Applied Information Technology And Computer Science Vol. 4 No. 2 (2023) 103-119

AITCS

Homepage: http://publisher.uthm.edu.my/periodicals/index.php/aitcs
e-ISSN :2773-5141

AnTScan: Android Application Testing and Scanning

Ariff Irfan Abdul Rahman[1], Mohd Azahari Mohd Yusof[1]*

[1]Fakulti Sains Komputer dan Teknologi Maklumat,
Universiti Tun Hussein Onn Malaysia, Parit Raja, Batu Pahat, 86400, MALAYSIA

DOI: https://doi.org/10.30880/aitcs.2023.04.02.007
Received 05 November 2023; Accepted 06 November 2023; Available online 30 November 2023

Abstract: Vulnerability assessment, also known as vulnerability scanning, is a testing process used to identify security flaws and vulnerabilities in computer systems, networks, applications, and infrastructure. It is also a procedure for determining the severity level of security vulnerabilities. This paper was produced to propose a vulnerability scanning tool that currently has limitations such as taking longer to complete the analysis process, categorizing vulnerabilities, and using command line interface. The objective of this project is to develop a vulnerability scanning tool capable of identifying vulnerabilities in an Android application as well as information about the application. The methodology of this project is object-oriented methodology which helps in code organization and modularity. This tool is being created using Python language. The tool is being developed based on prototyping model. The outcome for this project is the development of a vulnerability scanner capable of identifying vulnerabilities in a shortened time and reducing the number of vulnerabilities in an Android application.

Keywords: Penetration Testing, Vulnerability, Android Application

1. Introduction

Vulnerability assessment, also referred to as vulnerability scanning, is a recognized procedure for locating security flaws or vulnerabilities in computer networks, applications, infrastructure, or systems. It entails thoroughly examining the systems to find any known or potential vulnerabilities that attackers might exploit [12]. A vulnerability assessment's goal is to proactively find and fix security flaws before they can be exploited. Tools for vulnerability assessment are used to search for known vulnerabilities in applications, systems, and networks. To find potential weaknesses, these tools use a variety of techniques, including port scanning, service detection, vulnerability identification, and configuration checking. The assessment offers a thorough report outlining the discovered vulnerabilities, their seriousness, and suggested corrective actions. Nowadays, the development of new Android applications has been increasing through the years. The requirement for vulnerabilities scanning tools is increasing to identify vulnerabilities that exist in the Android application.

*Corresponding author: mdazahari@uthm.edu.my

Vulnerability assessment is a part of the pentesting process and is included in reconnaissance phase which is information gathering [13]. Currently, existing tools require longer time for analyzing and identifying vulnerabilities. According to puprplesec.us the duration of the scanning process might be influenced by the number of vulnerabilities found and the complexity of application. Thus, an automated scanning process capable of reducing time for the analysis process. It can identify the vulnerability on their own when the process is executed. The categorization of vulnerabilities is needed to help developer or pentester define the risk and action that needs to be taken. There are some tools that can perform analysis on android applications that do not have Graphical User Interface (GUI) such as AndroGuard. GUI can be help for the user to navigate through the tool.

There are few existing tools that have been selected to study the process of current vulnerability assessment tools work. The existing tools are also used as reference to design a new vulnerability assessment tool that can overcome the issues on the existing tools. The prototype model has been selected as the methodology of this project to ensure the flow of the development was on the right track. The proposed tool can identify the vulnerabilities during the scanning process and it will categorize the vulnerabilities based on the risk level. Then generate a report from the whole process of testing.

The rest of the paper is structured as follows: The literature review of the relevant literature and current application is covered in Section 2 of this article. Section 3 of this article provides a description of the method used to create the tool. In section 4 of this article will be discussed about the design and analysis of the tool. Section 5 will discuss the results of the tool, while section 6 will discuss the conclusions and future work of the project.

2. Related Work

This section discusses vulnerability assessment, anroid operating system, vulnerabilities of mobile applications and existing penetration testing tools.

2.1 Vulnerability Assessment

Vulnerability assessment, also known as vulnerability scanning, is a method for identifying security holes in computer systems and networks (Penetration Testing Tool for Vulnerability Assessment, 2022). The purpose of a vulnerability assessment is to not only find all possible vulnerabilities but also to assist the developer in fixing them. A network or system's vulnerabilities can be found, evaluated, and ranked through a process known as vulnerability assessment. In other words, vulnerability assessment is a tried-and-true method for assessing a system's security by locating vulnerabilities in a computer program or other application [9].

Vulnerability assessment usually has two techniques of analysis process which are static analysis and dynamic analysis. Both are two different approaches used in testing and analysis. Static analysis is a technique that analysis the code without executing it [6]. This technique will examine the code's structure, syntax, and other characteristics to identify potential issues, vulnerabilities, or violations of coding standards and best practices. It can be performed using pattern-based and flow-based techniques. Static analysis also can identify vulnerabilities in large number of applications in reasonable time [3]. Dynamic analysis is helpful for assessing performance, testing scaling and load balancing techniques, and finding problems that static analysis alone would miss. Dynamic analysis can uncover subtle flaws, vulnerabilities, and defects that may not be detectable through static analysis [11]. It focuses on the actual execution of the code and can provide insights into the behavior of the system under different conditions.

2.2 Android Operating System

Android OS is an open-source mobile operating system that is built on kernel Linux. The Android mobile operating system was created by Google and is primarily intended for touchscreen mobile

devices. It is based on a modified version of the Linux kernel and other open-source applications [15]. Android OS also provides basic operating systems, an application middleware layer, java software development kit (SDK) and collection of system applications [16]. Android operating system architecture consists of Linux kernel, Hardware abstraction layer, Native C/C++ libraries, Android runtime, Application Framework, application. Kernel provides drivers for hardware, networking, filesystem access, and process management. The Linux kernel used by Android is slightly different from "normal" Linux kernels used by desktop computers and non-Android embedded devices. The new features, which include low memory killer, wakelocks, anonymous shared memory (ashmem), alarms, paranoid networking, and Binder, are what set them apart.

Hardware abstraction layer (HAL) offers interfaces that provide the higher-level Java API framework access to the hardware features of the device, such as the camera, GPS, or Bluetooth module [2]. The libraries or native libraries is an open-source Web browser engine Web kit, well known library libc, SQLite database which is a useful repository for storage and sharing of application data, libraries to play and record audio and video, SSL libraries responsible for Internet security and other serving specific purposes [4]. Android runtime is in the third section of the architecture and available on the third layer from bottom. The Dalvik Virtual Machine, a type of Java Virtual Machine specifically created and optimized for Android, is a crucial element provided in this section [15] but Dalvik Virtual Machine is being replaced by Android Runtime Virtual Machine (ART VM) which it is equivalent of a Java Virtual Machine (JVM). Lastly, Applications (or apps), which are the programs that users directly interact with, are at the top of the stack. We distinguish between system applications and user-installed apps even though all apps have the same structure and are developed on top of the Android foundation.

