2. LITERATURE SURVEY

As technology grows, equally vulnerability also increases. Hence the more the research focuses on technology, the greater the emphasis laid on the area of detection and prevention of related vulnerabilities. This also applies to the field of Web applications and its security. There is a rich body of literature in this area of research. Some of the works provide vulnerability detection mechanisms, while some others provide both detection and prevention mechanisms.

2.1 INPUT VALIDATION TECHNIQUES

Input validation based attacks occur due to lack of inspection or insufficient inspection on the input provided by the clients. The two most common input validation based attacks are SQL injection attacks and Cross site scripting attacks.

2.1.1 SQL Injection Attacks

SQL language being a very rich language, paves the way for a number of attacks. The first real existence of SQL injection was explored in 1998 in a magazine Phrack [10]. In this magazine an article on implementing attacks using SQL injection was explained by Rain Forest Puppy (RFP). An extensive study on the SQL injection attack and its classification was provided by Halfond et al., [7]. This work classified SQL injection attack based on the type of user input, namely Injection through cookies, injection through server variables and second order injections. The attack was also categorized based on the goal of the attacker, namely identifying injectable parameters, performing database finger-printing, determining database schema, extracting data, adding or modifying data, performing denial of service, evading detection, bypassing authentication, executing remote commands, performing privilege escalation. David Litchfield [11] on other
hand, classified SQL injection attacks into 3 types, namely in-band attack, out-of-band attack, and inference attack.

Different Mechanisms to detect the presence of SQL Injection in a Web application have been proposed by researchers. Two approaches, namely static analysis and dynamic analysis are followed for mitigating SQL injection attacks. Some techniques employ combination of both static and dynamic analysis.

**Static Analysis**

The web application’s code is analyzed statically to verify the flow of data, transfer of control, the syntax of the code, intended semantics of the program and such others. Automatic change of code has been proposed in certain works [13, 17]. The advantage of this method is that the performance of the web application server is not affected. This analysis is done as a preventive measure before it is exposed to the real time attacks. The disadvantage of this method of analysis is that the intended semantics of the program is sometimes misunderstood and developer intervention or assistance is needed in many cases.

Y.W. Huang et al., [12] developed a web crawler tool named Web Application Vulnerability and Error Scanner (WAVES). It identifies all points in web application that can be used for SQL Injection and to verify if the code contains Cross Site scripting (XSS). It uses black box testing methodology and uses machine learning approaches to guide its testing. The limitation of using machine learning techniques is that the detection rate will depend on the training data and the learned parameters. Static code checkers like the Java DataBase Connectivity – Checker (JDBC-checker) [13] is a technique for statically checking the type correctness of dynamically generated SQL queries. This will be able to detect only one type of SQL vulnerability caused by improper type checking of input. This system works on reducing the chances of the attack rather than detecting the attacks.
Y.W. Huang et al., [14] in another work implemented a system named, WebSSARI, that detects input-validation-related errors using information flow analysis. In this approach, static analysis was used to check taint flows against preconditions for sensitive functions. One technique to detect when tainted input has been used to construct an SQL query has been proposed by Livshits et al., [15] and named information flow technique. It gets the vulnerability specifications from the user and uses this as static analyzer point. This technique detects SQLIA, XSS and Hyper Text Transfer Protocol (HTTP) splitting attacks.

An approach called SQLRand based on instruction-set randomization was developed [16]. It allows developers to create queries using randomized instruction instead of normal SQL keywords. This approach specifically designed for Common Graphical Interface (CGI) application. To retain the portability and security, a reverse proxy was used. This proxy-filter intercepted the queries to the database and de-randomized the keywords.

Bravenboer et al., [17] implemented a system named StringBorg, that breaks the barriers in cross language programming, where one language needs to construct sentences in another language. This system embedded the grammar of the guest language into the host language. For example, if the guest language is SQL and the host language is Java, then the construction of SQL query can be done by Java, whenever needed. It used an Application Programming Interface (API) to build sentences, and this process is called construction. It is very helpful for the programmers without good programming experience. The limitation of this system is that it cannot detect Semantic injection attack.

