for type annotation. However, type casts are often insufficient to provide a valid input and as discussed earlier, more than 60% of the input fields do not have any type annotation at all. We propose a supervised learning based system in order to predict the type annotation. In this scheme, we train a model with type annotated input fields using the textual content available in them as features.

Not much work has been done in doing predictions based on Android XML resources. Nan et al. has used Android UI XML in order to identify privacy sensitive input fields. The system can detect semantic information within XML data and identify privacy sensitive information. The textual contents from inputs fields were extracted and a classifier was trained from them. The classifier performed well as it was a binary classification. As we are interested in providing valid inputs to the input fields, the classification domain is expanded and correct prediction are required to ensure automated interaction with the app. [23]

**3.1 Key Concepts**

3.1.1 Supervised Machine Learning

Machine learning refers to a set of algorithms designed to learn from data and make predictions based on the learning. The algorithms are closely related to computational statistics. Most if not all machine learning algorithm comprises of three components: Representation, Evaluation and Optimization. The representation component processes the data and constructs a model based on it. The evaluation component refers to designing of an evaluation function that can evaluate the performance of the data. The data is represented as a vector where each entry is denoted as a feature. The optimization component is a mechanism of altering the model based on its evaluation. Very simply, a machine learning algorithm works as follows: A base model is constructed which can represent the data. Then the model is iteratively evaluated and optimized until the desired performance is attained. The resulting model then can be said to be able to make predictions in accordance to the defined performance constraint. The model is the considered as a black box and unlabeled data is fed as input to get the predicted result. [16]

Supervised learning refers to a subset of machine learning algorithms where the model is constructed based on labeled data in order to predict the label from unlabeled data. Mathematically, the algorithm(s) tries to approximate a function between the data and its label. This approximation is called a hypothesis. In supervised learning, the data is divided into two sets: training and testing. The model is constructed from training set and later evaluated on the test set. In practice, cross validation is applied. In cross validation, data is divided into N equal size folds. Testing is performed using each fold where the rest of the data is used for training. [17]

3.1.2 Bias-variance trade off

Bias refers to the errors made by the learned model and variance denotes the error from sensitivity to small change in the training set. Bias-variance tradeoff is a significant problem in supervised learning. In an ideal case, we want a model that represents the regularity in the training data as well as a model that is generalized so that it provides a satisfactory performance over unseen test data. However, it is almost impossible to attain both at the same time. If variance is maximized, the model often represents a large amount of noise present in the dataset which it turn leads to poor generalization. This situation is called overfitting. If bias is maximized, it leads to very simple models, failing to capture significant regularities present in the data. This is also known as underfitting. Ensemble methods tend to work well to resolve bias-variance trade off. [18]

3.1.3 Ensemble Method

An ensemble method uses multiple supervised machine learning algorithms, constructing multiple models from them in order to achieve better performance. As an ensemble method can be trained, it represents a single hypothesis; though the hypothesis space is not contained within the hypothesis space of the models it was built. Empirically, ensemble methods perform better when the data is significantly diverse. [19]

3.1.4 Decision Tree

Decision Tree is a machine learning algorithm which uses a decision tree as a model mapping examples to its labels. In the tree, leaves represent labels and branches represent logical conjunctions of features leading to that label. In other words, each node in the tree represents a test on a feature and each branch represents the outcome of the test. [20]

One of the key property of a decision tree is that the model is observable and can be easily interpreted (white box), compared to other learning algorithms where the model is hard to explain. [20]

Decision tree algorithms usually work on a top down approach, choosing a variable at each step which can best split the set of data. Different implementation uses different metrics to choose this variable. The Gini impurity measures how often a randomly chosen element from the data subset would be incorrectly labeled if it was labeled according to the distribution of the labels of the data subset. Information gain metric is measured based on the concept of entropy found in

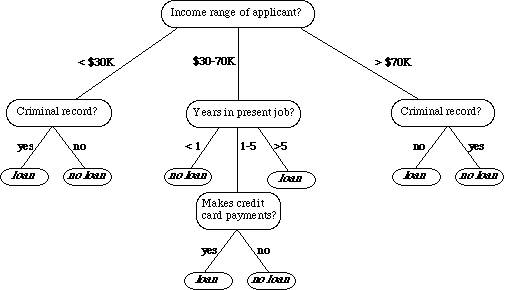


Figure 3.1. Sample decision tree to determine loan approval [21]

information theory. Variance reduction computes the total reduction in variance of the target variable sue to the split. [20]