2.3 Vulnerabilities in Mobile Application

A vulnerability is a flaw that may be used by hackers to get into a system without authorization [7]. Cyberattack can run malicious codes, install malware, and steal sensitive data by exploiting vulnerabilities.

2.3.1 Cryptographic Failure

Cryptographic failures include both poor encryption implementation and total absence of encryption. A cryptography failure might possibly disclose sensitive data, which is its main effect. Depending on the information that is not sufficiently safeguarded by encryption, the disclosure of sensitive data might result in compliance, reputational, or competitive business problems [6]. A cryptography failure can have effects beyond only stealing data from or from the user. A comprehensive database including hundreds of sensitive records, data theft, public disclosure, breaches, and several serious issues with business-related data can be obtained by attackers [4].

2.3.2 Injection

Injection is a risk category that describes threat actors' capacity to provide web apps with malicious input that causes the program to execute unforeseen and unwelcome instructions. When an app can't tell the difference between malicious input and its code, injection happens. Common injection threats include SQL injections, which load malicious SQL queries into the client-side of the web app, and JavaScript injections, which load malicious code into input areas. Typically, the threat agent provides data in the form of harmful code to the mobile app through a variety of channels [1]. Attackers often insert harmful code using input data. If this code is performed with privileged rights, it might compromise the system.

2.3.4 Hard Coded

Hard coding, also known as hard-coding or hardcoding, refers to the practice of embedding data directly into the source code of a program or executable object. Instead of obtaining data from external sources or generating it at runtime, hard-coded data is directly inserted into the code. This data is typically unchanging, such as physical constants, version numbers, or static text elements. Hard-coded data is often used for values that are fixed and do not need to be modified frequently. Hard coding can introduce vulnerabilities in software applications, especially when it involves the use of sensitive information like passwords or credentials. Hard coding passwords or credentials directly into the source code makes them easily discoverable by anyone with access to the code [10]. This can include developers, testers, or even attackers who gain unauthorized access to the codebase. If the code is accidentally published or shared publicly, the plaintext passwords become accessible to anyone, compromising the security of the system. Other than that, hard-coded credentials often remain unchanged for a long time, as they require manual modification of the source code and recompilation of the application. This lack of flexibility makes it challenging to rotate or update passwords regularly, increasing the risk of unauthorized access if the credentials are compromised.

2.4 Existing Penetration Testing Tools

Metasploit is a framework that both unethical and criminal hackers can use to look for systemic holes in servers and networks. It is an open-source framework that is adaptable and compatible with most operating systems [17]. One of the more well-known ruby language-based online penetration frameworks, it targets the services and applications of many vulnerable hosts and now offers 800 attack scripts [18]. It also has built-in modules for creating unique attack programs. After being bought by Rapid7 in 2009, the Metasploit framework quickly became well-known as a cutting-edge information security tool for assessing the susceptibility of computer systems. Metasploit 4.0, which was published in August 2011, provides tools to uncover software vulnerabilities in addition to exploits for known bugs. Metasploit has a variety of techniques on vulnerability scanning, one of them is port scanning. Nmap has been built in with Metasploit to allow user to do the scanning through all the port and gain the information regarding the port and potential vulnerabilities within it. Another method by using Nessus and NeXpose, both tools are vulnerability scanning tools to identify vulnerabilities on a web server [14].

Burp suite is a pen testing tool that is utilized by experts today as one of their main tools that enables users to complete challenging and unique work [18]. The ability to log, intercept, manipulate, and display the Hypertext Transfer Protocol (HTTP) or Hypertext Transfer Protocol Secure (HTTPS) is only one of the many features that make Burp Suite one of the most popular tools. The burp suite includes several utilities, including proxy, repeater, intruder, decoder, comparer, and extender. By intercepting proxy traffic, proxy enables the penetration tester to set up the traffic to go through Burp [18]. Burp Suite is one of the closed-source applications where some functions require a fee, however this project will employ an open-source application where users will have complete access to all the capabilities. Automatic vulnerability scanning is a feature of burp suites that speed up the scanning process because machines are quicker than humans. Extension for Processing and Recognition of Single Sign-On Protocols (EsPReSSO), an open-source Burp Suite plugin, was created to automate the manual analysis of Single Sign-On (SSO). The goal was to address the problems that arise when inexperienced penetration testers carry out the same repetitive task to identify the SSO protocol. Popular SSO protocols like OpenID, Mozilla BrowserID, Security Assertion Markup Language (SAML), Microsoft Account, OAuth, Facebook Connect, and Microsoft Account were all compatible with EsPReSSO.

A free android app scanner called QARK (Quick Android Review Kit) can be used to look for security flaws. This software can list security flaws and extract an apk file's source code. This tool can undoubtedly provide some important insights and help with issues for Android mobile apk, even though it isn't comprehensive [19]. QARK is made to scan source code or packaged APKs for various security-

related vulnerabilities in Android applications. It also performs source code analysis to identify useful information about the target. Qark is available in two modes: interactive mode and seamless mode. Interactive mode allows users to select options one by one, whereas seamless mode allows the pentester to do all tasks in a single command. Qark is simple to use, and it can find a lot of vulnerabilities in Android applications. On the other hand, it may generate ADB commands or fully functional APKs that can be used to "proof of concept" exploits the vulnerabilities. Furthermore, it can provide reports to serve as help for developers and pentesters in determining which vulnerabilities must be addressed to be secure.

Table 1: Comparison with the existing system

Features/System	Metasploit	Burp Suite	Qark	Proposed Tool
Open Source	√	X	√	√
Automatic Vulnerability Scanner	√	√	√	√
GUI	X	√	X	√
Vulnerability categorization	X	X	X	√
Mobile Support	√	√	√	√

Based on Table 1, it can be concluded that the three existing tools have different weaknesses that need to be overcome. Therefore, a new complete pen test tool will be designed. The table shows that the three existing tools do not have features that can categorize the vulnerabilities as high risk, low risk or medium risk. Thus, it can be one of the unique features in the proposed tools. The proposed tool requires the user to upload an application file or well known as APK file to the proposed tool. Automated scanning will be initiated after the file has been uploaded, it will perform some combination of analysis to get details about the vulnerability and vulnerability that is hidden in each file.

3. Methodology/Framework

The proposed tools for this project are being developed under the guidance of a prototyping model. The prototyping model is a tool development approach in which a prototype is produced, tested, and then tweaked until an acceptable output is obtained from which the entire system or product can be developed. This process enables developers to find any missing functionality as well as problems. Based on the client's response, the prototype is evaluated and refined, and the process is repeated until a final prototype with all of the required functionalities is generated. The five basic stages of the prototype technique are planning, analysis, design, prototype, and implementation. The finalized prototype will also serve as the model for the product's final design. Figure 1 shows the prototyping methodology model.

