**Thato Sebelemetja – ST10067544**

**&**

**Malik Fahad Mannan - ST10091422**

**PATHWAY: APDS7311**

**Lecturer: Isaac Leshaba**

**04 September 2024**

**Application Development Security – POE Task 1**

**TABLE OF CONTENTS**

[**1. DATA-FLOW DIAGRAM: 3**](#_Toc176318610)

[**1.1. How We Will Secure Inputted Information? 5**](#_Toc176318611)

[**1.2. How We Will Secure Data in Transit? 5**](#_Toc176318612)

[**1.3. Plan of Action Against: 5**](#_Toc176318613)

[**2. SECURITY PLAN DIAGRAM 8**](#_Toc176318614)

[**3. MOBSF IMPLEMENTATION FOR OPSC7311 APP (CLOCKWORK) 10**](#_Toc176318615)

[**4. SCOUTSUITE IMPLEMENTATION 14**](#_Toc176318616)

[**REFERENCES: 18**](#_Toc176318617)

# **DATA-FLOW DIAGRAM:**

A diagram of a diagram

Description automatically generated

Figure 1: Data-Flow Diagram

*Figure 1* above shows a overall sequential process in which data moves from customer login through secure processing, storage, and employee verification before reaching its ultimate destination, SWIFT. The diagram highlights the importance of securely managing data throughout the process, including input validation, encryption, and secure storage. Here's a breakdown of the nodes and the overall flow:

* **Customer Login**: This is where the customer initiates the process by entering their credentials (username, account number, password).
* **Input Validation**: The system checks the customer’s input to ensure it’s correctly formatted and secure, preventing invalid data from entering the system (medium, 2021).
* **Password Security**: Passwords are hashed and salted to ensure they are stored securely in the database (cybernews, 2022).
* **SSL/TLS Encryption:** All data in transit between the customer’s device and the server is encrypted using SSL/TLS to prevent interception (ssldragon, 2024).
* **Payment Processing**: Once logged in, the customer initiates a payment by providing the necessary transaction details.
* **Data Validation**: The payment details (like amount, recipient account info, and SWIFT code) are validated to ensure accuracy (bright, 2022).
* **Secure Database Storage**: The validated transaction details are securely stored in the database.
* **Employee Verification**: Bank employees access the system to verify the stored transactions.
* **Input Validation for Employee**: This step ensures that employee inputs are correctly formatted and secure.
* **SSL/TLS for Employee:** The data in transit during employee interactions with the system is encrypted to maintain security (ssldragon, 2024).
* **SWIFT Submission:** After verification, the transaction is securely sent to SWIFT for final processing.

## **How We Will Secure Inputted Information?**

* **Input Validation:** Implement rigorous validation rules to ensure that all input data is properly formatted, avoiding the possibility of injection attacks (medium, 2021). This includes validating user-provided data such as names, ID numbers, account numbers, and SWIFT codes.
* **Password Security:** Use strong, salted hash functions like bcrypt to store passwords securely. This ensures that even if the database is compromised, passwords remain protected (cybernews, 2022).
* **Encryption:** Encrypt sensitive data such as account numbers and ID numbers before storing them in the database. This protects data at rest, ensuring that unauthorized access to the database does not expose this information (cybernews, 2022).
* **Use of Secure Libraries:** Leverage established security libraries and frameworks that follow best practices for handling sensitive data (medium, 2021).

## **How We Will Secure Data in Transit?**

* **SSL/TLS Encryption:** Use SSL/TLS protocols to encrypt all data exchanged between the client and server. This ensures that even if data is intercepted during transmission, it remains unreadable to attackers (ssldragon, 2024).
* **Token-based Authentication:** Implement token-based authentication (e.g., JWT) to ensure that each request is authenticated and authorized, reducing the risk of session hijacking (instasafe, 2024).
* **HSTS (HTTP Strict Transport Security):** Enforce HTTPS connections by using HSTS, which ensures that browsers always connect using secure protocols and prevent downgrade attacks (certera, *s.d.*).