3.1.5 Random Forest Algorithm

Random Forest is an ensemble learning algorithm based on decision trees. It constructs multiple decision tree models and combines their results to generate the final class label. The trees are constructed using random selection of features which is also known as random subspace method. Random forests tend to correct for the overfitting in a typical decision tree. [22]

3.1.6 Bag of Words

The Bag-of-words model is a simple representation approach in natural language processing. In this model, a data sample is represented as a vector of the words that are present in them. The model does not consider the word order or grammar. Each entry in the vector represents a single word, the value represents the number of times the word has occurred in the data sample.

**3.2: Input Prediction methodology**

The methodology of input prediction comprises of the following steps:

3.2.1 Neighbor Retrieval

Most of the times an input field has a neighbor static text field beside it which helps the user to determine what information needs to be provided. We use the properties of the text field as features to the system alongside the properties of the input field. However, finding the proper text field is a hard problem as the layout XMLs are highly convoluted and often dynamically generated by the program. Moreover, developers often design their own layout structures containing custom properties in their UI.

To address this problem, we use a semi-heuristic neighbor search. If the parent layout of an input field is a known layout, we use the properties of the layout to determine the neighbor element. If the parent layout is a custom one, we simply use the nearest child element from our input element as neighbor. If the neighbor element is itself a layout, we recursively parse through it, using the previous scheme to find the nearest neighbor.

**Algorithm:** find\_neighbor: Finding neighbor element from UI XML

**input:** **layout**: XML Layout,

**field**: Input Field

**output:** Neighbor Text Field

**begin**

**if** type(layout) is known **then**

neighbor = find\_neighbor\_using\_properties(layout, field)

**if** neighbor is Layout **then**

position = find\_position\_relative\_to\_field(neighbor, field)

**while** (neighbor is layout) **do**

neighbor = find\_neighbor\_from\_position(neighbor, position)

**else**:

neighbor = get\_nearest\_neighbor(layout, field)

**if** neighbor is Layout **then**

position = find\_position\_relative\_to\_field(neighbor, field)

**while** neighbor is layout **do**

neighbor = find\_neighbor\_from\_position(neighbor, position)

output(neighbor)

Figure 3.2. Semi heuristic neighbor search algorithm

3.2.2 Relevant properties

The input field contains a lot of properties. Most of them contain information about its size, alignment and orientation in the UI. We discard such properties as they are irrelevant for our purpose and select the properties which are relevant and can provide information regarding the type of the input. As discussed earlier, every UI element in the xml contains an id field which is set by the developer. Most if not all developers provide meaningful names in this field as it is considered among the best practices for software development. Hence, we use the id field as a feature as it is most likely to contain some information about the type of the input field.



Figure 3.3. Android UI XML structure [23]

3.2.3 Corpus construction

Though the generic textual content present in the UI layout are structured, the id field often contains unstructured text and requires special techniques to parse and tokenize. As a result the typical word corpus or stop word corpus will not work very well for our problem. For instance, as we are considering variable names given by the developer, it can be 'user\_name', 'usrName', 'uName' for a input field retrieving the name of the user.

Keeping these factors in mind, we generate a corpus from the dataset. All text data present in the dataset are tokenized. We use '\_' as a token separator alongside typical token separators to extract tokens from variable names. Moreover if a token is camel cased (userName), we separate them accordingly and generate the corpus. We add some extra tokens as stop words alongside a typical stop words list by observing the data (e.g.: edit, view). We also discard some typical stop word from the original list as we found they are relevant for our purpose (e.g.: state). Tokens less than a length of 2 are discarded as they cannot possibly contain any useful information. Alongside stop words filtering, we discard the very least frequently used tokens from the corpus as they would tend to lead the model to be overfitting.