Ben Smith et al., [18] in their work named Idea, identifies error message information leakage and SQL Injection attack (SQLIA) vulnerabilities during unit testing and system level testing. It was tested and proved that though SQL injection vulnerabilities are not directly present, Error message information, which could be used for implementing SQL injection
attack, were prevalent in the code. They have shown that system level testing exposed vulnerabilities whereas unit testing fails.

**Dynamic Analysis**

Dynamic analysis is done during runtime of a web application. The input given by the user is extracted and is combined with the code of the application to check if there is an intended attack. Dynamic analysis unearths most of the real time attacks at the cost of response time, and thereby affecting the performance of the server.

Gregory et al., [19] and Zhendong su et al., [20] implemented systems that checked queries at runtime to see if they conform to a model of expected queries. Taint based approaches like the Security gateway, implemented by David Scott et al., [21], is a proxy filtering system that enforces input validation rules on the data flowing to a web application to detect and prevent SQLIA and XSS Attacks.

Konstantinos Kemalis et al., [22] developed a prototype SQL injection detection system (SQL-IDS). This system monitored Java-based applications and detected SQL injection attacks in real time. The proposed detection technique was based on the assumption that injected SQL commands had differences in their structure with regard to the expected SQL commands that were built by the scripts of the web application.

Shaukat et al., [23] proposed a prevention mechanism for the SQL injection attack. It used the hash function for the authentication of the username and password. Normally, login user details are captured and then stored in the database. To authenticate a user, SQL injection protector for authentication (SQLIPA) mechanism uses the stored procedure to generate hash value for the username and password and is stored in the database. The disadvantage of the stored procedure is that the procedure seems to be unsafe which may again cause vulnerability. If a script generates an SQL query string, then it may be vulnerable [24].
Prithivi et al., [25] implemented a technique named CANDID, to realize the SQL injection attack. It uses automated byte code transformation modified Java Virtual Machine (JVM) for the java based web application. Modified JVM tracks the candidate values entered in the login form. It provides better result compared with the taint approach. In a taint approach, when a web application calls a procedure, the returned tainted values have to be kept track and external libraries would also be needed. CANDID approach maintains real input, candidate input, under the constraint of length preserved function.

Collin et al., [26] implemented the ForceHTTPs protocol, which provided protection to the web browser. It tried to rectify the error made by the web application. With the help of secure cookies, it treated error as attack. It forced all HTTP requests to move through the Hyper Text Transfer Protocol Secure (HTTPS) protocol. This in turn dealt with self signed certificate verification and used the Transport Layer Security (TLS), to make use of cryptographic encryption techniques.

Liveshits et al., [27] proposed a system named SecuriFly to dynamically track the flow of data to protect the web application against the SQL injection and Cross site scripting. It provided a compiler based solution. Vulnerability specifications were expressed as Program Query Language (PQL). It used PQL to interpret the SQL, which was then converted into Nondeterministic Finite Automata (NFA). This automaton was used to find the vulnerabilities.

Helayne et al., [28] proposed various attack models that were used by the red team to find vulnerabilities of the web application to find the loophole. Attack models mostly followed the Unified Modeling Language (UML) diagrams – use case, sequence and state chart diagrams and eXtensible Markup Language (XML) flow to find the loophole. It used the document type definition of the XML harvesting the user details.
David Scott et al., [29] defined policy rules and implemented those rules to detect the attacks. Policy was written in a flexible manner to detect all kinds of attack by hiding the essential details in the dynamic environment. Policy writing tool used the Security Policy Descriptor Language -2 (SPDL -2) language which was converted as server side program using compiler, then it was fed into the application level gateway for filtering the HTTP request.

Combined static and dynamic analysis like the Analysis and Monitoring for NEutralizing SQL Injection Attacks (AMNESIA) [30] is a model based technique that combines static analysis and runtime monitoring. In its static part, AMNESIA uses program analysis to automatically build a model of the legitimate queries that could be generated by the application. In its dynamic part, the dynamically generated queries are monitored at runtime and checked for compliance with the statically-generated model.