## **Plan of Action Against:**

* **Session Hijacking:**
* Secure Session Management: Implement secure session management practices, including the use of secure, HTTP-only cookies with the Secure and SameSite flags. This prevents cookies from being accessed via client-side scripts and ensures they are only sent over HTTPS (Globalsign, 2021).
* Session Timeouts: Enforce session expiration and automatic logout after periods of inactivity to reduce the window of opportunity for session hijacking (Globalsign, 2021).
* **Clickjacking:**
* X-Frame-Options Header: Use the X-Frame-Options HTTP header to prevent the website from being embedded in an iframe, mitigating clickjacking attacks (medium, 2021).
* Content Security Policy (CSP): Implement a CSP that restricts which sources can be used to frame your content, adding another layer of defence against clickjacking (medium, 2021).
* **SQL Injection Attacks:**
* Prepared Statements/Parameterized Queries: Use prepared statements and parameterized queries to ensure that user input is never executed as part of a SQL command (bright, 2022).
* ORM (Object-Relational Mapping): Utilize ORM frameworks that abstract database interactions and automatically protect against SQL injection by escaping user input (bright, 2022).
* Input Validation: Implement strict input validation to ensure that inputs conform to expected formats and reject any input containing potentially malicious SQL code (bright, 2022).
* **Cross-Site Scripting (XSS) Attacks:**
* Output Encoding: Encode all user inputs before displaying them in the UI to prevent scripts from being executed in the browser. Use libraries such as OWASP’s ESAPI for this purpose (portswigger, *s.d.*).
* Content Security Policy (CSP): Implement a CSP to restrict the sources from which scripts can be loaded, preventing attackers from injecting malicious scripts (portswigger, *s.d.*).
* Input Sanitization: Apply sanitization techniques to user inputs to remove any potentially harmful code before processing or storing it (portswigger, *s.d.*).
* **Man-in-the-Middle (MITM) Attacks:**
* End-to-End Encryption: Use SSL/TLS to encrypt data from the client to the server and vice versa, ensuring that even if data is intercepted, it cannot be read or tampered with (fortinet. *s.d*).
* Public Key Pinning: Use HTTP Public Key Pinning (HPKP) to ensure that only trusted public keys are used during SSL/TLS handshakes, reducing the risk of MITM attacks through compromised Certificate Authorities (fortinet. *s.d*).
* DNS Security: Implement DNSSEC to protect against DNS spoofing, which is a common technique used in MITM attacks (fortinet. *s.d*).
* **Distributed Denial of Service (DDoS) Attacks:**
* Rate Limiting: Implement rate limiting on requests to ensure that your server can handle traffic without being overwhelmed by a large number of requests from a single source (cloudflare, *s.d*).
* Web Application Firewalls (WAF): Use a WAF to filter and monitor HTTP requests, blocking suspicious traffic patterns that resemble DDoS attacks (cloudflare, *s.d*)
* Network-Level DDoS Protection: Implement network-level protections such as using anti-DDoS services that can detect and mitigate DDoS attacks in real-time (cloudflare, *s.d*)