edit\_text\_userName

edit, text, user, name

Figure 3.4. Corpus Construction

3.2.4 Feature vector generation

To generate feature matrix, we use a simple 'bag of words' approach. We search for each token in our corpus in the textual properties of the input field. The weight of that entry is the number of times that token is found in the fields. There are usually a very small number of tokens available as features for each input compared to the length of the corpus. As a result, the feature matrices are mostly sparse.

acc access accident acct activate activation activity added

addition addr address addressbook adjust adr adresse

aendern afterhours age agent album alert alias amet amount

amt annual ans anti antilost app application apply apps apt

aptnum associated atm att auth author automatic avr

backup badge bal balance bank banking barcode base

basics bcc besked beskeder bill biller billing billpay bio birth

birthday blank body bonus book bookmark bot bottom budget bug

business buy calc calculate calendar call caption car card

cash cate categories category cell cena cent cfm challenge

change char chat check child choice choose cicalc city claim click client

close code color column comma command comment

common company compose computer concepto confirmation

conn conta contact content contract core cost count

country coupon cpr credentials credit creditcard csf cuenta

curr currency custom customer cvv database date day

dealer debt decimal del delay delete deposit depreciation des

describe descripcion description desired destino det details developer

deviation device dfe dial digit direct dis discount

display dob dollar domain drive drop dropbox due duration

editname editpage eff ein einrichten ema email emi emne

employer empty enterpass entity entry esignature etf

etiquetaeur euro event exchange exist exp expense expiration

expiry export facebook facebookpost family favorite feature fee feed

feedback fees file filename fill filler firstname fixed flag

flight fname folder food foot forget forgot format free friend fro fsn

fullname fund hand happy height help highlight hist history hnt

home homebank host hours http hyphen iban identifier identity

image imap import importe income indice information ing init

initial insert instruction insurance interest interestpay interval intro

investment invite invoice ipo irs issue iva job jobtitle jump key

keyword kod kontonavn kontonummer kundenr kuponrente lag

lastname ldescr leave ledgr left leg length letter level license light

limit link lit lname loan loancalc loc local location lock

login loss lost lots loyalty mac macd mail mailbox manag

manage map master max meddelelse meeting memo

mess message meter mid middle miles min mind minimum minus

miss mname mob mobil mobile mobiletopup model money month monthly

monthlypay mot mov msg naiyo name navn navnadresse near

newinvtx newtx nfc nick nickname nom nombre nop note notes notification nps num number

numer obligatoire offer onetime online onpassword onto onuser operation opt optin optional options

org orig otp outgoing own pag pago pal papirsoeg par para

pass passcode passe passwd password

passwort path pay paybill payee payload payment payout

Figure 3.5. Sample from the constructed corpus

such sure take taken than that the their them then there therefore these they thing things think thinks this those though

three through thus to today together too took toward

turn turned turning turns two under until upon use used uses

very want wanted wanting wants was way ways we well

wells went were what when where whether which while who

whole whose why will with within without work worked working

works would yet you young younger youngest your yours txt

text view true false field feature computer

Figure 3.6. Sample from stop words list [24]

3.2.5 Complex input type annotation

The input type field is the class label of our problem. For this field, there are 32 values which can be 'Or' ed with each other to generate a value for the field. As a result there are 2^32 possible values for the class variable. However, some of the values are improbable and through observation of data we found that most of the values in the possible values set never occur in our dataset. Nevertheless we develop our system in way so that it can be trained using any value that is possible.

text textCapCharacters textCapWords

textCapSentences textAutoCorrect textAutoComplete

textMultiLine textImeMultiLine textNoSuggestions

textUri textEmailAddress textEmailSubject

textShortMessage textLongMessage textPersonName

textPostalAddress textPassword textVisiblePassword

textWebEditText textFilter textPhonetic

textWebEmailAddress textWebPassword number

numberSigned numberDecimal numberPassword

phone datetime date

time none

Figure 3.7. Input types [25]

3.2.6 Model Training

After the feature set was extracted, we get a dataset of 22000 input fields, each containing 782 features. We apply several learning algorithms and generate trained models in order to find the model with best performance. To avoid overfitting, we use 10 fold cross validation. Among several models that had been trained, the model generated using RandomForest algorithm had the best accuracy over the test set (76%).