Kenneth et al., [31] used Deterministic Finite Automata (DFA) technique for the detection of illegitimate requests for the web application. HTTP requests were captured and fed into the tokenization process. Tokenized requests were given into the Burge DFA induction procedure, which produced the complex DFA. Previous process output size was reduced by normalization process. Similarity measurement was done between the DFA and web requests. Resultant similarity and deviation showed that changes needed to be made to the DFA to accept the request. It was used in combination with rules for reducing variability amongst requests and used heuristics for filtering. It provided high false-positive rate due to complex form of legitimate user requests.

Inyong Lee et al., [32] implemented a system that removed the value of SQL query attributes during submission. The query was then statically and dynamically analyzed to detect attacks. Dimitris Mitropoulos [33] proposed the signature based proxy driver between the application layer and database layer. The SQL queries were first extracted and these queries were checked against the stack traces for validation.
Williams et al., [34] proposed SQL Injection Vulnerability (SQLIV) detection mechanism, which created the prepared statement before the SQL statement. Attributes values of the SQL statement were replaced by the bind variables and placeholders in the prepared statement. The arguments given by the clients are matched up with the placeholders dynamically. The drawback of the system was that it might be complex to ensure that the arguments were matched with the correct placeholders, as dynamically generated queries would have number of placeholders.

2.1.2 Cross-Site Scripting Attacks

Certain works provide client side solutions [36, 37, and 39] for Cross-Site Scripting attacks where a change in the browser code or a fire wall set up is recommended. The disadvantages of these works are that client/user involvement is needed and the clients need to have technical knowledge for implementing these solutions. Peter et al., [35] implemented a system named Secure Web Application protocol (SWAP), that uses proxy layer and provides a server-side solution to detect and prevent XSS attack. It also uses a modified browser that detects script content that is included by the user. Noxes [36] is a client side solution for XSS attacks. It creates a personal firewall on the client side and checks every outgoing request and incoming response. It uses both manual and automatically generated rules to prevent XSS attacks on the client side.

Noncespaces [37] sets rules and in each document it randomizes the XML namespace prefix. These prefixes are identifiable by the user and the user will be able to identify the trusted content by the web application and untrusted content provided by the attacker. Anh Nguyen et al., [38] proposed a system that worked on PHP scripts. This system used a modified interpreter along with the PHP interpreter to identify untrusted sites. Ulfar Erlingsson et al., [39] proposed a client side solution. It was a Mutation event transform system which defined policies to be enforced in the client side browser. It defined security policies based on monitoring client behavior.
2.2 VULNERABILITY DETECTION IN SMARTPHONE APPLICATIONS

Recent researches carried out in the field of Smartphone security have concentrated mostly on Android operating system (OS). The security issues in Android are a concern owing to the fact that it is open source and developing of android application is very easy. Adrienne Porter felt et al., [40] surveyed and classified mobile malwares based on their behavior. Some of the commonly noted behaviors were, collecting user information, sending premium-rate Short messaging Service (SMS), malware written for amusement, for credential theft, search engine fraud and ransom. Different defensive mechanisms that were available against these malwares were discussed and evaluated.

Threats have been classified as physical threats, application-based threats, network-based threats, web-based threats, and mobile vulnerabilities by Jalaluddin Khan et al., [41]. They presented a survey on already existing privacy threats and defensive mechanisms. Biometric authentication was suggested as a defensive mechanism for these attacks.

Alexios Mylonas et al., [42] classified users based on their trust and awareness on web applications. They provided a database guideline for trusted repository, license and agreement, poor security checks, pirated applications and the users’ perception. Another type of attack called the android scraping was reported by Ken Munro et al., [43]. Android scraping is the unauthorized extraction of information from Smartphones. Various methods for scraping data were discussed, namely

- Cable based attacks,
  - Android Debug bridge,
  - AllWinner,
  - SpfFlash tool,
  - UART,
- attacks with screw driver,
• wiping of data that deletes Factory Access Tables (FAT)

Five major approaches are followed in detecting smartphone application vulnerabilities, namely behavioral analysis, signature based analysis, system based vulnerability analysis, permission analysis, and WebView analysis.