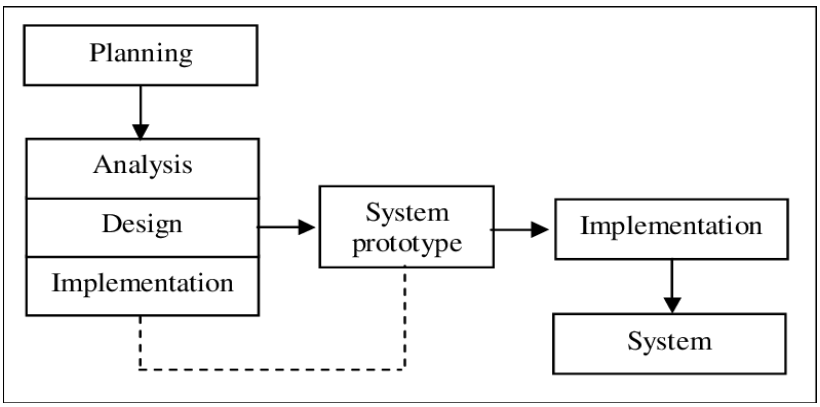


Figure 1: Prototype Model Phases

3.1 Planning

The planning phase is the first step in the development of the Prototype model. This phase is concerned with developing the project's concept. Problems, objectives, scope, and expected results are identified during this phase. This step will also generate a Gantt chart to ensure that each work is completed on time. This will ensure the project's overall success, the timeline's adherence, and the task's completion within the allocated amount of time many days for each stage. By doing this, the project will continue to move in the proper path.

3.2 Analysis

The analysis phase's objective is to thoroughly study and evaluate the viable options identified in the feasibility phase to identify the optimal high-level solution that will satisfy the needs of the client and the project's constraints. The data that is gathered includes user requirements, tool requirements, material source references, and software requirements. A literature review was carried out at this phase to serve as a guide for the creation of the tools. To spot any characteristics that are missing from the suggested tool, existing tools are also contrasted. Additionally, by creating a comparison table with the suggested tools, the advantages and disadvantages of the current tools are thoroughly reviewed.

Functional requirements specify how a system must operate. It outlines the steps that must be taken by the system to meet user demands or expectations. NFRs, also known as non-functional requirements, are a set of specifications that describe the operational capabilities and restrictions of the system while aiming to increase its functionality (AltexSoft, 2019). Table 2 and Table 3 show the functional and nonfunctional requirements that are needed in this project.

Table 2: Functional Requirement

No	Module	Functional Requirement
1.	Upload Module	This module allows the user to upload the APK file from the devices into the tool.
2.	Scanner Module	This module will be used to start the vulnerabilities scanning to identify the vulnerabilities that occur in the android application.
3.	Vulnerability Module	This module will show the list vulnerability that have been identified during scanning process.
4.	Report Module	Report modules are used to generate a report for all the data and status of the analysis and process after all the processes are complete.

Table 3: Non-functional requirements for the project

No	Requirement	Explanation
1.	Performance	The proposed tools can operate at the best performance without having any failure.
2.	Scalability	The tools can scan the vulnerabilities and exploit it to ensure the application is able to withstand cybersecurity attack.
3.	Compatibility	The tool must support android applications that are running on different versions of android operating system.

3.3 Design

During the Design Phase, the system is created to fulfil the requirements identified in the prior stages. The requirements found in the Requirements Analysis Phase are used to build a System Design Document that accurately reflects the design of the system and may be used as an input to system development in the subsequent phase. One of the jobs that will be accomplished in this phase is designing the necessary interfaces. Through the process of analysis, a logical plan is converted into a physical execution blueprint. During this stage, the tools' wireframe, which serves as guidelines, is designed. Wireframe creation for this project will be done using draw.io. One of the online tools for creating wireframes, database designs, and UML diagram is Draw.io.

The use case diagram in Figure 32 shows the interaction between the pentester and the proposed tool. The pentester or user can upload the APK file into the proposed tools. The uploaded application will be scanned after the user triggers the button to start the analysis process. During the scanning process it will identify all the vulnerabilities that are contained in the APK file including gain some information related with the uploaded file. When the process is complete the user can view the vulnerabilities that have been identified and the user or pentester allow user to generate report and download the file.

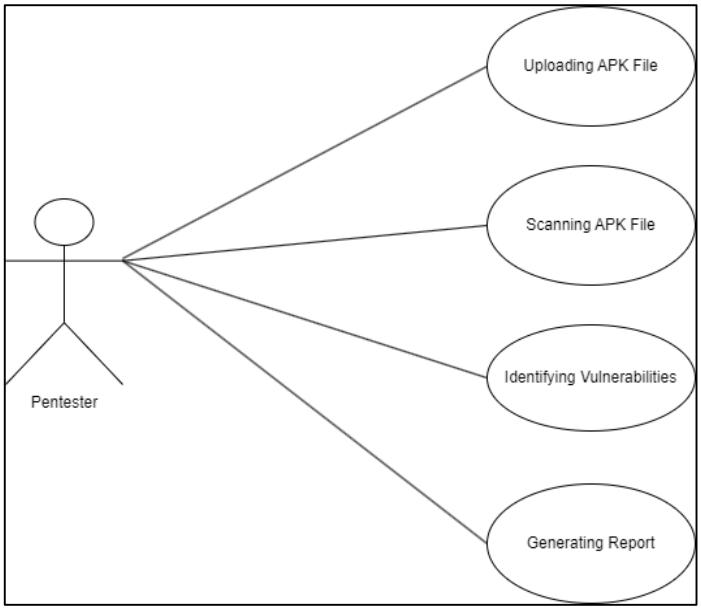


Figure 2: Use Case Diagram

Figure 4 shows the interface for vulnerability scanner as a sample of the interface proposed tool. In this Figure are examples where all the information related to application will be shown in this interface.