# **SECURITY PLAN DIAGRAM**

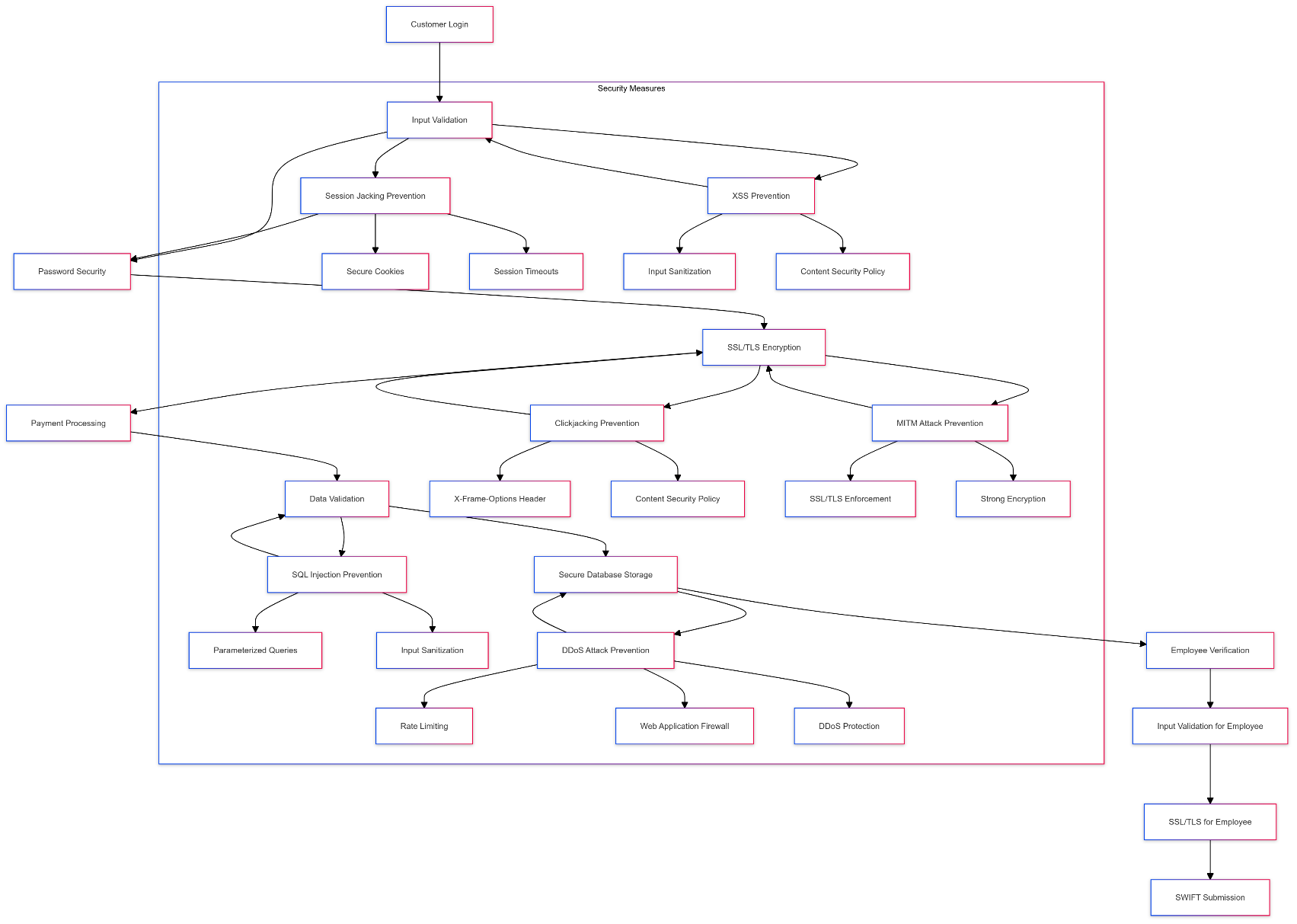


Figure 2: Security Plan Diagram

The illustration in Figure 2 outlines a methodical strategy for safeguarding an application, which involves implementing essential security protocols throughout different stages such as customer login, payment processing, and secure database storage.

The diagram's layout is strategically created to protecting against various possible dangers, incorporating preventive and reactive tactics. Here's a breakdown of the nodes and the overall flow:

* **Customer Login**: The starting point where users access the system. Secure login is crucial for protecting against unauthorized access.
* **Input Validation:** Ensures that all user inputs are properly validated to prevent malicious data from entering the system and to also protect against attacks such as Session Jacking and Cross-Site Scripting (XSS).
* **Password Security:** Implements strong password policies, including complexity requirements and encryption, to protect against brute force attacks and unauthorized access.
* **SSL/TLS Encryption:** Ensures that data transmitted between the user and the server is encrypted, preventing Man-in-the-Middle (MITM) attacks.
* **Payment Processing:** Handles sensitive financial transactions. This process is protected by both data validation and encryption to ensure that payment details are secure.
* **Data Validation:** A further step to ensure that all data, especially during transactions, is accurate and secure from manipulation. This also includes SQL Injection prevention methods.
* **Secure Database Storage:** Ensures that all sensitive data is stored securely, using encryption and access controls to protect against unauthorized access or breaches.
* **Employee Verification:** Verifies the identity and credentials of employees accessing the system, ensuring that only authorized personnel have access to sensitive areas.
* **Input Validation for Employee:** Similar to customer input validation but focused on employee actions, ensuring that internal operations do not introduce security risks.
* **SSL/TLS for Employee:** Ensures that all communication between employees and the system is secure, preventing MITM attacks within the internal network.
* **SWIFT Submission:** A specialized node for financial transactions that require extra security measures, such as those used in banking and international transfers.