2.2.1 Behavioral Analysis

Jacob et al., [44] conducted a survey of different reasoning techniques used by behavioral detectors. They classified these detectors into main families, namely stimulation based detectors and formal detectors depending on their data collection and data interpretation mechanisms.

Timothy K. Buennemeyer et al., [45], in their work introduced capabilities developed for Battery-Sensing Intrusion protection system (B-SIPS) that raised an alert when an abnormal current change occurred. They developed a Correlation Intrusion Detection Engine (CIDE) that provides power profiling for mobile devices and a correlated view of B-SIPS and Snort alerts. The B-SIPS client was designed with customizable features to accommodate varying user skill levels. Users with advanced computer skills could configure the application to provide more refined detection and alert information. The basic users could effectively operate the system with default settings.

Eithan Menahem et al., [46] studied the behavior of the application during execution. They used classifiers to learn patterns to classify applications as malicious or benign. Garcia-Teodoro et al., [47] gave a detailed survey on Anomaly-based Network Intrusion Detection Systems (A-NIDS) and discussed the advantages of A-NIDS. They classified A-NIDS system based on three techniques, namely statistical, prior knowledge, and machine learning.

Asaf Shabtai et al., [48] designed a machine learning system taking into consideration various features of the system. Some of them are
messaging, battery usage, CPU usage etc. In another work [49], an assessment of the security in the android framework was done. High risk threats were identified by conducting methodological qualitative risk analysis. Yuval Shahar et al., [50] introduced a framework called Resume for the creation of abstract interval based concepts from time-stamped clinical data which was based on KBTA. This framework contained four modules external patient database, temporal fact base, temporal reasoning mechanism and domain knowledge base. It splits KBTA into five subtasks which were solved using five different temporal abstraction mechanisms.

Similar to this work, Asaf Shabtai [51] also developed a Knowledge Based Temporal Abstraction (KBTA) for a Smartphone system. This system continuously measured data (e.g., number of running processes) and events (e.g., software installation) and integrated with a temporal abstraction knowledge-base.

Asaf Shabtai et al., [52] extracted static features from executable files and used machine algorithms to detect malicious code. Various detection challenges, namely file representation patterns, feature selection methods, imbalance problem, and active learning, were addressed. In another work [53] OpCode n-gram patterns were used as features to identify unknown malicious code. These OpCode n-gram patterns were generated by disassembling the affected executable files. Document Frequency (DF) was used for feature selection method. To improve the accuracy of this framework classifiers were built on different types of features and trained accordingly.

Aubrey-Derrick Schmidt et al., [54] in their work extracted the features that described the state of the device. These features were processed in a remote server and were used for anomaly detection. Similar to this work, Paranoid Android [55] provided an architecture that did multiple security checks without overburdening the device. It took a backup of all user data in the cloud and created a replica of the device. This replica was replayed in an emulator running on a remote security server. Another system proposed by Amir Houmansadra et al., [56] also used a cloud-based Smartphone-
specific intrusion detection and response engine. The system analyzed the Smartphone for detecting change in behavior. The engine performed in-depth analysis on smartphone without considering computational and storage limitation.

Guillermo Suarez-Tangil et al., [57] presented the differences between executing anomaly detection tool in mobile platform and executing the same tool on a cloud server. The cost involved with computation, communication, data size and operation frequency was compared. The limitation was that only one cloud and one device was considered. Communication with other devices was not considered.

Ter Louw et al., [58] developed a framework for dynamically extracting the behavior of a mobile application under a controlled environment. The application’s bytecode recorded the controls and dataflow during the time of execution. This system was a part in an integrated system named Droid Reasoning, Analysis and Protection Engine (DRAPE) to discover hidden malware in Android applications.

A technique considering the dynamics of mobile phone malware that propagated by proximity contact was proposed by Gjergji Zyba et al., [59] and the potential defenses against it were evaluated. They employed analytical epidemiological model and synthetic model. For evaluating the detection and defense, the levy walk synthetic mobility model was used.

RiskRanker [60] is an automated system that analyses the mobile application to check for malicious behavior. This system proactively recognizes a zero-day android malware. The mobile application is divided into high risk, medium risk and low risk based on the degree of exploits on mobile system.