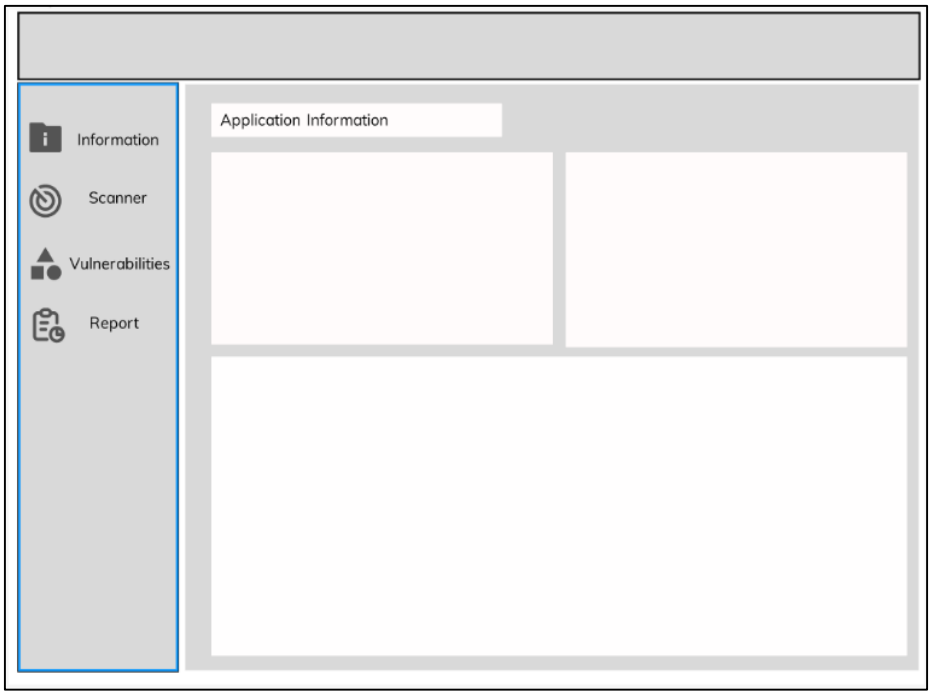


Figure 3: Interface for Proposed Tool

3.4 Implementation

The construction or implementation of a project is referred to as the "implementation phase." The "implementation phase" is typically used to refer to the testing, inspecting, adjusting, correcting, and certifying of facilities and systems to ensure that the project operates as intended. Python will be the primary programming language used in the proposed tools. The Python programming language is frequently used for creating software and websites, automating repetitive tasks, and analyzing and displaying data. Thousands of libraries and frameworks are also available for use. Metasploit framework will be used in the implementation of the proposed tools. Metasploit framework is an open-source framework that can be downloaded free on the internet. The Metasploit Framework is a framework for vulnerability assessment that offers a payload generator, a database of vulnerabilities and exploits, and other helpful modules.

3.5 Testing

Every line of code is written and run during testing. Delivering trustworthy software can be challenging without sufficient testing. The proposed tool will also be tested after completion to check if it satisfies the functional and non-functional requirements identified during the analysis stage. In this stage, the proposed tool will test all the functionality. It also tests by using a few APK files to ensure all the output can be achieved.

3.6 System Development Workflow

There are total of five phases from the prototype model. As shown in Table 4, each phase has its own assignment and output that is needed to produce during the entire project development. Besides that, the output was completed within the specific days that have been given.

Table 4: Software development activities and their task

Phase	Task	Output
Planning	● Proposed the project. ● Determine the project schedule, activities, and output. ● Identifying problem statement, scope, and objective	● Project proposal ● Develop Gantt chart.
Analysis	● Analyze the existing system. ● Choose the best methodology. ● Collect and analyze information. ● Choose the best methodology.	● Literature review ● Requirement Definition ● Hardware and software specification ● Comparison table between proposed tool and existing tools.
Design	● Design wireframes and prototypes. ● Design the interfaces	● Wireframe UI ● Prototype tool ● Architecture of tool
Implementation	● Code the tool ● Implement the code to the interface. ● Implement the selected framework.	● Prototype functional tool
Testing	● Tool testing functionality ● Identify flaws and bugs	● Report flaws or in the tool ● Improve tool.

4. Results and Discussion

In this section, the screen captured of the tool and code segment including the result will be shown. The implementation and the result of the proposed tool will be discussed in this section.

4.1 Implementation and Result

The code segment in Figure 4 represents the uploading process for the proposed tool. In this code segment, when the user or pentester trigger the upload button. When the file is being uploaded it will validate the file whether the file is APK file or not. If not, you are required to upload other APK files. If the file is APK continue the process and store information about the file in the database and there will be folder name upload that store all the uploaded files. After that it will past the file to the other function that.

```
@app.route('/upload_apk', methods=['GET', 'POST'])
def uploaded_file():
    if request.method == 'POST':
        # Check if the post request has the file part
            f = request.files.get('filedata')
            if f:
                filename = secure_filename(f.filename)
                #verify is it apk file
                if filename.split('.')[-1] == 'apk':
                    f.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
                    # Store the file path in the session
                    session['file_path'] = os.path.join(app.config['UPLOAD_FOLDER'], filename)
                    #call the database
                    db = get_db()
                    cursor = db.cursor()
                    #input the query statement
                    query ="INSERT INTO apkfile (apk_name, apk_date, apk_path, apk_size) VALUES (%s, %s, %s, %s)"
                    now = datetime.datetime.now()
                    file_size = os.path.getsize(os.path.join(app.config['UPLOAD_FOLDER'], filename))
                    data = (filename, now, os.path.join(app.config['UPLOAD_FOLDER'],filename), file_size)
                    cursor.execute(query, data)
                    db.commit()

                    return redirect(url_for('info_apps_result', filename=filename))
                else:
                    flash('That was not apk file')
                    return redirect(url_for('upload'))
```

Figure 4: Code Segment for Uploading File

Figure 5 shows the HTML code segment where the file is being uploaded. When the file is uploaded, the action attribute in the form element will identify the route called "/upload_apk" where the route contains the function for uploading the APK file. The accept attribute in the file <input> type, sort out all the files that only have '.apk' which eases the user to find the APK file in their folder.

```
    </div>
    <div class="container d-flex justify-content-center align-items-center p-5 text-center bg-light">
        <div class="card">
          <h3>Upload Files</h3>
          <div class="drop_box">
            <header>
              <h4>Select Your Apk file here</h4>
            </header>
            <form method="POST" enctype="multipart/form-data" action="/upload_apk">
                <input type="file" name="filedata" accept=".apk" id="fileID">
                <button class="btn" type="submit">Upload</button>
            </form>
          </div>

        </div>
      </div>

    <div id="successMessage" style="display: none;"></div>
```

Figure 5: HTML Code segment for Uploading

Figure 6 shows the code segment for extracting information from the APK file. The extracted information will be stored into the database, and it will be stored temporarily in a session function. As shown in figure 6, information such as application name, package, version, and encryption are being extracted. This file is being extracted by using a library named AndroGuard. AndroGuard is an extraction APK tool that can extract and decompile the APK file.

