The Journal of International Scientific Researches
2021, 6(1)

Usage of Artificial Intelligence to Improve Secure Software Development

Yapay Zekâ ile Güvenli Yazılım Geliştirme

Abstract

C# is an object-oriented software language that was developed by Microsoft and runs on the .NET framework. It has the ninth in Github, and it would cause a great impact in .NET. Main method for the situation is mainly using the machine learning techniques to analyze the source code and determine the security level of a source code. By evaluating the source code elements of software, the framework could proclaim that either a software product is secure or not, thus developer can make necessary arrangement in his or her code. It is important, because securing the code during the creation of the software is easy as compared with the end of the software creation. As part of a limited literature review study, we have analyzed 3 main articles in detail. An assessment of the review is conducted according to artificial neural network logic to help improve the Security of C# software development and further evaluations and deductions are being conducted.

Ahmet Efe
PhD, CISA, CRISC, PMP, Ankara Development Agency, Ankara Turkey,
mail: icsiacag@gmail.com
https://orcid.org/0000-0002-2691-7517

Article Type / Makale Türü
Research Article / Araştırma Makalesi

Keywords
.NET, C#, ASP, Software Security, Artificial Neural Network.

Anahtar Kelimeler
.NET, C #, ASP, Yazılım Güvenliği, Yapay Sinir Ağı.

Submitted: 11 / 11 / 2020
Accepted: 20 / 02 / 2021

Öz

C #, Microsoft tarafından geliştirilen ve .NET Framework üzerinde çalışan bir nesne yönelimli yazılım dili olup, Github'da dokuzuncu sıradadır. Güvenli yazılımda önemli olan ana yöntem, kaynak kodunu analiz etmek ve bir kaynak kodun güvenlik seviyesini belirlemek için çoğunlukla makine öğrenme tekniklerini kullanmaktır. Yazılımın kaynak kod unsurlarını değerlendirerek, çerçeve bir yazılım ürününün güvenli olup olmadığını, dolayısıyla geliştiricinin kodunda gerekli düzenlemeleri yapabileceğini ilan edebilir. Yazılımın oluşturulması sırasında kodu güvence altına almak, yazılımın oluşturulmasının sona ermesine kıyasla daha kolaydır. Sınırlı bir literatür taraması çalışmasının bir parçası olarak üç adet makaleler detaylı olarak incelenmektedir. Bu çalışmamızda gözden geçirmenin bir değerlendirmesi, C # yazılım geliştirme güvenliğinin arttırılmasına yardımcı olmak için yapay sinir ağı mantığına göre yapılmaya çalışılmakta ve buna göre değerlendirme ve çıkarsamalar yapılmaktadır.

Introduction

Advanced machine learning, intelligent applications, digital twin (virtual copy) projects and speech systems have a revolutionary destructive power in the industry today. The industrial use of artificial intelligence can take place together with the security several enhancements to the area. Antivirus companies, including Microsoft, Google, Cisco, Symantec senior technology companies such as cyber safe artificial intelligence applications, big budgets are working on these issues.

Threat detection in today's AI and machine learning technology is an important focal point. This kind of software is not only just watching human behavior, but also detects things that went wrong, can also provide timely warnings. Recently, the ever-expanding data volumes (large) aggregates, monitor and analyze applications; hackers and malicious actors carefully positioned to be one-step ahead of the recent war. It is inevitably a machine learning strategy with the use of large industrial companies, reduce cyber threats became a starting point. As a result, company networks, internet and machines connected to a large data streams provided through and analyzes that can interpret them with advanced technology (Amos, 2013).

In fact, it is not possible to put a limit on the artificial intelligence that recommends what to do with. For what purpose to be used, depends on the imagination of developer. Some machine learning is used in cyber defense applications are as follows;

- Spam Filtering
- Network Intrusion Detections and Preventions
- Fraud detections
- Credit scorings and best offerings
- Botnet Detections
- Secure User Authentication
- Hacking Incident Forecasting
- Cyber-Security Ratings
- Secure Software development

Research question is defining a framework for determining the security level of a C# code. C# as an object-oriented language developed by Microsoft, has the ninth in Github and it would cause a great impact in .NET. Main method for the situation is mainly using the machine learning techniques to analyze the source code and determine the security level of a source code. Our focus point is the source code files that written in C#. By evaluating the source code elements of software, the framework could proclaim that either a software product is secure or not, thus developer can make necessary arrangement in his or her code. It is important, because securing the code during the creation of the software is easy as compared with the end of the software creation (Github, 2018).

There are many applications on android operating system are to determine an android application is secure or not. I need to inspect the android researcher's approach. Mainly, their approach is divided into two ways: Static and dynamic. Static analysis mainly based upon the peculiarities of a device, framework or operating system, the static analysis may be shallowly thought as a basic if else structure, that is say, if operating system is on a specific status, think it as a unsecure state. The second approach is implementing a dynamic analysis on the code.