Alessandro et al., [61] implemented a approach named CopperDroid, that was designed to perform an analysis of Android system behavior. This approach invokes the system-calls and the resultant system behavior is
categorized as low-level OS specific and high-level Android specific. This observation was used as training set to train the system. CopperDroid was able to identify if the behavior of malware was initiated by a java code or a native code.

Gianlula Dini et al., [62], described the Multilevel Anomaly Detector for Android Malware (MADAM). It provided a multilevel approach that concurrently monitored android at the kernel level and user level to detect real malware infections using machine learning techniques to distinguish between standard behavior and malicious ones. This prototype was able to detect several malware found android based Smartphones.

Karim O. Elish et al., [63] designed a classification approach to detect sensitive data being leaked from Smartphones. Android applications were statically analysed to extract unique behavioural properties from benign programs and designing corresponding classification policies. Their research showed that static analysis could not detect malware when obfuscation techniques were used.

2.2.2 Signature Based Analysis

Griffin et al., [64] used signature-based method that depended on the identifying unique signatures that defined the malware. This work used string signatures, each of which was a contiguous byte sequence that potentially could match many variants of a malware family. A similar work had been proposed by Yu Feng et al., [65]. A system named Apposcopy, which used a semantics-based approach to identify a prevalent class of Android malware that stole private information, was developed. It used a high level language for specifying signatures of malware families and performed static analysis to decide if an application was benign or malicious.

G.Suarez-Tangil et al., [66] introduced a system called dendroid based on text mining and information retrieval. This system automatically classified malicious applications as families based on common code structure. For
family classification this system used vector space model. This also provided relationship among families and their evolution from common ancestors. If an unknown application was installed, the system would automatically identify its family and assign this application to that family. The limitation of these systems was that it can be applied only to known malicious code.

A malicious code could be transformed to appear like an anti-malware product. This was addressed by Vaibhav Rastogi et al., [67]. In their work, they illustrated that transformations of malware were easy to perform and execute. A tool named Droid Chameleon was developed to test anti-malware products for transformed malicious code.

ScanDroid [68] did a modular analysis to allow incremental checking of applications as they were installed on an Android device. It extracted security specifications from manifests that accompanied such applications and checked whether data flowing through those applications were consistent with those specifications. Another framework that was a client-side automated and that was capable of performing automated vulnerability test on android smart phones was proposed by Rafal Fedler et al., [69]. A configuration file was put up in the remote server. This file contained information on currently available exploits and their respective payloads. During start up this configuration file was fetched and information on the exploit to be executed was extracted. Following this, the payload to be executed was also extracted from the server.

2.2.3 System Based Analysis

Y. Park et al., [70] studied the attacks caused by illegally escalated privilege to the system level. In android systems root level privileges should be restrictively used. If root privilege was acquired the attacker could do anything with the system. The authors introduced an extension called Rooting Good-Bye on Droid (RGBdroid). RGBdroid was not a prevention approach. The system reacted to an attack, which has already occurred. It used the principle of least privilege to protect the resources.
Asaf Shabtai et al., [71] in their work, implemented SELinux in Android in order to harden the Android system and to enforce low-level access control on critical Linux processes that ran under privileged users. By choosing this route, the system could be protected from exploits caused by vulnerability in one of the high privileged processes.

Mauro Conti et al., [72] proposed crepe system for the android devices. It allowed users and other predefined trusted parties to define context-related policies, which could be installed, updated at runtime. These policies could be applied in a fine-grained manner, that is, for each application. The lack of the possibility to regulate the behavior of Smartphones, based on their context, made it difficult to adopt this technology to its full potential.

Nai-Wei Lo et al., [73] designed a new tool LRPdroid which detected the leakage of personal information. The user perception on the personal information and their ranking score on a scale of 0 to 5 was received from the user and pre-processed. The app execution flow, user perception setting and leakage awareness detection, the information leakage and privacy assessments are measured. If leakage was found, it warned the user. The focus is on the completeness and correctness of the data leaked. This was marked as sensitive and non-sensitive.