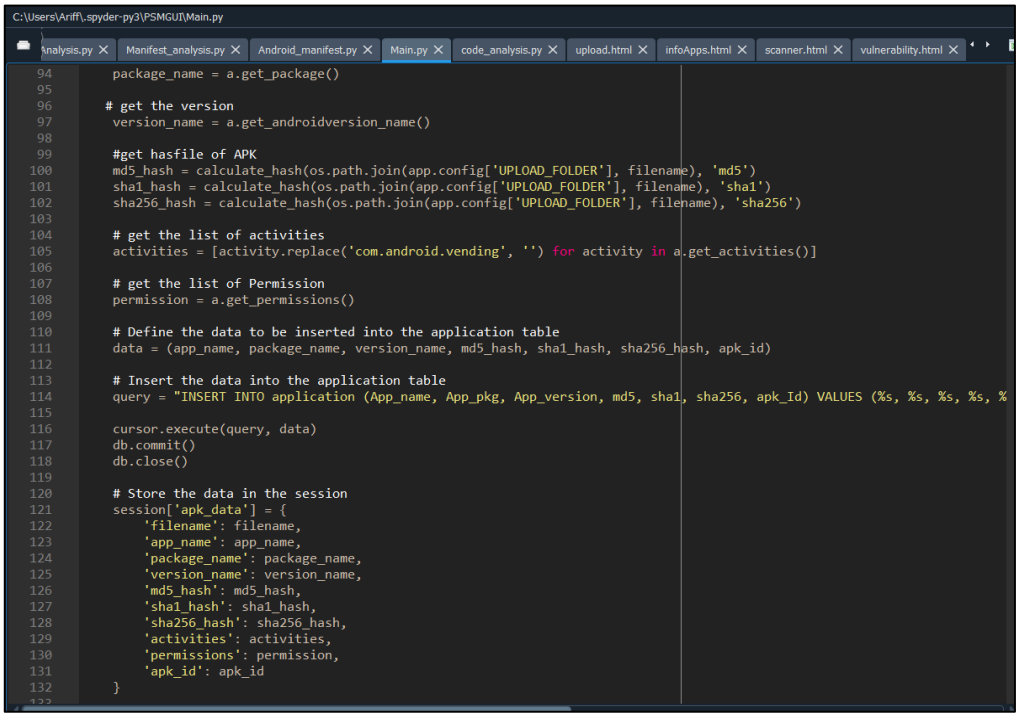


Figure 6: Code segment for extracting and decompiling APK files

Figure 7 is the example output from the extraction process. As can be seen, the application information, file information, permissions and activities are listed. All the listed information is from the extraction process in Figure 6.

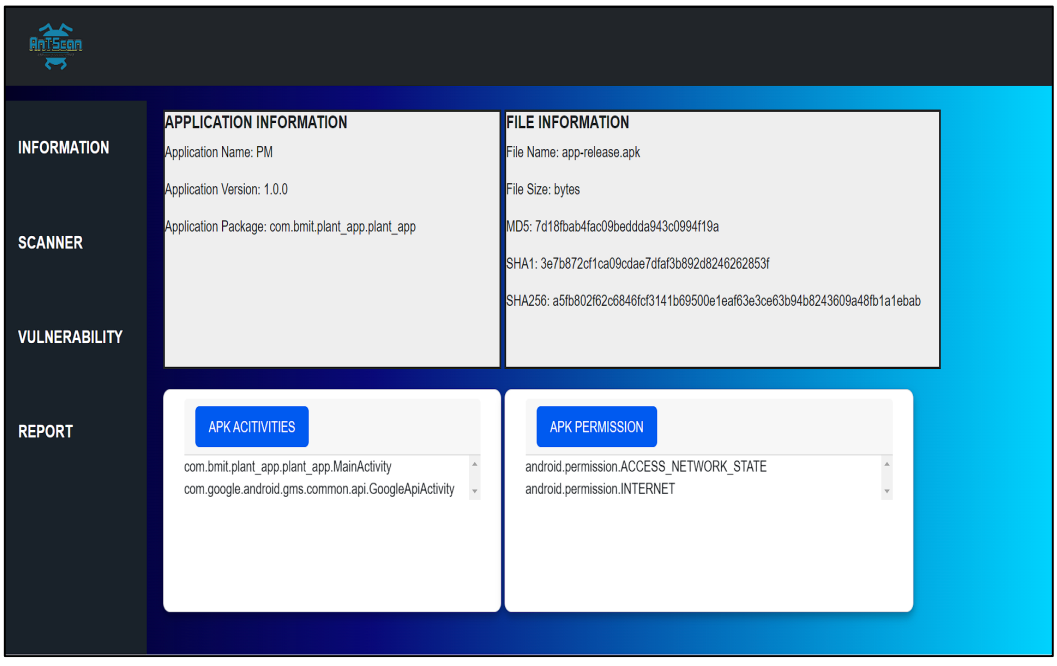
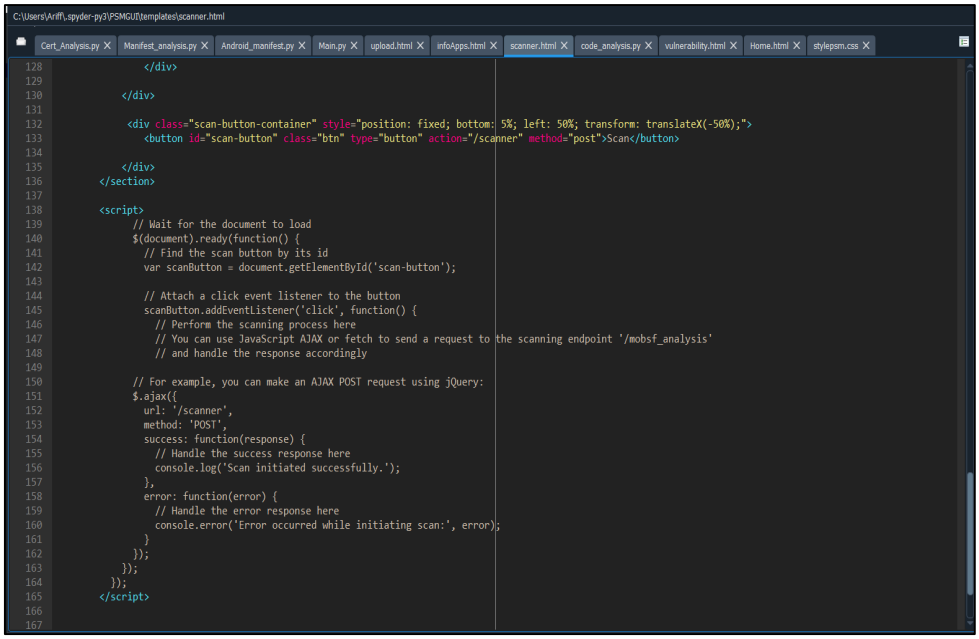


Figure 7: Interface for Information Application

In Figure 8 is a process for scanning, in this code it shows that there is button that will be as initiator to execute the analysis process. When the user clicks the button, the action attribute in the <button> type will identify the route that has name "/scanner". From that all the functions under the route /scanner will be executed and the analysis process will begin.