The main challenging point is making a dynamic analysis, because there are a wide variety of developers that various expertise levels and the code set that the developer has created are very immense. Machine learning is the sole approach for implementing the dynamic analysis and different machine learning algorithms are suitable for the different security requirement. Finding the proper algorithm is the main key point about the dynamic analysis. In my inspection, Artificial Neural Network and its variations is the main learning approach to implement learning (Moser, 2007).

The C# code can run only on .NET framework, and its security is main concern. Microsoft creates .NET framework. .With .NET, developers can use multiple editors, languages, and libraries to build for mobile, web, desktop, gaming, and IOT (Microsoft, 2018). The current version of it is 4.7.2.

.NET Framework security system implements an Extensible derived class inheritance pattern. The hierarchy is as follows:

- Algorithms type classes, such as Symmetric Algorithm, Asymmetric Algorithm or Hash Algorithms. This level is abstract.
- Algorithms classing inherits from an algorithm type; for example, AES, RC2, or ECDiffieHellman.
- Implementation of algorithms classing that inherits from an algorithm; for example, AesManaged, RC2CryptoServiceProvider, or ECDiffieHellmanCng.

Using this scheme, of classes derived from a new algorithm or a new application of an existing algorithm is easy to add. A specific algorithm to create a new application that you create a derived class that is not abstract algorithm. An algorithm can be used for different applications as an example, consider a symmetric algorithm. Basis for all symmetric algorithms, Symmetric Algorithm, inherited by the following algorithms:

- DES;
- TripleDES.
- Aes;

- Rijndael;
- RC2;

Aes since inherited by two classes: AesCryptoServiceProvider[1] and AesManaged[2] . AesCryptoServiceProvider Class, Aes, the Windows Cryptographic API (CAPI) application is a wrapper around the AesManaged class, is written entirely in managed code. Application is sure to include the next generation (CNG), managed encryption and there is a third type of CAPI applications. CNG algorithm is an instance of the ECDiffieHellmanCng[3] . CNG algorithms available in Windows Vista and above.

You can choose which application is best for you. Managed applications, supporting .NET Framework is not available on all platforms. CAPI applications can be used on earlier operating systems and is now being developed. CNG new development projects will take place here is the most recent application. However, managed applications, Federal information processing standards (FIPS) have not been approved and the wrapper classes can be slow.

C# can be used at Web application framework such as ASP.NET Web Forms and ASP.NET MVC. Since MVC is newer, I will investigate the ASP.NET MVC framework. Artificial Neural Networks are the important concept at machine learning. It is a simulation of mammal brain programmatically and organizing the neurons of it results as a learning machine. The learning machine here is determining the security level of a source code and if it is not secure, warn the programmer.

1. General Problems of Coding and AI Usage

The rapid increase in the number of applications in the field of Artificial Narrow Intelligence (ANI) all over the world and the creation of successful products, sometimes seem like a joke and stay away from "Will my profession disappear in the future? continues to make us ask his question. First studies on software source codes; based on code magnificence and quality. Number of lines in the code, spaces, comments, functions, parameters, variables, conditions, loops, etc. different criteria were calculated. "Cyclic Complexity - Cyc.Comp." Calculated by static code analysis methods. We can say the oldest of them. Along with the object-oriented programming, other criteria such as RFC, WMC (Weighted Methods per Class), DIT (Depth of Inheritance Tree), CBO (Coupling Between Object Classes), LCOM were derived by reference to basic criteria such as "coupling and cohesion". Again, using source codes and various statistics-based calculations, "software effort and time estimation (eg, how many man-days / hours does this job take?)" Is one of the topics that have been studied for a long time. Function score, IFPUG (International Function Point Users Group), object score, scenario score, COSMIC score and COCOMO are some of the metrics that are widely used in both academia and industry.

“Finding the author of the source code” is one of the spectacular works in ANI field. Program writing habits / styles of software developers, naming choices, design approaches, average number of lines written per function, style of using comment lines, etc. Trying to create a pattern or to train a model by using many variables as input. "Finding similarity" between source codes is one of the important issues in the security category (plagiarism). Natural language processing techniques (preprocessing, stems etc.) and approximate text similarity finding algorithms (Levenshtein, Jaccard, Jaro, N-gram etc.) or new generation Deep Learning architectures are used. There are also different studies on the matching of the error records with the source codes. For example, the location of an error entered in the source code (in which file, class, function, line?) Ie "bug localization" is one of the most studied subjects and is also used in the industry. Automatically locating a transmitted error record or suggesting possible source codes to the software developer makes life easier for

[1] For detailed information see: https://docs.microsoft.com/tr-tr/dotnet/api/system.security.cryptography.aescryptoserviceprovider?view=netframework-4.7.2
[2] For detailed information see: https://docs.microsoft.com/tr-tr/dotnet/api/system.security.cryptography.aesmanaged?view=netframework-4.7.2
[3] For detailed information see: https://docs.microsoft.com/tr-tr/dotnet/api/system.security.cryptography.ecdiffiehellmancng?view=netframework-4.7.2

developers. Finding similar errors for a transmitted error record and the assumption that "the resolution of similar errors will be similar" is one of the frequently used approaches. For error similarities, clustering, information retrieval, and natural language processing methods, which are generally included in the "unsupervised" learning category, are used.