# **MOBSF IMPLEMENTATION FOR OPSC7311 APP (CLOCKWORK)**

**Introduction:**

This report presents the findings from the MobSF (Mobile Security Framework) static analysis of the ClockWork mobile application, which was developed as part of the OPSC7311 course in semester one. The purpose of this analysis was to evaluate the security posture of the application and assess whether MobSF is a suitable tool for regular use by the organization’s security team.

**Overview of Findings:**

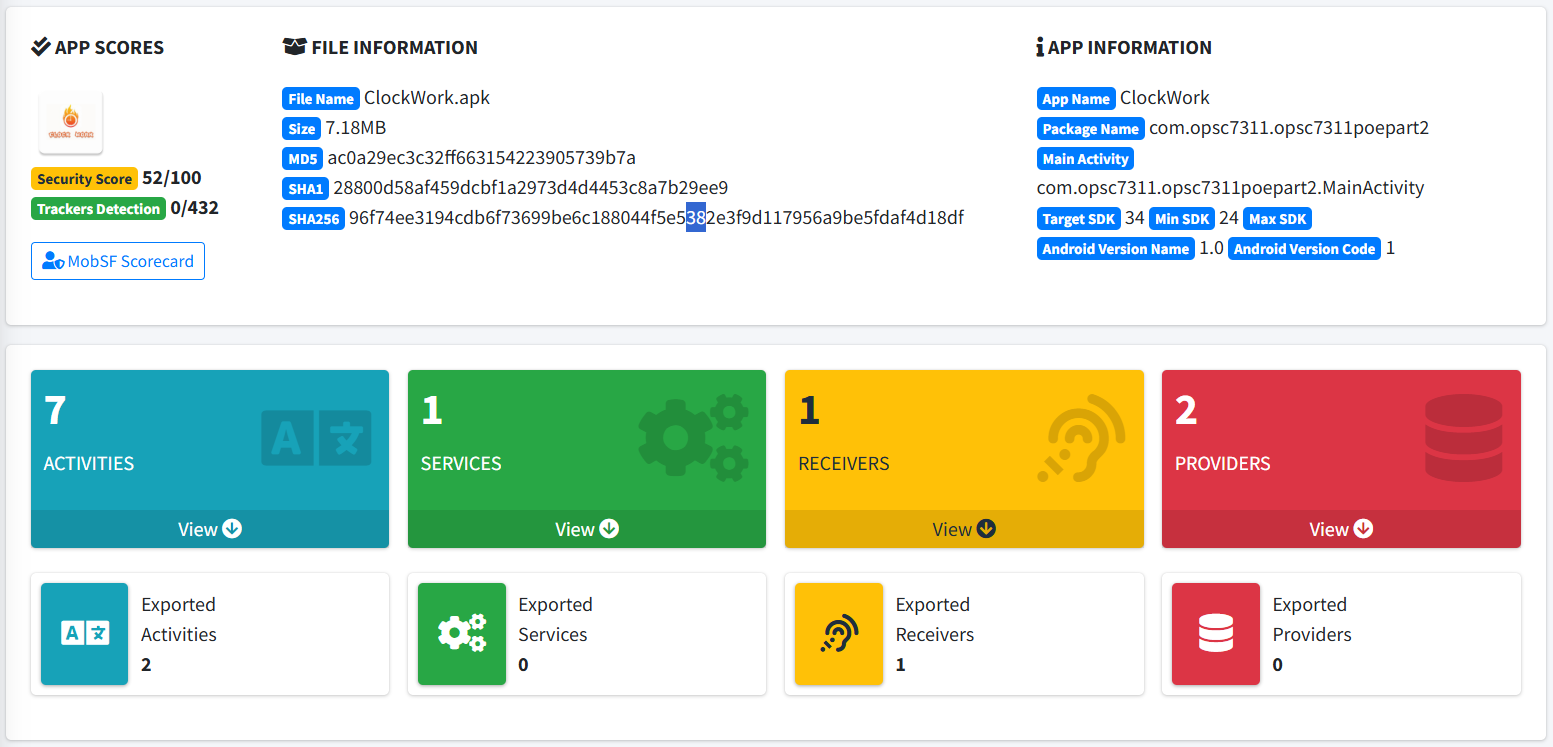
****

Figure 3: MobSF Dashboard displaying the analysis results of the ClockWork

A screenshot of a computer

Description automatically generated

Figure 4: PKID Analysis indicates potential protection techniques used in the app.

The static analysis of the ClockWork app resulted in an App Security Score of **52/100**, categorizing it as Medium Risk. The report identified a range of issues with varying levels of severity:

* High Severity Issues: 1
* Medium Severity Issues: 7
* Information: 1
* Hotspots: 1

**Key Issues Identified:**

* App Can Be Installed on Vulnerable Android Versions (High Severity):

The app is compatible with Android 7.0 and higher, which includes versions with known security vulnerabilities. This poses a significant risk, as older versions may not receive security updates.