## Results

Initially, we chose a smaller dataset to generate our corpus and stop words list. The dataset comprised of 2510 apps and we found 5485 input fields with valid annotations from a total of 7010 input fields. The corpus was constructed from all 7010 fields as we do not need type annotation for that purpose.

After the corpus was constructed, we generate a supervised learning model from the input fields containing valid annotations. To handle complex input fields, experimentally we decomposed them into simple input types. As the types were 'OR' ed with each other, for each complex input type we generated simple input types with each of the type present using the same features that were present in the complex input type. We trained several algorithms and tested them with both complex and simple input types to measure their performance.

After conducting the experiment, we trained our model with a larger dataset, consisting of 12734 apps. We found 48745 input fields, amongst those fields 21601 fields contained valid input annotations. We generate feature vectors using the same corpus and stop word list we generated from the smaller dataset. Various models were trained using algorithms that performed relatively better.

Figure 4.1. Data distribution for simple input types

From the figure, we can observe that the types number and numberDecimal are the most frequently occurring input types. The types text, textPassword and numberPassword occur with more or less similar frequency. The rest of the types found in the dataset had similar frequency except textUri, which is the rarest type found in the dataset. We found 13 different types of annotations in the dataset, the rest of the types were not found. We discarded none type as it is provided by developers trying to ignore type annotation and including it will only result in poor classification accuracy.

After careful observation of the data, we found that complex input types are rare and input type is provided mostly as a single value. The amount of data with a complex input type is very small compared to the entire data. Decomposing such data into simple input types only increases the confusion among types in the model, which is reflected in the results.

Figure 4.2. Accuracy on small dataset

From the results, it is clearly observable that our experimental setup of decomposing complex types into simple types did not work well. It only increased confusion among the classes, resulting in poor performance, as we can clearly see that the accuracy is less in simple input types for all the algorithms that were used to generate a model. As a result, we proceed forward considering only complex cases as class labels.

Figure 4.3. Data distribution on large dataset

The larger dataset had more class types than our smaller dataset. As a result, the

classification performance degraded a bit. There were significant numbers of entries with complex input types. As a result the classification problem became harder.

Figure 4.4. Accuracy for different models on large dataset

From the results found from constructing model for large dataset, we can see that the accuracy decreased a bit compared to the small dataset. This is understandable as a larger dataset often contains more outliers and variations. Comparative performance among the algorithms remained somewhat the same, as we can see RandomForest performed better than other algorithms with 76.26% accuracy. RandomTree performed slightly better than SVM:NormalizedPolykernel as the accuracy for RandomTree (74.52%) is better than SVM (72.72%).

Confusion matrix is defined as a square matrix where the rows and columns are the class types of a classifier. Each entry (i,j) denotes the number samples of class i classified as class j. The diagonal of the confusion matrix hence shows the number of correctly classified samples, whereas the other columns represent the number of misclassified samples and the type of the misclassified class.

Another performance measure for classifier is based on precision and recall. The measure is known as F-measure. Precision is defined as the ratio of samples classified correctly and the total number of samples classified whereas Recall is measured as the ratio of number of samples classified correctly and the total number of samples present in the dataset. F-measure is computed from the harmonic mean of precision and recall.

F-measure is defined as:

Table 4.1. Confusion matrix for RandomForest algorithm

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **a** | **b** | **c** | **d** | **e** | **f** | **g** | **h** | **i** | **j** | **k** | **l** | **m** | **n** | **o** | **p** | **q** | **r** | **s** | **t** | **u** | **v** | **w** | **x** | **y** |
| **a** | 125 | 0 | 0 | 12 | 0 | 6 | 0 | 2 | 0 | 0 | 0 | 17 | 2 | 1 | 14 | 0 | 3 | 4 | 4 | 1 | 0 | 0 | 0 | 1 | 1 |
| **b** | 0 | 64 | 0 | 1 | 0 | 18 | 0 | 0 | 4 | 0 | 1 | 7 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 2 | 0 | 2 | 0 | 3 |
| **c** | 0 | 0 | 361 | 8 | 0 | 1 | 41 | 5 | 1 | 0 | 1 | 34 | 0 | 0 | 0 | 0 | 4 | 5 | 0 | 2 | 3 | 0 | 20 | 1 | 0 |
| **d** | 6 | 1 | 6 | 2172 | 0 | 23 | 26 | 21 | 220 | 12 | 6 | 75 | 7 | 10 | 18 | 1 | 10 | 182 | 4 | 5 | 17 | 0 | 22 | 6 | 8 |
| **e** | 0 | 2 | 0 | 4 | 58 | 22 | 0 | 0 | 1 | 0 | 0 | 9 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 2 | 3 | 1 | 3 | 0 | 5 |
| **f** | 2 | 10 | 4 | 25 | 16 | 608 | 5 | 4 | 20 | 1 | 4 | 61 | 4 | 0 | 12 | 0 | 11 | 21 | 0 | 3 | 23 | 3 | 14 | 2 | 51 |
| **g** | 0 | 0 | 3 | 19 | 0 | 3 | 2448 | 8 | 4 | 1 | 0 | 13 | 3 | 0 | 1 | 0 | 5 | 14 | 0 | 1 | 2 | 0 | 27 | 0 | 2 |
| **h** | 1 | 1 | 1 | 25 | 1 | 6 | 27 | 588 | 5 | 0 | 0 | 60 | 5 | 2 | 12 | 0 | 12 | 6 | 1 | 5 | 9 | 0 | 18 | 3 | 1 |
| **i** | 0 | 1 | 0 | 233 | 0 | 22 | 2 | 1 | 1514 | 4 | 3 | 30 | 5 | 13 | 2 | 0 | 3 | 77 | 0 | 0 | 22 | 0 | 0 | 1 | 12 |
| **j** | 0 | 0 | 0 | 33 | 0 | 1 | 0 | 0 | 20 | 43 | 0 | 2 | 1 | 1 | 1 | 0 | 0 | 4 | 0 | 0 | 4 | 0 | 0 | 0 | 1 |
| **k** | 0 | 0 | 0 | 11 | 0 | 1 | 0 | 0 | 7 | 0 | 95 | 2 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| **l** | 15 | 8 | 22 | 115 | 2 | 79 | 55 | 67 | 45 | 3 | 2 | 1601 | 15 | 5 | 116 | 6 | 70 | 41 | 1 | 18 | 71 | 0 | 58 | 23 | 24 |
| **m** | 1 | 0 | 1 | 27 | 1 | 2 | 4 | 4 | 11 | 0 | 0 | 29 | 202 | 0 | 11 | 0 | 1 | 6 | 0 | 2 | 3 | 0 | 2 | 0 | 0 |
| **n** | 0 | 0 | 0 | 28 | 0 | 2 | 0 | 0 | 30 | 1 | 0 | 4 | 0 | 63 | 0 | 0 | 0 | 4 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| **o** | 6 | 1 | 1 | 19 | 1 | 12 | 2 | 18 | 3 | 0 | 1 | 92 | 9 | 0 | 873 | 6 | 40 | 7 | 3 | 4 | 31 | 0 | 8 | 1 | 2 |
| **p** | 2 | 0 | 0 | 1 | 1 | 0 | 1 | 4 | 1 | 0 | 0 | 8 | 1 | 0 | 18 | 83 | 15 | 0 | 0 | 0 | 0 | 0 | 3 | 0 | 0 |
| **q** | 1 | 1 | 0 | 16 | 0 | 9 | 7 | 18 | 4 | 0 | 5 | 108 | 1 | 0 | 75 | 15 | 552 | 7 | 0 | 4 | 36 | 1 | 55 | 2 | 1 |
| **r** | 2 | 0 | 4 | 198 | 6 | 13 | 24 | 6 | 117 | 2 | 2 | 35 | 3 | 4 | 12 | 0 | 3 | 1647 | 1 | 4 | 10 | 0 | 17 | 0 | 4 |
| **s** | 9 | 0 | 0 | 2 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 3 | 0 | 0 | 6 | 0 | 0 | 2 | 99 | 0 | 0 | 0 | 1 | 0 | 0 |
| **t** | 2 | 2 | 2 | 11 | 0 | 7 | 7 | 9 | 1 | 0 | 1 | 61 | 2 | 0 | 12 | 0 | 5 | 6 | 0 | 48 | 5 | 0 | 9 | 4 | 1 |
| **u** | 1 | 1 | 0 | 13 | 0 | 21 | 1 | 1 | 5 | 0 | 2 | 40 | 2 | 4 | 34 | 2 | 17 | 7 | 0 | 2 | 577 | 1 | 9 | 1 | 19 |
| **v** | 0 | 1 | 0 | 0 | 0 | 4 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 126 | 0 | 0 | 1 |
| **w** | 1 | 2 | 5 | 22 | 0 | 8 | 51 | 17 | 2 | 2 | 1 | 58 | 3 | 0 | 6 | 1 | 24 | 11 | 0 | 4 | 4 | 0 | 2192 | 6 | 0 |
| **x** | 2 | 0 | 2 | 17 | 2 | 4 | 9 | 1 | 3 | 1 | 0 | 32 | 0 | 0 | 1 | 0 | 1 | 5 | 0 | 0 | 2 | 0 | 5 | 210 | 0 |
| **y** | 0 | 2 | 1 | 13 | 5 | 63 | 0 | 0 | 19 | 0 | 1 | 24 | 2 | 1 | 3 | 0 | 1 | 5 | 0 | 2 | 11 | 0 | 3 | 0 | 125 |