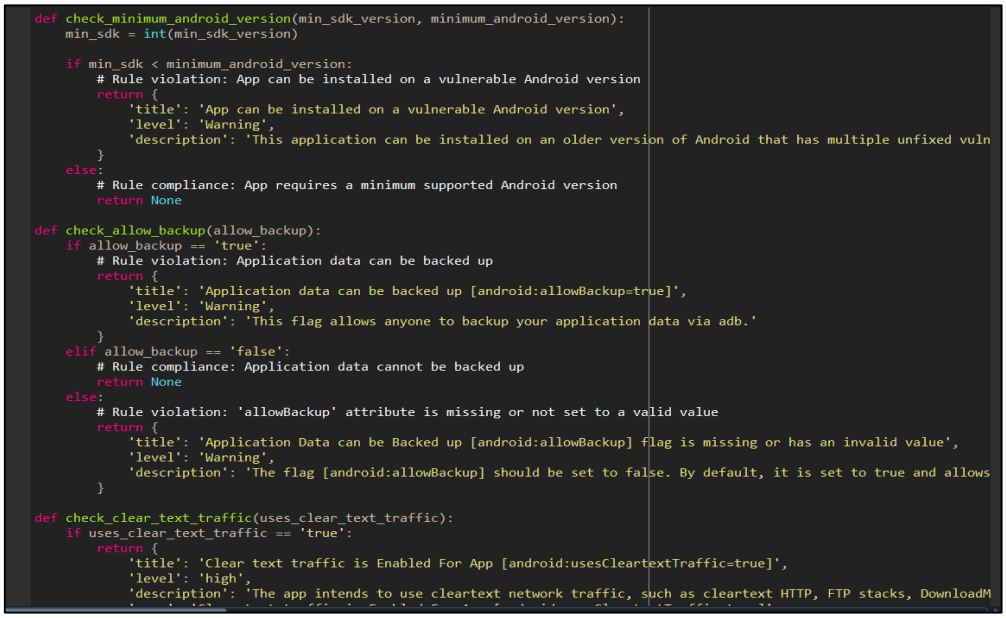
```
C:\Users\Ariff\.spyder-py3\PSMGUI\templates\scanner.html

Cert_Analysis.py | Manifest_analysis.py | Android_manifest.py | Main.py | upload.html | infoApps.html | scanner.html | code_analysis.py | vulnerability.html | Home.html | stylepsm.css

128                </div>
129
130            </div>
131
132            <div class="scan-button-container" style="position: fixed; bottom: 5%; left: 50%; transform: translateX(-50%);">
133                <button id="scan-button" class="btn" type="button" action="/scanner" method="post">Scan</button>
134
135            </div>
136        </section>
137
138        <script>
139            // Wait for the document to load
140            $(document).ready(function() {
141              // Find the scan button by its id
142              var scanButton = document.getElementById('scan-button');
143
144              // Attach a click event listener to the button
145              scanButton.addEventListener('click', function() {
146                // Perform the scanning process here
147                // You can use JavaScript AJAX or fetch to send a request to the scanning endpoint '/mobsf_analysis'
148                // and handle the response accordingly
149
150              // For example, you can make an AJAX POST request using jQuery:
151              $.ajax({
152                url: '/scanner',
153                method: 'POST',
154                success: function(response) {
155                  // Handle the success response here
156                  console.log('Scan initiated successfully.');
157                },
158                error: function(error) {
159                  // Handle the error response here
160                  console.error('Error occurred while initiating scan:', error);
161                }
162              });
163            });
164          });
165        </script>
166
167
```

Figure 8: Code segment for Button

As for the analysis or scanning process, there are three processes which will be done which is code analysis, manifest analysis, permission analysis, and certification analysis. All this analysis will be executed when the user clicks the scan button, and it will start the process. As in Figure 9, it is a code segment that contains rules for manifest analysis. During the analysis, manifest analysis will analyze xml code to identify whether the rule is being followed or not, if the rules is not being followed it will return the title, the level risk or severity and the description of the rules. All the analysis process must be based on the rules that have been set. The other rules segment for each analysis are being pasted at the appendix.



```
def check_minimum_android_version(min_sdk_version, minimum_android_version):
    min_sdk = int(min_sdk_version)

    if min_sdk < minimum_android_version:
        # Rule violation: App can be installed on a vulnerable Android version
        return {
            'title': 'App can be installed on a vulnerable Android version',
            'level': 'Warning',
            'description': 'This application can be installed on an older version of Android that has multiple unfixed vuln
        }
    else:
        # Rule compliance: App requires a minimum supported Android version
        return None

def check_allow_backup(allow_backup):
    if allow_backup == 'true':
        # Rule violation: Application data can be backed up
        return {
            'title': 'Application data can be backed up [android:allowBackup=true]',
            'level': 'Warning',
            'description': 'This flag allows anyone to backup your application data via adb.'
        }
    elif allow_backup == 'false':
        # Rule compliance: Application data cannot be backed up
        return None
    else:
        # Rule violation: 'allowBackup' attribute is missing or not set to a valid value
        return {
            'title': 'Application Data can be Backed up [android:allowBackup] flag is missing or has an invalid value',
            'level': 'Warning',
            'description': 'The flag [android:allowBackup] should be set to false. By default, it is set to true and allows
        }

def check_clear_text_traffic(uses_clear_text_traffic):
    if uses_clear_text_traffic == 'true':
        return {
            'title': 'Clear text traffic is Enabled For App [android:usesCleartextTraffic=true]',
            'level': 'high',
            'description': 'The app intends to use cleartext network traffic, such as cleartext HTTP, FTP stacks, DownloadM
```

Figure 9: Rules segment for Manifest analysis

As in Figure 10, it shows the sample code segment for manifest analysis. Show in the Figure 10, if the rule result is follow as the rules that have been set as in Figure 9, it will return the result which the title of the, the level or severity of the rules and the description of rules, if the rule result is not match with the rules, it will not return any result to the function at the main file. The remaining code segment in analysis process will be attached to the appendix.

```
# Check minimum Android version rule
rule_result = Android_manifest.check_minimum_android_version(min_sdk_version, minimum_android_version)
if rule_result:
    results.append({
        'rule_violation': True,
        'title': rule_result['title'],
        'level': rule_result['level'],
        'description': rule_result['description']
        })

# Check allowBackup rule
rule_result = Android_manifest.check_allow_backup(allow_backup)
if rule_result:
    results.append({
        'rule_violation': True,
        'title': rule_result['title'],
        'level': rule_result['level'],
        'description': rule_result['description']
        })

# Check direct-boot aware rule
rule_result = Android_manifest.check_direct_boot_aware(direct_boot_aware)
if rule_result:
    results.append({
        'rule_violation': True,
        'title': rule_result['title'],
        'level': rule_result['level'],
        'description': rule_result['description']
})
# Check vulnerable OS version rule
rule_result = Android_manifest.check_vulnerable_os_version(min_sdk_version, minimum_android_version)
if rule_result:
    results.append({
        'rule_violation': True,
        'title': rule_result['title'],
        'level': rule_result['level'],
        'description': rule_result['description']
        })
```

Figure 10: Code segment in Manifest Analysis

For Figure 11 is a sample of the analysis, it shows the result from permission analysis which the other analysis might not have any match rules. The output will be shown in the vulnerability interface which shows the vulnerability name, the severity of the vulnerability or level and description of the vulnerability. The result is being returned from the analysis process and being sent to the html code to print out the result for user view, on that page the user is only able to view the result and can do the scanning process again by entering the scanner interface.