* Application Data Backup (Medium Severity):

The android:allowBackup=true flag is set, which permits backing up application data via ADB. This could lead to unauthorized access to sensitive data if a device is compromised.

* Unprotected Exported Activities (Medium Severity):

Two activities, GenericIdpActivity and RecaptchaActivity, are exported and unprotected, making them accessible to other applications on the device. This exposes the app to potential misuse by other apps.

* Potential Hardcoded Sensitive Information (Medium Severity):

The analysis flagged potential hardcoded sensitive information such as API keys and secrets in the code, which could be exploited if discovered by an attacker.

* External Storage Read/Write Permissions (Medium Severity):

The app can read and write to external storage, making it vulnerable to data theft or tampering by other apps with similar permissions.

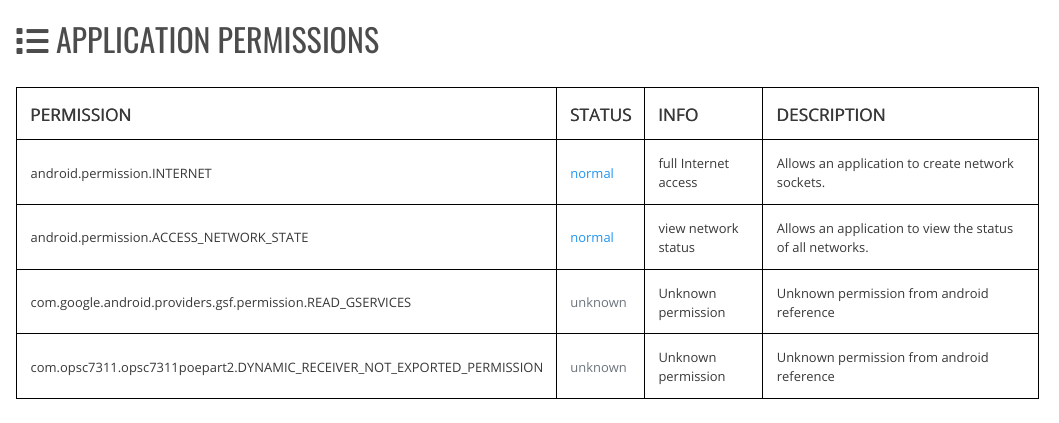


Figure 5: Permissions that are flagged as potentially risky or unknown.

* Sensitive Information Logging (Information Severity):

The app logs certain information, which may include sensitive data. Logging such information could lead to exposure in log files.