Using the criteria (X1 ... Xn) of the source code, errors (Xn + 1) and the source codes that threw the error (y) together as an attribute, the status / degree of a written source code (program part) to be defective or not, a classification or It can be taken as a regression problem and a solution can be found. At this point, it is necessary to remember the fact that "the last modified code will probably explode" and add this information to the data set as an attribute. It is also important that the modules containing the source code with the strongest probability of error should be tested the most. Although "automatic source code generation" using the requirement documents is a difficult issue in the field, there are different studies that have been done. In these studies, natural language processing techniques and rule-based methods are generally used. POS (Part-of-Speech Tagging) tagging, that is, understanding whether words are nouns, adjectives, verbs, pronouns or adverbs is very critical at this point. For example, in a system developed with the object-oriented paradigm, the names (noun) in the requirement documents are generally set out with the assumption that the candidate class and properties are. Often mentioned names are probably class names. Nouns and verbs in the same sentence can be class behaviors (method / function), etc. Using rules and machine learning models, source codes can be generated automatically (Kılınç, 2019).

2. Research Method

Main research methodology is a limited literature survey and analysis based on the data gathered from study examples. Literature survey may mainly divide into two groups, security principles of C# code and inspecting the prior papers of Android application security papers. By carefully inspecting the Android papers and android security products and understand their methods and approaches. This approach will guide us to create a library of a product for determining the security level of a C# source code. After all, we would deliberately write an unsecure source code to evaluate the performance of the newly devised approach.

At that point, we have analyzed “Deep Learning for Secure Mobile Computing Review”, ”MARVIN: Efficient and Comprehensive Mobile App Classification Through Static and Dynamic Analysis”, ”Security Assessment of Software Design using Neural Network” and ”DynaLog: An Automated Dynamic Analysis Framework For Characterizing Android Applications”. We have analyzed further articles; however, we could not include them all in this study.

3. Analysis Of Papers

There are many studies regarding application of AI within software development. Here we selected a few of them for our analysis:

3.1. Deep Learning for Secure Mobile Computing Review

At that passage, we have inspected the paper, *Deep Learning for Secure Mobile Edge Computing, Yuanfang Chen, Yan Zhang and Sabita Maharjan.*

Mobile edge computing is becoming more and more prevalent and as new types of technology arise, new types of attacks could arise. Considering the fact of mobility of any user of mobile device, an attacker can tamper with the address of the connected device in certain location, by tampering; an attacker would manipulate the connection.

The research problem described in the paper is as illustrated in Figure 1. Assume that an attacker tries to harm the network by using the customer’s device. When a customer tries to connect device, so the attacker initiates an attack towards device 1 and device 1 would repeal the attack. As the customer moves, it attempts to connect to the closest edge-computing device 2 in location B, and so the attacker would attack to device 2. At that moment, since customer is now not in location A, the connection to edge computing device 2 is successful.

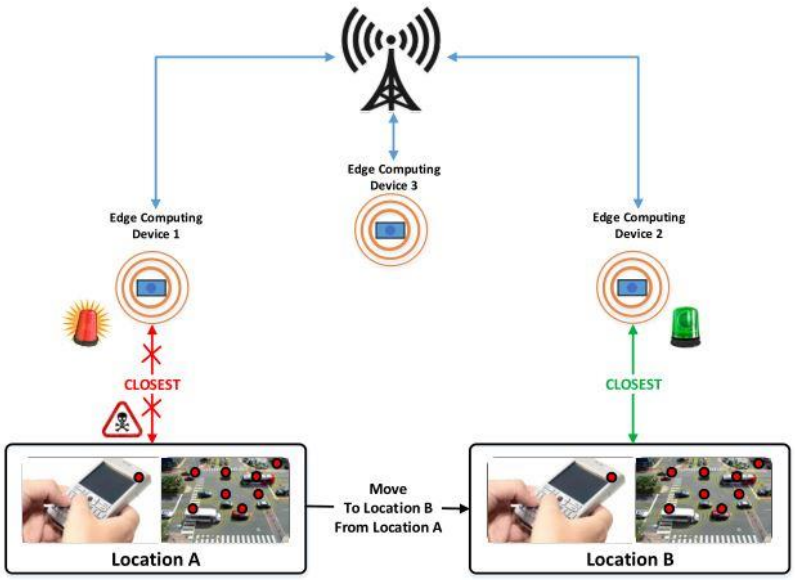

Figure 1. An Overview of Tampering With the Network

In the paper, they try to investigate the methods that try to detect such attacks. There are numerous of studies that association of machine learning and security, in their paper, they incorporated the location information. They use deep learning to detect malicious applications. They classify the existing malicious detection systems into main two categories as static analysis and dynamic analysis. Static analysis inspects the source code of the applications without executing them. This approach also inspects permissions, used hardware components and data-flow usage.