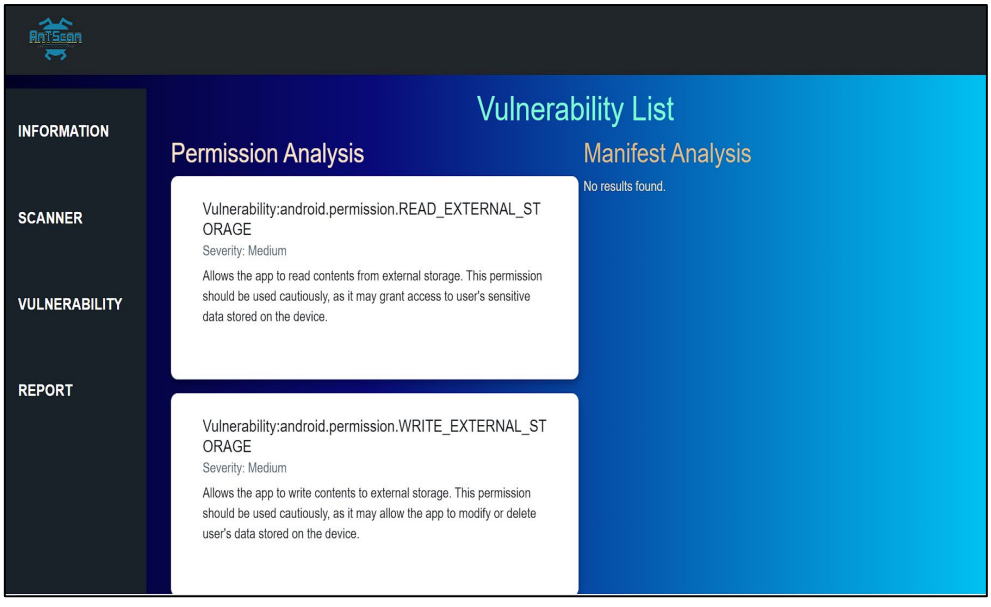


Figure 11: Vulnerability Interface

```python
@app.route('/report')
def report():
    return render_template('report.html')

@app.route('/generate_pdf')
def generate_pdf():
    # Specify the path to the HTML file
    html_file = 'print report.html'

    # Specify the output path and filename for the generated PDF report
    pdf_file = '/Reports/report.pdf'

    # Set up the configuration for pdfkit
    config = pdfkit.configuration(wkhtmltopdf='C:/Users/Ariff/.spyder-py3/wkhtmltopdf/bin/wkhtmltopdf.exe')

    # Generate the PDF report using pdfkit
    pdfkit.from_file(html_file, pdf_file, configuration=config)

    # Return the generated PDF as a response
    return send_file(pdf_file, attachment_filename='report.pdf', as_attachment=True)
```

Figure 12: Code Segment Report Generation

In Figure 12 it shows a code segment for generate pdf file from an html file. The purpose of this process is to generate a report after the process vulnerability scanning is complete. The tool that is being used to convert the html file into pdf file is Wkhtmltopdf. One of the tools that able to be convert html file into pdf file. As can be seen in Figure 12, the path of the pdf file has been set and it set the configuration for the pdf tools. Then it will return the pdf file as response. A button has been set to be an executor for the report generated.

```
13          <script src="https://unpkg.com/jspdf@latest/dist/jspdf.umd.min.js"></sc
14          <script>
15          const { jsPDF } = window.jspdf;
16         function generatePDF() {
17      // Create a new jsPDF instance
18      const doc = new jsPDF();
19
20      // Generate the PDF content
21      const reportContent = document.getElementById('report-content');
22      doc.text(reportContent.innerText, 10, 10);
23
24      // Save the PDF
25      doc.save('report.pdf');
26    }
27
28        </script>
```

Figure 13: Code Segment for Trigger Report Generation

As shown in Figure 13, the process of the report generation will be started when the button is clicked. It will get the id of the html code segment to identify which html that will be convert to PDF file. The completed report will automatically be downloaded into the download folder.

4.2 Test Case Plan

In this section, the test case plan is carried out to determine the result of AnTScan. The test case plan shows the list of tests and result of the testing. The expected result of the test plan is also included in the table to as standard test to the proposed tool.

Table 5: Test Case Plan

No.	Test Cases	Description	Result	Expected Result
	TEST 100			
1.	TEST_100_001	Tools should be able to connect with localhost.	Pass	Able to connect with localhost.
2.	TEST_100_002	Tools should be able to read rules that integrate in the code.	Pass	Able to read integrated rule.
3.	TEST_100_003	Tool should be able to read and call function in other file.	Pass	Able to Read function in external file.
4.	TEST_100_004	Tools should be able upload apk file.	Pass	Able to Upload apk.
5.	TEST_100_005	Tool should be able to do scanning process.	Pass	Able to do scanning.
6.	TEST_100_006	Tools should be able to do download report	Pass	Able to generate report.
	TEST 200			
1.	TEST_200_001	Upload button in AnTScan should be able to function well	Pass	Upload button able to upload.
2.	TEST_200_002	Scanning button should be able to function well	Pass	Scanning button able to initiate Scan.
3.	TEST_200_003	Download button should be able to function well	Pass	Download button able to download pdf report.
	TEST 300			
1.	TEST_300_001	Tools should be able to view the information of apk	Pass	Able to view apk information
2.	TEST_300_002	Tools should be able to view the list of vulnerability	Pass	Able to view list vulnerability
3.	TEST_300_003	Tools should be able to view report	Pass	Able to view report.

4.3 Test Case Result

There are 12 test cases used to test the developed tools. The tool passed 10 out of 12 test which is equivalent to 83% of the case. Table 6 sow the result of the test on AnTScan that being done by three lecturers that experience with vulnerability assessment.

Table 6: Overall Test Result of AnTScan

ID	Total Test Cases	Total Passed Cases
TEST_100	6	6
TEST_200	3	3
TEST_300	3	3
Overall Total	12	12

Based on Table 6, It shows the result of the test case according to the plan that has been discussed before. As shown in Table 6, for ID TEST_100 it shows that all the six tests have passed. For ID TEST_200, it has three total test cases, and all the three cases are passed. The ID TEST_300 shows all the tests have been passed. In details, TEST_100 is for the functionality of the proposed tool features while TEST_200 is for test the functionality of button, icon, or any element that clickable. The TEST_300 is focused on the viewing result from all the process. Thus, this tool has achieved the objective. All the three respondents have passed all the three cases and got the same result from the test. The three lectures have used the different apk file to identify any problems on this tool and find the similarity result with other existing tools.

5. Conclusion

This project is work on vulnerability assessment and android application. As we realized, Android smartphones have increased significantly over the last few years. Thus, there are a lot of applications that have been developed to make daily activities easier. The applications nowadays mostly are

connected to the internet which gives opportunity to the attackers to exploit it and steal all the information that is contained in the application.

As for this project, AnTScan has been designed and developed to overcome current weaknesses that occur in current existing tools. The objective of this project should be complete and AnTScan have been successfully being developed. The tools can do automated scanning to identify the vulnerabilities which help in reducing the time of analysis. The tool is easier to navigate as it has a simple and clear interface as their guideline to do the vulnerability assessment. The benefit of this tool is that it helps to reduce the number of vulnerabilities in an application and as this tool only focuses on android applications it is easier for the developers to choose which tool they can use.