**Evaluation of MobSF as a Tool**

* **Advantages:**
* Comprehensive Analysis: MobSF provides a thorough examination of mobile apps, covering various aspects such as permissions, app components, and potential vulnerabilities (medium, 2023).
* Actionable Insights: The tool offers detailed descriptions of identified issues, making it easier for developers to understand and mitigate risks (medium, 2023).
* Automation: As an automated tool, MobSF allows for quick and repeated assessments, making it ideal for continuous integration pipelines (medium, 2023).
* **Disadvantages:**
* Limited Dynamic Analysis: While MobSF excels at static analysis, it does not offer extensive dynamic analysis, which is crucial for detecting runtime vulnerabilities (appknox, *s.d*).
* False Positives: The tool may flag non-issues or less critical items as significant, potentially leading to unnecessary work or overlooked critical risks (appknox, *s.d*).
* Usability Concerns: The interface, while functional, may require some learning curve, and integrating MobSF into existing workflows could demand additional setup (appknox, *s.d*).

**Conclusion**

MobSF proved to be an effective tool for identifying potential security issues in the ClockWork app. The findings indicate areas where the app's security could be improved, particularly concerning compatibility with older Android versions, data backup permissions, and the protection of sensitive information. However, while MobSF offers significant benefits, the security team should consider complementing it with additional dynamic analysis tools to cover a broader spectrum of potential vulnerabilities.

Given the analysis, it is recommended that MobSF be included in the organization’s security toolkit, if it is used alongside other tools and processes that address its limitations.

# **SCOUTSUITE IMPLEMENTATION**

**Overview**

ScoutSuite is a tool for auditing multi-cloud security that is open-source and allows security teams to evaluate the security status of their cloud environments. It was utilized in this project to review the AWS cloud infrastructure and pinpoint misconfigurations, excessively permissive policies, and other possible security risks (medium, 2021).