Static Analysis

DERBIN and DroidMat are the prominent solution for static analysis. Both approaches inspect Android.xml and tracing system calls. In the work of Sanz, the authors have tested many algorithms and they conclude, "the Bayesian network is a better classifier compared with those based on a random forest and decision tree" (Sanz, 2012).

Dynamic Analysis

Shabtai et al proposed an artificial neural network based system to detect unknown malicious Android applications that are based on analyzing permissions of the applications' and system calls. After that, they have analyzed four different machine-learning algorithm, decision tree, random forest, SoftMax regression and support vector machine (Shabtai, 2012).

They propose deep learning model to detect security threads, and they use two different engines: a feature-preprocessing engine and a malicious application detection engine.

Feature Preprocessing Engine

Each application's features shall be analyzed in advance to characterize the applications. These features are required permissions, sensitive APIs and dynamic behavior.

Malicious Application Detection Engine

They have used Support vector machine, K-means algorithm, Decision tree as a static analyzer and neural network and support vector machine as a dynamic analysis. Among all, deep learning model is the most suitable approach. That is because it has a great specialty named feature learning.

Evaluation of Their Approach

They followed the traditional approach of 0.8 percent train data and 0.2 test data. In addition, they have utilized other machine learning algorithms such as decision tree, random forest, softmax regression, and support vector machine. According to their test results, deep learning methods outperform other approaches in terms of gain and accuracy.

Conclusion

In the paper, they have proposed a deep learning-based model to detect malicious attacks. Deep learning-based learning model has significant advancement in terms of accuracy and reliability.

3.2. Marvin: Efficient and Comprehensive Mobile App Classification through Static and Dynamic Analysis

Android is the most popular smartphone operating system, with the market share of 85% (IDC, 2018). Unlike the IOS, the Google Play is the not the sole source of application. To prevent application epidemic that grows, Google introduced Bouncer in February 2012 (Lockheimer, 2012). This application checks the new coming application prior to publishing on the Play Store, and it allows the publication of the application or not. Its claim is reducing the malicious applications by 40%. When you choose an application to install a device, the end user would consider some information:

1- Trustworthiness of the application's origin
2- Application comments that written by other users
3- Permissions asked by the application
4- Google's application verification service results
5- Antivirus scanner results

Every approach here has own shortcomings. In the first step, trustworthiness is hard to establish, and it prevents even secure applications that being developed by rookie developers. In the second step, application reviews that written by users are not accrue because ordinary users would not be aware of the security flaws at the applications (Chia, 2012). Besides, the attacker can himself manipulate the application comments. In the third step, the average user does not know the sophisticated permissions (Felt, 2012). In the fourth step, anti-virus scanners would cause strain on battery and CPU that limited in a mobile device (Cheng, 2007; Oberheide, 2008).

Main differences between detecting the malware in Windows malware and Android malware are that there is a limited CPU and power resource in Android. SmartSiren uses collaborative virus detection system (Cheng, 2007). Oberheide et al. proposed a virus scanning system that runs apart from the mobile device (Oberheide, 2008). Paranoid Android simulates the execution of an android application in the cloud for perform resource intensive detection techniques such as antivirus scanning and dynamic taint analysis (Portokalidis, 2010). ThinAV changes the Android Package Installer for consult web-based scanners before installing the application (Jarabek, 2012).

As we seen before, there are two different approaches on malware detection. The detection can be done based upon static features and dynamic features. In the paper, as static features, Kirin ranks security-critical combinations of requested permissions based on a manual assessment (Enck, 2009). AppProfiler alerts end users to privacy-connected issues by matching the application against a knowledge based on more than two hundred API calls (Rosen, 2013). Apvrille et al. suggest a heuristic engine that statically pre-processes and prioritizes samples to foster the notice of new Android malware in the wild (Strazzere, 2012).

Keeping in mind of our ultimate goal of automatically detect malicious applications, RiskRanker (Grace, 2012). and DroidAPIMiner are the most closely related research (Aafer, 2013). Even though these approaches were valuable, all those approaches lack the ability to analyze code that is obfuscated or loaded dynamically at runtime, a prevalent feature of apps as evidenced by a recent large scale study (Lindorfer, 2014), unless they are complemented by some form of dynamic analysis, as recently proposed in StaDynA (Zhauniarovich, 2015). Their proposed system classifies more malware correctly as compared with its counterpart, as well as with a lower false positive rate (Shabtai, 2012).