Table 4.2. Legend for confusion matrix

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| textPersonName | **a** | numberDecimal | **i** | textCapSentences | **o** | textCapWords | **u** |
| textCapSentences | textAutoCorrect | textMultiLine | textLongMessage | **b** | numberSigned | **j** | textLongMessage | textCapCharacters | **p** | textCapWords | textAutoComplete | textLongMessage | **v** |
| textVisiblePassword | **c** | date | **k** | textLongMessage | **q** | textEmailAddress | **w** |
| textWebPassword | **d** | text | **l** | numberPassword | **r** | textUri | **x** |
| textCapSentences | textAutoCorrect | textMultiLine | **e** | textCapCharacters | **m** | textPersonName | textCapCharacters | **s** | textCapSentences | textAutoComplete | **y** |
| textMultiLine | **f** | numberSigned | numberDecimal | **n** | textWebEditText | **t** |  |  |

From the confusion matrix, we can observe that the maximum number of misclassification occurred for the class textWebPassword. This is understandable as there were more samples of textWebPassword than any other (3084). 233 samples of this class were misclassified as numberDecimal and 198 samples were misclassified as numberPassword. Alternatively, 220 samples from class numberDecimal were misclassified as textWebPassword and 182 samples from class numberPassword were misclassified as textWebPassword. 117 samples of numberDecimal were misclassified as numberPassword and 220 of them were misclassified as textWebPassword. 108 samples of text were misclassified as textLongMessage. 116 samples of textCapSentences were misclassified as text.

From table 4.3 we observe that the class textCapWords | textAutoComplete | textLongMessage has the highest precision (0.955) and the class textPassword has the highest recall (0.958). The class textWebEditText has the least precision (0.479) and recall (0.246). The data conforms to the confusion matrix as we found earlier that the class WebEditText had the maximum number of confusions found in our model.

As RandomForest performed better in both the datasets, we decided to do further analysis with the algorithm. As discussed earlier, Randomforest is an ensemble classifier which uses a bunch of decision tree classifiers within it and combines their outputs. We varied the number of decision trees our model can use and tested its performance.