Shih-Hao Hung et al., [74] provided a framework named as pasDroid to protect the system against the leakage of private data like location, microphone, camera, microphone, databases(SMS, storage address book), id-data (IMEI, phone number, SIM). These were marked as tainted devices and flags were assigned. PasDroid traced the suspicious traffic flow of data and notified the user on the leakage of the private data on the fly. The system was designed to eliminate false alarms. The limitation was that it could cause a serious strain on the CPU of the system.

Abhijit Bose et al., [75] proposed an approach for automated and proactive security model. Automated identification algorithm was used to
identify an attack from the vulnerable clients. Proactive containment framework described two common mechanisms whenever a worm or virus was suspected, namely rate-limiting and quarantine. The evaluation was done by dynamically generating vulnerable clients into messaging networks. SMS customer network was used to evaluate the effectiveness of the proactive security.

Iker Burguera et al., [76] proposed a new framework for smartphone application activity. A detector was embedded within the overall framework and it traced the malware from crowdsourcing. Central servers had two types of datasets namely artificial malware and real malware. This method was used to effectively isolate the malware and alerted the users when a malware was downloaded. This way, it avoided the spreading of a detected malware to a large community.

Machigar Ongtang et al., [77] proposed a system named Secure Application INTeraction (SAINT) framework, that provided a modification in the infrastructure of the android system. This modifications administered the dynamic permission assignment during installation time. The system was augmented with semantics application policy that controlled the applications run-time use. The drawback of this system was that due to install-time permission assignment certain functionalities of a benign applications could become void.

Takamasa Isohara et al., [78] proposed a kernel based behavior in android malware. Malware attempts to leak the personal details in order to get the root privilege. Audit framework logcat was introduced to monitor the application behaviors, which was used only for debugging. This framework audited the application activities and security inspections. Kernel based behaviors were classified into two types, namely log collector and log analyzer. Log collector was used to record the calls, events. Log analyzer matched the malicious activities, thereby initiating necessary actions.
2.2.4 Permission Based Analysis

Muneer Ahmad Dar et al., [79], in their work compared the security features of Android and iOS, with the intention to implement the Need Based Security (NBS) model in Android which selectively granted permission to access resources on a Smartphone at run time. This was done by reverse engineering the application and imposing changes in the code manually to alter the permissions. The application was then repackaged and could be run on any android smart phones. The possibility was that some parts of the code might not function properly which could lead to the crashing of the application.

An empirical analysis of permission based security models was proposed by David Barerra et al., [80]. This system made novel use of Self Organizing Map (SOM). A detailed analysis of applications and their permissions was made. The strengths of the permission model and its shortcomings were also analyzed.

An automatic extraction tool was proposed by Dong-uk Kim et al., [81] that provided a solution for the problem of resource limitation using machine learning on android market and the problem of static detection based on permission by performing static detection methods on API's.

Another tool TaintDroid [82] performed dynamic taint analysis. It tracked the real-time flow of sensitive data through applications to detect inappropriate sharing. This system gave the adequate control over the sensitive data and information on how the third party applications used their sensitive data. This system provided real-time analysis by utilizing android virtualized execution environment. The taint source was Application Program Interface (API) data, and the network was the sink. They tracked only data flow, but not control flow. Complementary to TaintDroid is Inter-application Communication (IPC) Inspection implemented by Adrienne porter et al., [83], that tracks API return values but does not prevent API calls from being made. Details about permission redelegation and its risks were discussed. A new
OS mechanism for defending against permission re-delegation was presented.

Kirin Security services implemented by William et al., [84] used a variant of security requirements and engineering techniques to perform an in-depth security analysis of Android to produce a set of rules that matched malware characteristics. An analysis was made to unearth certain permission combinations that might be viably malicious. A similar approach was also proposed by Veelasha Moonsamy et al., [85]. A mining tool that mined patterns on permissions of both clean and malicious android applications was designed. Benchmarking technique was used to find the popularity of a particular permission among the clean and malicious applications. Support based techniques and apriori based techniques were employed to check for common permissions and unique permissions among each category. To distinguish between the requested and used permissions and requested but not used permissions for each application, contrast permission mining was used. The advantage of this system was that not only the permissions but the co-occurrence of certain permissions were considered for categorizing benign and malicious applications.