STREAM is another framework for evaluating mobile malware classifiers based on the same features as Andromaly but an equally limited testing set of only 50 applications (Amos, 2013). STREAM suffers much from the substantial false positive rates. Afonso et al. dynamically inspect Android apps to use the number of invocations of API and system calls as atomic-grained features to train various classifiers (Afonso, 2014). The main common point of the approaches here mainly suffers from the size of the data sets; data sets on the field are fairly limits.

One of the major developments that were observed in a recent study is the increasing importance of dynamic analysis to completely capture an application's characteristic features, as a result, relying on numerous existing approaches solely relying on s static analysis obsolete (Lindorfer, 2014). For example, a harmful application could be hidden in a static source, even in an innocuous

looking image that downloaded from external sources (Albertini, 2014). This phenomenon makes static analysis of the complete application is impossible. In addition, malware applications become more and more clever. Conversely, approaches solely depend on dynamic analysis can be bypassed by malware. Their proposed program, Marvin is supposed to combine both static and dynamic analysis based upon common features of Android operating system and it becomes more accurate and robust.

Approach

Normally, an application can either classify as a benign program (0) or malware (1). However, in real life, there are some gray zones between them; for example, an adware is neither a benign program nor malware. Their supposed system a step forward against the accurately classifies applications.

User can submit any application that they have suspicion over and they can get an input in a less-technical, user-friendly manner.

Main difference between Marvin and Google's Bouncer is, Google Bouncer only scans the applications published by Google Play to approve or reject. Marvin can be considered as a lightweight alternative for antivirus scanners.

Details of Marvin

There are two modes of operation in Marvin, they are training mode and classification mode.

Training Mode

Marvin operates in two distinct modes, they are training node and classification node. In training node, it gathers necessary information to create a model for classification. At this stage, Marvin first analyses the comprehensive features set from these applications via static and dynamic analysis. At that point, Marvin tries to select the best feature that determines the determinateness of an application considering avoiding overfitting.

Classification Mode

Marvin accepts user submission via a mobile or web interface. Based upon both the features exposed by the application that submitted and the training data that Marvin has encountered, Marvin assesses the risk and outputs the application's malicious score. The first-time analysis of an application would take considerable amount of time, say several minutes. However, thanks to growing number of cache results of prior analysis reports, this time could reduce.

Feature Extraction

Feature selection is the most vital point of Marvin, allowing the scoring the new coming application’s determinateness. Static analysis could give valuable information about the application that because can be escaping the attention, or features that cannot be captured from the dynamic analysis because of the possible weak code coverage of dynamic analysis. Dynamic analysis shows the features of the runtime aspects of application, even from external sources, or unhindered executables during executables.

Dynamic Analysis Features

As research on x86 malware showed static analysis techniques are suffering to evasion by code obfuscation techniques (Moser, 2007). Furthermore, features should inherently represent the malicious behavior to be detected, so attackers may conceal the features intently for evading the learning method, e.g. with mimicry attacks (Laskov, 2014).

They take different precautions for different protocol level, say for SMTP, FTP, DNS, HTTP, and IRC. For grabbing the dynamic analysis features, they upgraded the automated and publicly available dynamic analysis sandbox Andrubis that they proposed in previous work (Lindorfer, 2014). It analyses the application in four minutes, according to their research, this time is optimal for tradeoff between the utilization of analyze resources and observed features in the experiments before.

Choosing a Classifier

The classifier is a hard-hitting component of Marvin, as its accuracy will directly determine the success of the malice score.

Support Vector Machine (SVM)

An SVM works the same way as a linear classifier, with a hyperplane splitting the feature space into two sections.

Prediction Probabilities as Scores

In the common case, the sign of the margin give the result of a binary linear classifier: it either attributes the input to one or the other class.

3.3. Security Assessment of Software Design using AI and Neural Network

Because of the dazzling competition on the information technology market, most of the software developers ignore the security aspects of information technology. This comes out as a potential danger for attackers, and because of the increasing importance of the software. Detecting and solving the security issue of a software can only be accomplished at the start of the

Related Works on Security Assessment of Software Design

For the model the security aspect of an application, there are many approaches such as thread modelling, architectural risk analysis and so on.

Inner Workings of the Application & Conclusion

They have used three layered feed-forward back-propagation neural network for its simplicity and speed. The main issue there is capturing the attack scenario. They capture them and they are "source of attack", "the attacker", "dependencies", "input validation", "target of the attack", "attack vector", "output encoding to external applications / services", "attack type", "authentication", "http security", "access control", "error handling and logging". After the performance of application, it gives the expected result.

3.4. DynaLog: An Automated Dynamic Analysis Framework for Characterizing Android Applications

D.1 Introduction

DynaLog is a framework that can both implement static analysis and dynamic analysis. DynaLog has several packages. These include:

a) Application trigger/exerciser
b) APK instrumentation module
c) Emulator-based analysis sandbox
d) Behavior/features logging and extraction
e) Log parsing and processing scripts.