Table 4.3. Precision, Recall and F-measure for Randomforest algorithm

|  |  |  |  |
| --- | --- | --- | --- |
| **Class Name** | **Precision** | **Recall** | **F-measure** |
| textPersonName | 0.71 | 0.648 | 0.678 |
| textCapSentences | textAutoCorrect | textMultiLine | textLongMessage | 0.66 | 0.615 | 0.637 |
| textVisiblePassword | 0.874 | 0.741 | 0.802 |
| textWebPassword | 0.718 | 0.76 | 0.738 |
| textCapSentences | textAutoCorrect | textMultiLine | 0.617 | 0.513 | 0.56 |
| textMultiLine | 0.65 | 0.673 | 0.661 |
| textPassword | 0.903 | 0.958 | 0.93 |
| textNosuggetions | 0.759 | 0.745 | 0.752 |
| numberDecimal | 0.743 | 0.778 | 0.76 |
| numberSigned | 0.614 | 0.387 | 0.475 |
| date | 0.76 | 0.792 | 0.776 |
| text | 0.665 | 0.65 | 0.658 |
| textCapCharacters | 0.757 | 0.658 | 0.704 |
| numberSigned | numberDecimal | 0.606 | 0.47 | 0.529 |
| textCapSentences | 0.71 | 0.766 | 0.737 |
| textLongMessage | textCapCharacters | 0.728 | 0.601 | 0.659 |
| textLongMessage | 0.71 | 0.601 | 0.651 |
| numberPassword | 0.798 | 0.779 | 0.788 |
| textPersonName | textCapCharacters | 0.876 | 0.798 | 0.835 |
| textWebEditText | 0.444 | 0.246 | 0.317 |
| textCapWords | 0.689 | 0.759 | 0.723 |
| textCapWords | textAutoComplete | textLongMessage | 0.955 | 0.947 | 0.951 |
| textEmailAddress | 0.888 | 0.906 | 0.897 |
| textUri | 0.805 | 0.707 | 0.753 |
| textCapSentences | textAutoComplete | 0.479 | 0.445 | 0.461 |
| Weighted average: | 0.76 | 0.763 | 0.76 |

Figure 4.5. Performance of RandomForest for varying number of decision trees

From the figure, we can observe that the accuracy tends to increase as we increase the number of trees, but not by much. As the features in trees are mutually exclusive, increasing the number of trees will lead to a high-biased, overfitted model which will lead to poor generalization. As a result we selected the model with 25 trees, having the accuracy of 76.99%.

**Future work**

Upon completion of the thesis I plan on extending the system in several ways. The major plans I wish to undertake are as follows:

* Improve the accuracy of the system.
* Integrating the system with SMV-Hunter
* Perform a full-quantitative analysis of the improvement in vulnerability analysis due to integration
* Generate a manually annotated dataset. The labels we are using for prediction are not provided for prediction but for Android OS to display proper keyboard. A model construted from a manually annotated dataset would definitely have better performance upon integration.

**CONCLUSION**

The research was conducted in order to aid automated interaction of android apps. Automated interaction aids in execution of vulnerable code paths and thus detecting vulnerable apps. In this dissertation we presented a robust and novel method of predicting input types of android user interfaces. The task was challenging as there was very little previous work and the problem was different than typical natural language processing tasks as the data is unstructured and contains tokens not present in English language. The research can be segregated into three segments. First, we had to construct appropriate corpus and stop words list from data. Then we retrieve the input types and their neighbors and construct feature vectors and generate the dataset. In the next step we conducted the experiment where we decomposed complex input types into simple ones and construct another dataset. We run several supervised learning algorithms and performed 10.fold cross validation. Among these algorithms, the model constructed with RandomForest algorithm, an ensemble learning algorithm had the best accuracy of 76.99%.

**ReFERENCES**

[1] IDC. www.idc.com,. (2015). *IDC: Smartphone OS Market Share*. Retrieved 4 December 2015, from http://www.idc.com/prodserv/smartphone.os.market.share.jsp

[2] Wikipedia,. (2015). *Google Play*. Retrieved 4 December 2015, from https://en.wikipedia.org/wiki/Google\_Play

[3] Wikipedia,. (2015). *Transport Layer Security*. Retrieved 4 December 2015, from https://en.wikipedia.org/wiki/Transport\_Layer\_Security

[4] Tools.ietf.org,. (2015). *RFC 5246 . The Transport Layer Security (TLS) Protocol Version 1.2*. Retrieved 4 December 2015, from https://tools.ietf.org/html/rfc5246

[5] Wikipedia,. (2015). *Certificate authority*. Retrieved 4 December 2015, from https://en.wikipedia.org/wiki/Certificate\_authority

[6] Wikipedia,. (2015). *Man-in-the-middle attack*. Retrieved 4 December 2015, from https://en.wikipedia.org/wiki/Man.in.the.middle\_attack

[7] Android, I. (2015). *Introduction to Android | Android Developers*. *Developer.android.com*. Retrieved 4 December 2015, from http://developer.android.com/guide/index.html

[8] Sounthiraraj, D., Sahs, J., Greenwood, G., Lin, Z., & Khan, L. (2014, February). Smv-hunter: Large scale, automated detection of ssl/tls man-in-the-middle vulnerabilities in android apps. In *Proceedings of the 19th Network and Distributed System Security Symposium*.