**Key Findings**

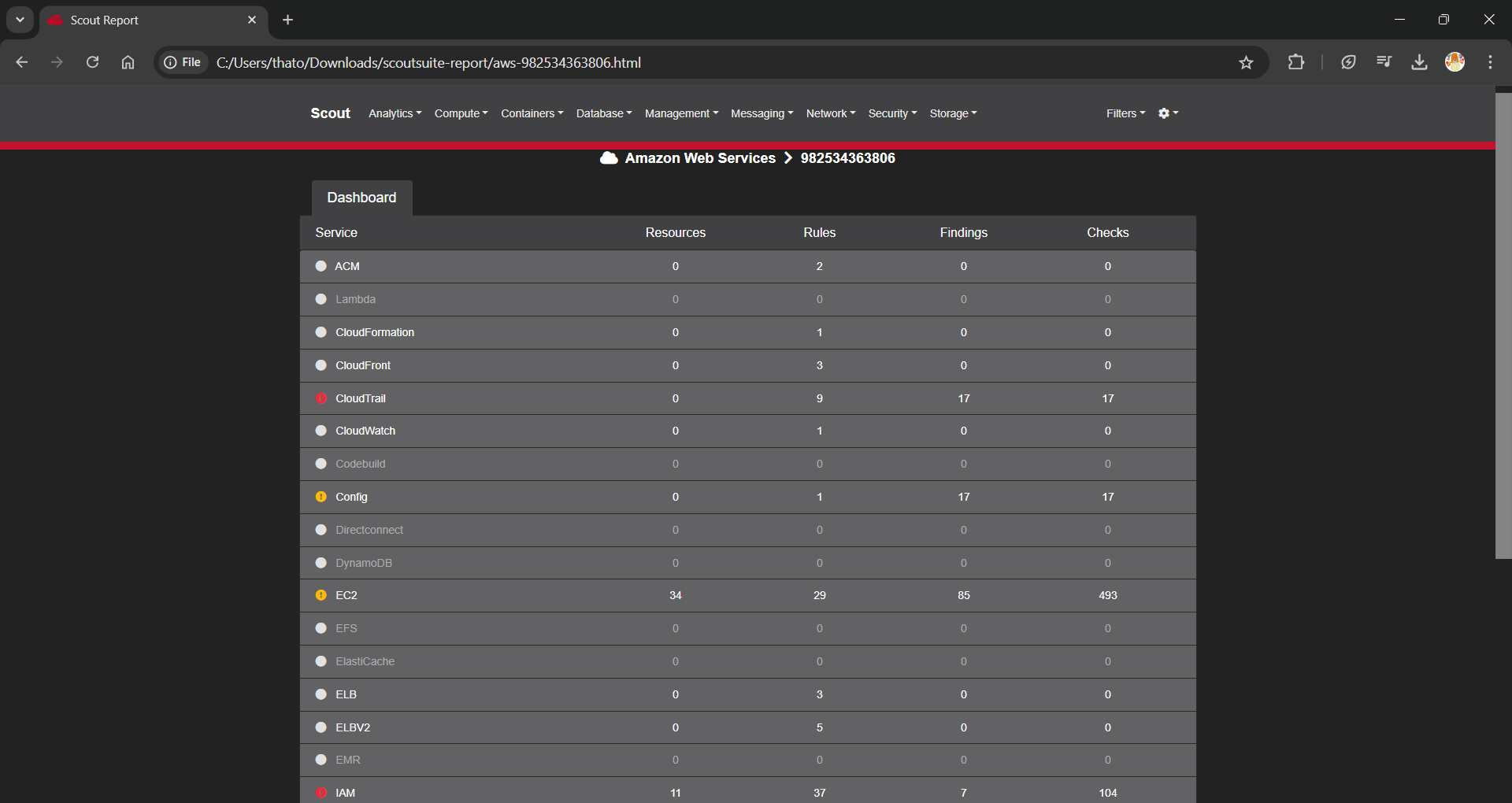


Figure 6: Dashboard of overall analysis from ScoutSuite

ScoutSuite was executed to scan the entire AWS account, focusing on critical services like CloudTrail, EC2, IAM, and S3. The tool generated a comprehensive report detailing the security posture of each service, highlighting misconfigurations and vulnerabilities.

**Key Issues Identified:**

A screenshot of a computer

Description automatically generated

Figure 7: Dashboard view of the CloudTrail service analysis

As can be seen in *Figure 7*, The CloudTrail platform detected important problems in the logging and monitoring of AWS account actions. The primary concern was the disabled logging, logs lacking encryption, and incomplete configurations for data logging. This risk may lead to insufficient monitoring of user actions, causing difficulties in conducting audits and possible security breaches.

A screenshot of a computer

Description automatically generated

Figure 8: Dashboard view of the EC2 service analysis

The EC2 service raises important security concerns, such as unencrypted EBS volumes and security groups with excessively lenient rules which can be seen in *Figure 8* above. These problems put instances at risk of unauthorized access and data breaches, underscoring the importance of stronger access controls and encryption.

A screenshot of a computer

Description automatically generated

Figure 9: Dashboard view of the IAM service analysis

The analysis of the IAM SERVICE uncovered several critical problems, such as improperly set roles and policies, ineffective password policies, and the absence of multi-factor authentication (MFA) for root accounts (See *Figure 9* above). These weaknesses have the potential to result in unauthorized entry, elevation of privileges, and compromised security throughout the AWS system.

**Conclusion**

ScoutSuite provided valuable insights into our AWS environment, helping us identify and address key security risks across services like IAM, EC2, and CloudTrail. By using ScoutSuite, we improved our cloud security posture, ensuring our infrastructure is better protected against potential threats.

# **REFERENCES:**

Medium. 2021. Three Steps to Perform on Input Data to Make Your Software More Secure, 18 June 2021. [Online]. Available at: <https://medium.com/codex/three-operations-on-input-data-to-make-your-software-more-secure-e5fc5aca2e70#:~:text=Three%20kinds%20of%20operation%20performed%20on%20input%20data%20can%20be> [Accessed 04 September 2024].

Cybernews. 2022. Hashing vs encryption vs salting: what's the difference? 14 March 2022. [Online]. Available at: <https://cybernews.com/security/hashing-vs-encryption/#:~:text=Hashing%20is%20similar%20to%20encryption%20in%20that%20it%20scrambles%20the> [Accessed 04 September 2024].

ssldragon. 2024. What is SSL/TLS? The Complete Guide to SSL Encryption, 21 May 2024. [Online]. Available at: <https://www.ssldragon.com/blog/what-is-ssl-tls/> [Accessed 04 September 2024].