Code coverage is an important issue because it is practically impossible to cover all application's path and source code during analysis. Droidbox contains MonkeyRunner, an application that implements the automated testing of applications. It sends random events for Android application under test.

Dynalog Framework Evaluation

As expected, they have performed this application over 2226 malware and bening applications in order to investigate its capability of differentiating between benign and malware applications. They used the capability of the Droidbox provides before. In particular, the component "Recvaction" is used to extract several 'events' and 'actions' shown in Table III.

Table 1. Android Events and Actions Related To Malware (Zhou, 2012)

Event	Abbreviation	Event	Abbreviation	Event	Abbreviation
BOOT_COMPLETED	BOOT (Boot Completed)	SMS_RECEIVED WAP_PUSH_RECEIVED	SMS (SMS/MMS)	ACTION_MAIN	MAIN (Main Activity)
PHONE_STATE NEW_OUTGOING_CALL	CALL (Phone Events)	UMS_CONNECTED UMS_DISCONNECTED	USB (USB Storage)	CONNECTIVITY_CHANGE PICK_WIFI_WORK	NET (Network)
PACKAGE_ADDED PACKAGE_REMOVED PACKAGE_CHANGED PACKAGE_REPLACED PACKAGE_RESTARTED PACKAGE_INSTALL	PKG (Package)	ACTION_POWER_CONNECTED ACTION_POWER_DISCONNECTED BATTERY_LOW BATTERY_OKAY BATTERY_CHANGED_ACTION	BATT (Power/Battery)	USER_PRESENT INPUT_METHOD_CHANGED SIG_STR SIM_FULL	SYS (System Events)

There are numerous features that dynalog has extracted. They are phone state, package manager and services that a phone consumes. It analysis the features, and from that point, and they realized that some events frequently exploited by malware, such as event BOOT_COMLETED.

Results of malware and benign samples comparison

At that point, they compare the malware applications and well-natured application's main interlocutor events, and they realized that most malignant applications consume specific Android events.

Limitations of the DynaLog framework

DynaLog is not prone to the limitations of dynamic analysis that well known by the academia (Hung, 2014; Reina, 2013; Afonso, 2014). In addition, they do not log anywhere the result of security analysis result of an application.

Related Works

At that point, they analyzed the approaches of information technology for security applications. As we stated, they embraced the static and dynamic analysis. The static analysis's main concern is, overcoming the security scrutiny is easy. Dynamic analysis can be divided into three parts:

- Machine learning Approach
- Open Source Frameworks
- Online services

Machine learning approach's weak spot is limited data to train. The machine learning based frameworks has trained based upon a limited set of malign applications, so their confidence is questioned. Open source frameworks are on the rise, and it is gradually increasing its popularity. The most recent online service is Google's Bouncer; however, Oberheide and Miller presented their research regarding Bouncer (Oberheide, 2016). Andrubis is another online service solution for online services by International Secure Systems Lab, however it is not open source (Weichselbaum, 2014).

4. Original Wording of the Problem

MVC is an important Web application framework that for run a both basic and enterprise web site. As stated above, security peculiarities should be stated well for a MVC web application. For example, login process is an important from the security point of view. Passwords should not keep unencrypted at the database, so, creating a new user process code shall contain encryption process. At that point, the devised security framework should allow programmer if he or she tries to save unencrypted passwords to database. Here is the critical aspect of a web site that can be identified from source code:

Authorization & Authentication

1. Login forms shall send data through a secure connection
2. Login form shall use HTTP Post rather than GET
3. Application shall validate access request
4. web.config shall not contain clear users credentials.
5. web.config shall be configured to most secure settings for cookies.

Filtering User Input

1. All instances of the *request* object shall be identifying to be sure you properly filtered this input.
2. Other forms of indirect input shall be searched for and filtered, including input from the application itself.

Database Security

1. Database connection string shall be kept secret at the web.config.
2. Application shall be protected from SQL injection attacks.
3. SqlParameters shall be used for database operations if entity framework solutions have not been used.
4. Deep TabNine As New tools for AI based Software development

Deep TabNine[4], an artificial intelligence application, is able to analyze lines of code written by developers so far and automatically add new lines of code. Supporting multiple programming languages, this tool will help developers to code faster. Artificial intelligence that automatically completes code of developers. The Deep TabNine application, which uses the GPT-2 deep learning

[4] For more details see: https://tabnine.com/blog/deep/

model developed by OpenAI, was developed by Jacob Jackson, a computer science student at the University of Waterloo, Canada. Deep TabNine was trained using GitMub's database of over 2 million files, enabling automatic prediction of codes. It supports programming languages such as Python, Java, JavaScript, PHP, C, C++, C #, Kotlin, TypeScript, HTML, Objective-C, Go, CSS, Rust, Ruby, Swift, Haskell, Scala, Perl, OCaml, SQL and Bash. However, Deep TabNine does not perform very well on laptops at this time because it requires a lot of computing power.