[9] McKinley, H. (2003). *SSL and TLS: A Beginners Guide* (1st ed., pp. 4.7). SANS Institute. Retrieved from https://www.sans.org/reading.room/whitepapers/protocols/ssl.tls.beginners.guide.1029

[10] Wikipedia,. (2015). *Android (operating system)*. Retrieved 4 December 2015, from https://en.wikipedia.org/wiki/Android\_%28operating\_system%29#/media/File:Android.System.Architecture.svg

[11] DuPaul, N. (2014). *Man in the Middle (MITM) Attack*. *Veracode*. Retrieved 4 December 2015, from http://www.veracode.com/security/man.middle.attack

[12] Georgiev, M., Iyengar, S., Jana, S., Anubhai, R., Boneh, D., & Shmatikov, V. (2012, October). The most dangerous code in the world: validating SSL certificates in non-browser software. In *Proceedings of the 2012 ACM conference on Computer and communications security* (pp. 38.49). ACM.

[13] Egele, M., Brumley, D., Fratantonio, Y., & Kruegel, C. (2013, November). An empirical study of cryptographic misuse in android applications. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security* (pp. 73.84). ACM.

[14] Fahl, S., Harbach, M., Muders, T., Baumgärtner, L., Freisleben, B., & Smith, M. (2012, October). Why Eve and Mallory love Android: An analysis of Android SSL (in) security. In *Proceedings of the 2012 ACM conference on Computer and communications security* (pp. 50.61). ACM.

[15] Fahl, S., Harbach, M., Perl, H., Koetter, M., & Smith, M. (2013, November). Rethinking SSL development in an appified world. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security* (pp. 49.60). ACM.

[16] Wikipedia,. (2015). *Machine learning*. Retrieved 4 December 2015, from https://en.wikipedia.org/wiki/Machine\_learning

[17] Smith, M., & Martinez, T. (2011). Improving classification accuracy by identifying and removing instances that should be misclassified. *The 2011 International Joint Conference On Neural Networks*. http://dx.doi.org/10.1109/ijcnn.2011.6033571

[18] Wikipedia,. (2015). *Bias–variance tradeoff*. Retrieved 4 December 2015, from https://en.wikipedia.org/wiki/Bias%E2%80%93variance\_tradeoff

[19] Wikipedia,. (2015). *Ensemble learning*. Retrieved 4 December 2015, from <https://en.wikipedia.org/wiki/Ensemble_learning>

[20] Wikipedia,. (2015). *Decision tree learning*. Retrieved 4 December 2015, from <https://en.wikipedia.org/wiki/Decision_tree_learning>

[21] Cse.unsw.edu.au,. (2015). *Induction of Decision Trees*. Retrieved 4 December 2015, from <http://www.cse.unsw.edu.au/~billw/cs9414/notes/ml/06prop/id3/id3.html>

[22] Wikipedia,. (2015). *Random forest*. Retrieved 4 December 2015, from <https://en.wikipedia.org/wiki/Random_forest>

[23] Nan, Y., Yang, M., Yang, Z., Zhou, S., Gu, G., & Wang, X. (2015, August). Uipicker: User-input privacy identification in mobile applications. In *USENIX Security* (pp. 993.1008).

[24] Lextek.com,. (2015). *Stopword List 1*. Retrieved 4 December 2015, from <http://www.lextek.com/manuals/onix/stopwords1.html>

[25] Developer.android.com,. (2015). *TextView | Android Developers*. Retrieved 4 December 2015, from http://developer.android.com/reference/android/widget/TextView.html#attr\_android:inputType

**VITA**

Md Shamsul Arifin got his BS in Computer Engineering in 2011 from North South University, Dhaka, Bangladesh. He worked as a software engineer at Eyeball Networks from 2011 to 2012. Later he worked at Research Assistant at North South University in Spring 2013. He joined graduate school of University of Texas at Dallas in the fall of 2013. His research interests include Machine Learning, Natural Language Processing and Digital Signal Processing.