Certera. [*s.d*]. HSTS Explained – Detailed Guide on HTTP Strict Transport Security. [Online]. Available at: <https://certera.com/blog/hsts-explained-detailed-guide-on-http-strict-transport-security/#:~:text=Know%20what%20is%20HSTS,%20how%20it%20works,%20advantages%20of%20HTTP> <https://cybernews.com/security/hashing-vs-encryption/#:~:text=Hashing%20is%20similar%20to%20encryption%20in%20that%20it%20scrambles%20the> [Accessed 04 September 2024].

Instasafe. 2024. What Is Token-Based Authentication and How Does It Work? [Online]. Available at: <https://instasafe.com/blog/what-is-token-based-authentication-and-how-does-it-work/#:~:text=Token-based%20authentication%20works%20by%20having%20a%20server%20generate,temporary%20digital%20signature%20to%20confirm%20the%20user%27s%20identity>. [Accessed 04 September 2024].

Globalsign. 2021. What is Session Hijacking and How Do You Prevent It? [Online]. Available at: <https://www.globalsign.com/en/blog/session-hijacking-and-how-to-prevent-it#:~:text=Session%20hijacking%20is%20a%20real%20threat%20and%20users%20are%20at> [Accessed 04 September 2024].

Medium. 2021. CSP frame-ancestors vs. X-Frame-Options for Clickjacking prevention., 29 November 2021. [Online]. Available at: <https://medium.com/@shaialon/csp-frame-ancestors-vs-x-frame-options-for-clickjacking-prevention-30383a713772#:~:text=Clickjacking%20(UI%20redressing)%20is%20a%20creative%20attack%20in%20which> [Accessed 04 September 2024].

bright. 2022. SQL Injection Attack: How It Works, Examples and Prevention, 8 April 2022. [Online]. Available at: <https://brightsec.com/blog/sql-injection-attack/#:~:text=Vulnerabilities.%20What%20is%20a%20SQL%20Injection%20Attack%20(SQLi)?%20SQL> [Accessed 04 September 2024].

portswigger, *s.d.* How to prevent XSS. [Online]. Available at: <https://portswigger.net/web-security/cross-site-scripting/preventing> [Accessed 04 September 2024].

fortinet. *s.d*. Man-in-the-Middle Attack: Types and Examples. [Online]. Available at: <https://www.fortinet.com/resources/cyberglossary/man-in-the-middle-attack> [Accessed 04 September 2024].

Cloudflare. *S.d.* How to prevent DDoS attacks | Methods and tools. [online]. Available at: <https://www.cloudflare.com/learning/ddos/how-to-prevent-ddos-attacks/> [Accessed 04 September 2024]

Medium. 2023. What is MobSF? 17 March 2023. [online]. Available at: <https://medium.com/@debasishkumardas5/secure-your-mobile-apps-with-mobsf-a-comprehensive-guide-to-android-and-ios-security-analysis-2ae7c928bf1d#:~:text=MobSF%20(Mobile%20Security%20Framework)%20is%20an%20open-source,#:~:text=MobSF%20(Mobile%20Security%20Framework)%20is%20an%20open-source>, [Accessed 04 September 2024]

appknox, *s.d*. Why MobSF Isn’t Ideal for Application Security Testing? [online]. Available at: <https://www.appknox.com/blog/why-is-mobsf-never-enough-for-your-mobile-app-security-testing#:~:text=Drawbacks%20of%20using%20MobSF%20for%20application%20security%20testing%201>. [Accessed 04 September 2024]

medium. 2021. Scoutesuite, 1 May 2021. [online]. Available at: <https://medium.com/info-script-intrusive/scoutsuite-340a7bb7958b#:~:text=Scout%20Suite%20is%20an%20open%20source%20multi-cloud%20security-auditing> [Accessed 04 September 2024]