Another work that explored the co-occurrence of permission was proposed by Wei Wang et al., [86]. The work was two fold. First the risk caused by individual permission was considered and then risk caused by combination of permissions was exploited. Feature ranking methods, namely mutual information, correlation coefficient, and T-test were used to assign rank to individual permission. To identify the subset of risky permission they used sequential forward selection and principal component analysis.

Abijith Bose et al., [87] represented the behavior of malwares based on key observations. Using these observations they proposed a framework to detect mobile worm, virus and trojans. In this work, a two stage mapping technique was used to construct behavior signature at run-time by using API calls in symbian OS. Support Vector Machine (SVM) was used to classify malicious behavior and normal behavior. The accuracy rate of detection was
found to be 96%. This work also defined various rules to identify the inter-process relationships. A similar system which worked on identifying the inter-process relationship has been proposed by Adrienne Porter Felt et al., [88]. A software tool was developed to determine over privileges in compiled android applications. This tool determined these API calls of an application and mapped the API calls to permissions.

Abdulah K. Talha et al., [89] proposed a system called Android Application Package (APK) Auditor which was permission based malware detection system. It consisted of a central server, a signature database, and an android client. The user submits a request to analyze an APK using the android client interface. Signature database contains information about application and analysis result. Central server manages the whole process besides communicating between client and signature database. Accuracy of this system was found to be 88%.

Yuan Zhang et al., [90] developed a tool called Vetdroid. The tool analyzed the permission requested by an application and the resources requested by the application were segregated. It presented to the user, a list of sensitive system resources that had been requested for access by the application. Further, utilization of these acquired permission-sensitive resources was also analyzed. It helped in detecting information leaks and the cause of the information leak.

Junho Choi et al., [91] proposed a proactive inference based security access control model. This system used context ontology reasoning of permission access and API resource information. The application was categorized under one of the 25 categories using the category authorization service. This was then analyzed using the context sensor and the authentication and security issues were checked by the access control manager. The inference engine was designed to manage policies for each category and ontology status stored the analyzed status authorization details. This system detected existing malicious code and new unknown attacks.
Kathy Wain Yee Au et al., [92], implemented a system named Permission Scout (PScout). It is a static analysis tool that analyzed the code of an application to identify the permission required to execute their API calls. This system provided a mapping between the API's and their corresponding permissions. The drawback was that, the system did not check if API call was malicious or benign and if it was relevant for the application.

Sven Bugiel et al.,[93] described eXtended Monitoring on Android (XmanDroid), a security framework that extended the monitoring mechanism of Android to detect and prevent application level privilege escalation attacks at runtime. This was based on a system centric policy. This system dynamically analyzed application’s transitive permission usage, while inducing a minimal performance overhead that was unnoticeable by the user. This system simulated the known application level privilege escalation attacks and demonstrated successful detection of those attacks.

Hao Hao et al., [94] presented the effectiveness of bytecode rewriting for API level access control. They tried to provide modified behaviors that allowed fine grained access control at Java API level. The security sensitive APIs were identified in the code. These codes were modified by adding customized access control logic before they were being invoked during runtime. These attackers could bypass the API access control on it and it was difficult to implement.

2.2.5 Webview Based Vulnerabilities

Qing Li et al., [95] in their work investigated the potential threats that could affect the mobile phones security. The risk factors involved in smartphone platforms were discussed. The iOS and Android mobile OS follow certain technique to separate the applications based on privileges. That is, the underlying kernel isolates each application into an execution sandbox that shields the application from unauthorized access to its data. An application makes access requests to gain the necessary permissions to protected resources—these permissions are granted either by the kernel or
explicitly by the user. It is also viewed that the web-based applications on both iOS and Android have all the vulnerabilities found in their desktop counterparts. Moreover, it is also observed that the WebView technology adopted by both the iOS and Android systems to embed browser functionalities inside the applications can be exploited to tamper the protection mechanisms in the underlying mobile Operating system.