Conclusion and Discussion

In this new world of artificial intelligence, software developers do not need to design a unique algorithm for every problem. Instead, most studies must focus on building datasets that reflect desired behavior and manage the educational process. It is known that the programming model that uses the most important developments in software today does not require a significant amount of real programming. For the future of software development, this means it is necessary to consider the following possibilities (Bornstein, 2018):

1. Programming and data science must increasingly converge. Most software will not require "end-to-end" learning systems for the foreseeable future. They have to rely on data models to provide clear logic and basic cognitive abilities to interface with users and interpret results. "Should I use an artificial intelligence or a traditional approach to this problem?" such questions will be heard increasingly. Designing smart systems requires expertise in both.

2. Ordinary AI developers - not just highly intelligent academics and researchers - will be counted among the most valuable human resources for software companies in the future. This could mean an irony for traditional coders who have automated jobs in other industries since the 1950s and are now moving to partial automation of their own work. While the demand for their services is certainly not reduced, those who want to stay in the foreground may need to be skeptical at a certain dose and test developments in the field of artificial intelligence.

3. Secure software applications that work in harmony with artificial intelligence need to be created. Machine learning is no longer in a primitive stage. Research has shown that many artificial intelligence models are difficult to explain, easy to deceive, and prone to bias. However, sophisticated measures are also increasing in this regard. Tools to address these issues, among others, may be needed to unlock the potential of AI developers.

4. A computer instruction metaphor will likewise be familiar to developers and users. This will reinforce the belief that computers do exactly as we say, and similar inputs always produce similar outputs. On the contrary, AI models tend to behave like independent systems that live, breathe, and apply what they learn. New tools will allow them to behave more like open programs, especially in security-critical environments, but if we set security measures very tightly from the architecture, we may not run the risk of losing the value of these systems. As we develop and use artificial intelligence applications, we need to understand and accept risk-taking results.

Confidentiality and accessibility problems will continue in training deep learning models. Firms will not want to share their source codes for training deep learning models. The widespread use of open-source code may be a solution to this problem to some extent. In addition to automatic code generation, automatic (and smart) API generation will be possible and applications will be able to stand up quickly. Artificial Narrow Intelligence (ANI) applications will help software developers, but this will make software developers with in-depth knowledge of computer architecture, operating system structure/design, data structures, algorithm analysis, programming language design, and software design/architecture (much) more valuable.

New roles/job descriptions will begin to emerge within the software team (or IT team) (especially in the configuration management section) to increase the quality of ANI applications and to feed them with correct data. During the code review phase, chatbots (or assistants) will be engaged. Maybe they will not be very smart, but they will guide teams according to the business concepts (for example; airline, finance, e-commerce, fashion, etc.), system requirements, architectural patterns, design patterns and software quality criteria. The need for clarity of software

requirements will continue to be a handicap for ANI applications. The complex relationships between requirements will make it difficult to fully resolve these requirements.

The security framework that devised in a manner that identify the code sections of an ASP.NET Web project. As stated above, security section is within the Authorization & Authentication, processing the input and database contact phases. Defining details of these sections in a project is a hard task. At that point, framework shall determine the section of the code. To do this, we should determine the keywords of the language and read entire source code file. When framework encounters the keywords, it concludes the proper section of the code. When it encounters the code section, artificial neuron network shall be used. It reads the section and train the artificial neural network for determine the security level of the source code.

The artificial neural network shall be fed in advance by benign and malicious code snippets; therefore, it trains its model. After that, the network becomes trained and it will able to classify an incoming source code. Training the network needs some time and data, so it needs time to mature.

Even though neural networks are powerful tool for detecting security concern, it is not an only answer. Static analysis shall also be utilized. By harmonizing static analysis and dynamic analysis, they will be able to close the gap between each other. Every scan result of a source code shall be reported to a central database; therefore, devised application should support the service-oriented architecture. One of the possible shortcomings here is reservation of software developers. Software developers may not feel comfortable the product that reads his or her source code.

References

Aafer, Y. W. D. (2013). DroidAPIMiner: Mining API-Level Features for Robust Malware Detection in Android. International Conference on Security and Privacy in Communication Networks (SecureComm).

Afonso,V. M. M. F. (2014). Identifying Android malware using dynamically obtained features. Journal of Computer Virology and Hacking Techniques.

Agarwal, A. (2016). How to integrate security into your SDLC. http://searchsoftwarequality.techtarget.com/tip/0,289483,sid92_gci1174

Albertini, A. (2014). Hide Android Applications in Images. Black Hat Europe.

Amos, B. H. A. (2013). Applying Machine Learning Classifiers to Dynamic Android Malware Detection at Scale. International Conference on Wireless Communications and Mobile.