AnTScan itself still has its own limitations and can be improved in future. First of all, AnTScan is capable of doing static analysis only which analysis the code without executing it. Dynamic analysis can be added to make the result become more accurate. As the name of this tool has "Testing" phrase it should be capable to do testing or exploitation activity to identify hidden flaws that can't be identified by both static and dynamic analysis. Lastly add more scanning function, as this tool is capable of doing a few analyses on the APK file.

In conclusion, AnTScan has accomplished the objectives, it has benefits in term of time and cost for identifying vulnerability. However, it still has limitations and areas to be upgraded in future, in term of functionalities, rules, and navigation of the tool. Hopefully, AnTScan is capable of being one of the reliable tools and widely used by pentester and developers.

Acknowledgment

The authors would like to thank the Faculty of Computer Science and Information Technology, Universiti Tun Hussein Onn Malaysia for its support.

References

[1] H. Abdullah and S. R. M. Zeebaree, "Android Mobile Applications Vulnerabilities and Prevention Methods: a Review," *2021 2nd Information Technology to Enhance e-learning and Other Application (IT-ELA)*, Dec. 2021, doi: https://doi.org/10.1109/it-ela52201.2021.9773615.

[2] D. R. Almeida, P. D. L. Machado, and W. L. Andrade, "Testing Tools for Android context-aware applications: a Systematic Mapping," *Journal of the Brazilian Computer Society*, vol. 25, no. 1, Dec. 2019, doi: https://doi.org/10.1186/s13173-019-0093-7.

[3] E. Blázquez and J. Tapiador, "Kunai: a Static Analysis Framework for Android Apps," *SoftwareX*, vol. 22, p. 101370, May 2023, doi: https://doi.org/10.1016/j.softx.2023.101370.

[4] Guyton, F., & Li, W. (2021). Feature Selection on Permissions, Intents and APIs for Android Malware Detection. In *ProQuest Dissertations and Theses*. https://www.proquest.com/dissertations-theses/feature-selection-on-permissions-intents-apis/docview/2524308633/se-2

[5] O. Hiremath, "Introduction to Cryptographic Failures," *Software Secured*, Jul. 25, 2022. https://www.softwaresecured.com/introduction-to-cryptographic-failures/

[6] Kim, J., & Corbo, D. (2020). Burp Suite: Automating Web Vulnerability Scanning. In *ProQuest Dissertations and Theses*. https://www.proquest.com/dissertations-theses/burp-suite-automating-web-vulnerability-scanning/docview/2403935378/se-2?accountid=50362

[7] V. Lenarduzzi, F. Pecorelli, N. Saarimaki, S. Lujan, and F. Palomba, "A critical comparison on six static analysis tools: Detection, agreement, and precision," *Journal of Systems and Software*, vol. 198, p. 111575, Apr. 2023, doi: https://doi.org/10.1016/j.jss.2022.111575.

[8] O. Nir, "OWASP Top Ten for 2022 - a Complete Review," *Reflectiz*, Aug. 31, 2022. https://www.reflectiz.com/blog/owasp-top-ten-2022/ (accessed Nov. 22, 2022).

[9] A. T. Tunggal, "What Is a Vulnerability?" *Upguard.com*, Sep. 06, 2019. https://www.upguard.com/blog/vulnerability

[10] Z. Quan-yin, J. Yin, X. Chengjie, and G. Rui, "A UML Model for Mobile Game on the Android OS," *Procedia Engineering*, vol. 24, pp. 313–318, 2011, doi: https://doi.org/10.1016/j.proeng.2011.11.2648.

[11] BeyondTrust, "What is Vulnerability Assessment?" *Beyondtrust.com*, 2014. https://www.beyondtrust.com/resources/glossary/vulnerability-assessment (accessed Jan. 25, 2020).

[10] BeyondTrust, "What are Hardcoded Passwords/Embedded Credentials?" *www.beyondtrust.com*, Aug. 18, 2021. https://www.beyondtrust.com/resources/glossary/hardcoded-embedded-passwords (accessed Apr. 20, 2021).

[11] N. Dupaul, "Static Testing vs. Dynamic Testing," *Veracode*, Dec. 03, 2013. https://www.veracode.com/blog/secure-development/static-testing-vs-dynamic-testing

[12] Imperva, "What Is Vulnerability Assessment | VA Tools and Best Practices | Imperva," *Learning Center*, 2022. https://www.imperva.com/learn/application-security/vulnerability-assessment/

[13] Praveen, "Understanding the Five Phases of the Penetration Testing Process," *Cybersecurity Exchange*, Mar. 28, 2022. https://www.eccouncil.org/cybersecurity-exchange/penetration-testing/penetration-testing-phases/#:~:text=The%20third%20penetration%20testing%20phase (accessed Jun. 19, 2023).

[14] A. Singh, N. Jaswal, M. Agarwal, and D. Teixeira, *Metasploit Penetration Testing Cookbook: Evade antiviruses, bypass firewalls, and exploit complex environments with the most widely used penetration testing framework, 3rd Edition*. Packt Publishing Ltd, 2018. Accessed: Jun. 19, 2023. [Online]. Available: https://books.google.com.my/books?id=EOlODwAAQBAJ&lpg=PP1&ots=tuOenPwjEQ&dq=metasploit%20method%20vulnerability%20analysis&lr&pg=PA80#v=onepage&q&f=false

[15] Κοντολέων, Δ., Kontoleon, D., & Νταντογιάν, Χ. (2018). Penetration Testing in Android Os. In *PQDT - Global*. https://www.proquest.com/dissertations-theses/penetration-testing-android-os/docview/2715582979/se-2?accountid=50362

[16] Zhu, Q. Y., Jin, Y., Xu, C., & Gen, R. (2011). A UML model for mobile game on the Android OS. *Procedia Engineering*, *24*, 313–318. https://doi.org/10.1016/j.proeng.2011.11.2648

[17] M. Buckbee, "What Is Metasploit? the Beginner's Guide," *www.varonis.com*, Feb. 24, 2022. https://www.varonis.com/blog/what-is-metasploit

[16] H. Gupta and R. Kumar, "Protection against Penetration Attacks Using Metasploit," *2015 4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions)*, Sep. 2015, doi: https://doi.org/10.1109/icrito.2015.7359226.

[18] G. Simran T and S. D, "Vulnerability Assessment of Web Applications using Penetration Testing," *International Journal of Recent Technology and Engineering*, vol. 8, no. 4, pp. 1552–1556, Nov. 2019, doi: https://doi.org/10.35940/ijrte.b2133.118419.

[19] AAT Team, "QARK - Free Android App Scanner to Find Security Vulnerabilities | All about Testing," 2021. https://allabouttesting.org/qark-free-android-app-scanner-to-find-security-vulnerabilities/ (accessed Dec. 21, 2022).