Tongbo Luo et al., [96], in their work based on exploitation of WebView API’s, classified the attacks as web-based and user-interface based. Web-based attacks were classified into two categories, namely, attacks from Web pages and attacks from applications. The attacks such as Excess-authorization attack, JavaScript Injection, Event sniffing and Hijacking, Frame confusion attacks and cross-site scripting attacks were explained. Tongbo Luo et al., [97], in another work discussed a different type of exploit named touch jacking attacks. Various forms of the attack such as WebView redressing attack, Invisible WebView/Transparent WebView hijacking attack, Key stroke hijacking attack and Event simulation attack were also discussed.

Erika Chin et al., [98] developed a tool, Bifocals for detecting the vulnerabilities inflicted to the Smartphones. It characterized the prevalence of vulnerable code in smartphone applications. The tool could explore two types of web view vulnerabilities, namely excess authorization attack, which invoked a malicious javascript into an Android application code and file based Cross-Zone Scripting attack, which was used to expose a devices’ file system to an attacker.

A mechanism for launching cross-site scripting attacks without using core WebView APIs was proposed by Bhavani [99]. HTTP Client APIs were used to load malicious scripts dynamically from remote servers for launching cross-site scripting attacks. The session hijacking attack was implemented by stealing cookies information. Accessing and extracting sensitive information like contacts stored on device using HTTP GET and HTTP POST APIs was also addressed in this work.
Jing Yu et al., [100], in their work discussed the various Javascript attacks carried out through WebView. They provided a solution for mitigating the attack dynamically by performing an access control on the certain APIs that were responsible for the security of the contents of the device.

Matthias Neugschwandtner et al., [101] gave an overview of the WebView targeted threats. These threats were distinguished as the server compromising attacks, like the SQL injection attacks, Cross-Site scripting attacks and the traffic compromising attacks, like the man-in-the-middle attack, HTML attacks, Javascript attacks.

Veelasha Moonsamy et al., [102], in their work claimed that 10% of the applications in their training dataset extracted information from the smartphone using third party advertising libraries. A leaky application was defined to be a vulnerable application that was used by a third party to extract information. WebView was found to be one of the vulnerable applications. A study on how the leak occurred and the destination of the leaked data was carried out in their work.

Leonid Batyuk et al., [103], in their work targeted the problem of detection and mitigation of unwanted activities within Android applications. Their contribution was twofold. Firstly, a static analysis service was described, which gave a detailed study of the application’s internals and was comprehensible for an end-user. Secondly, a strategy for monitoring unwanted activities was presented.

A.D. Schmidt et al., [104] presented an analysis of security mechanism in Android Smartphones with a focus on Linux. These results could be applicable to android as well as Linux-based Smartphones. Android framework and the Linux kernel were analyzed and the security functionalities were compared. Well accepted security mechanisms and tools which could increase device security were surveyed. Details on adoption of these security tools on Android platform and overhead analysis of techniques in terms of resource usage were illustrated. Their second contribution
focused on malware detection techniques at the kernel level. The applicability of existing signature and intrusion detection methods in android platform were tested. Observations on the kernel, like identifying critical kernel event, log file, file system, network activity events and making efficient mechanisms to monitor them in a resource restricted setting were made. Using these observations a simple decision tree for deciding the suspiciousness of the application was constructed.

2.3 SUMMARY

The static analysis to mitigate input validation attacks such as JDBC-Checker [13], WebSSARI [14] needs extensive sophisticated full code analysis to reduce the number of false positives. Techniques such as the AMNESIA [30], SQLCheck [20], SQLGaurd [19] will not be able to detect stored procedure type of attacks as they use parse trees for static and dynamic query analysis. The complexity of the algorithms of these system is $O(n^3)$ which makes it slower in real time. This thesis extracts the user provided data from the query and then the query is compared with a preprocessed standard query which makes the system simple and uncomplicated to execute.

Certain systems demand automatic change in source code [16, 17, 25, and 36] which could change the intent of the program. The studies employing only machine learning techniques require a large dataset to train the system and the trained system will respond effectively only if the real world test data lies well within the trained range of values. The literature discusses techniques to detect if an installed Smartphone application is malicious, but does not address a benign application turning malicious after installation and updation. In permission based analysis, the literature considers system defined permissions. The developer defined permissions are not discussed.