Bornstein M. (2018) "The Coming Revolution In Software Development" ForbesMagazine, https://www.forbes.com/sites/valleyvoices/2018/08/24/software-developer-revolution/#3d422d051e4a

Cheng, J. S. H. (2007). SmartSiren: Virus Detection and Alert for Smartphones. International Conference on Mobile Systems, Applications, and Services (MobiSys).

Chia, P. H. Y. Y. (2012). Is This App Safe? A Large-Scale Study on Application Permissions and Risk Signals. International Conference on World Wide Web (WWW).

Enck, W. M. O. (2009). On Lightweight Mobile Phone Application Certification. ACM Conference on Computer and Communications Security (CCS).

Felt, A. P. (2012). Android permissions: User attention, comprehension, and behavior. Symposium on Usable Privacy and Security (SOUPS).

Github. (2018, 1). Github language stats. https://madnight.github.io/githut/#/pull_requests/2018/1

Grace, M. Y. Z. (2012). RiskRanker: Scalable and Accurate Zero-day Android Malware Detection. International Conference on Mobile Systems, Applications, and Services (MobiSys.

Hung, W.-C. W.-H. (2014). DroidDolphin: A Dynamic Android Malware Detection Framework Using Big Data and Machine Learning. Conference on Research in Adaptive and Convergent Systems.

IDC. (2018). Smartphone OS. https://www.idc.com/promo/smartphone-market-share/os

Jarabek, C. D. B. (2012). ThinAV: Truly Lightweight Mobile Cloud-based Anti-malware. Annual Computer Security Applications Conference (ACSAC).

Kılınç, D. (2019) Yapay Zekâ Uygulamalarının Yazılım Geliştirme Sürecine Etkisi Ne Olacak? Yazılımcılar İşsiz mi Kalacak?, Medium Blog, https://t.ly/vNu3

Laskov, N. S. (2014). Practical Evasion of a Learning-Based Classifier: A Case Study. IEEE Symposium on Security and Privacy (S&P).

Lindorfer, M. N. (2014). Andrubis - 1,000,000 Apps Later:A View on Current Android Malware Behaviors. International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS).

Lockheimer, 1. (2012). Android and Security. http://googlemobile.blogspot.com/2012/02/android-and-security.html

Meier, J. D. (2010, 10 24). Threat modelling web applications. http://msdn.microsoft.com/en-us/library/ms978516.aspx

Microsoft. (2018, 5 11). What is .NET? https://www.microsoft.com/net/learn/what-is-dotnet

Moser, A. C. K. (2007). Limits of Static Analysis for Malware Detection. Annual Computer Security Applications Conference (ACSAC).

Oberheide, J. a. (2016, February, 14). https://jon.oberheide.org/blog/2012/06/21/dissecting-the-android-bouncer/

Oberheide, J. K. V. (2008). Virtualized In-cloud Security Services for Mobile Devices. Workshop on Virtualization in Mobile Computing (MobiVirt).

Portokalidis, G. P. H. (2010). Paranoid Android: Versatile Protection for Smartphones. Annual Computer Security Applications Conference (ACSAC).

Redwine, S. T., & et al. (2004). Process to produce secure software: Towards more secure software. National Cyber Security Summit, Vol. 1.

Reina, A. F. (2013). A system call centric analysis and stimulation technique to automatically reconstruct android malware behaviors. EuroSec.

Rosen,S. Z. Q. (2013). AppProfiler: A Flexible Method of Exposing Privacy-Related Behavior in Android Applications to End Users. ACM Conference on Data and Application Security and Privacy (CODASPY).

Sanz, B. I. S.-P. (2012). On the automatic categorization of android applications. Conference on Consumer Communications and Networking (CCNC).IEEE,in 9th IEEE, s. 149–153.

Shabtai, U. K. (2012). Andromaly: A behavioral malware detection framework for android devices. Journal of Intelligent Information Systems, vol. 38, no. 1, s. 161–190.

Strazzere, A. A. (2012). Reducing the Window of Opportunity for Android Malware: Gotta catch ’em all. (s. 61–71). Journal in Computer Virology, vol. 8, no. 1-2.

Strazzere, T. (2009). Downloading market applications without the vending app. http://www.strazzere.com/blog/2009/09/downloading-market-applications-without-the-vending-app

Weichselbaum, L. N. (2014). Andrubis: Android Malware Under the Magnifying Glass. Vienna University of Technology.

Zhauniarovich, Y. M. A. (2015). StaDynA: Addressing the Problem of Dynamic Code Updates in the Security Analysis of Android Applications. ACM Conference on Data and Application Security and Privacy (CODASPY).

Zhou, Y. a. (2012). Dissecting android malware: Characterization and evolution. (s. 95-109). Security and Privacy (SP), 2012 IEEE Symposium on IEEE.