Aatif Imtiaz Ahmed Muhammed - ST10241922
Qhawe Ngcongo - ST10283124
Uvalan Naidoo - ST10181713
Aiden Rungasami - ST10317201
INSY7314

Information systems Part 1

Table of Content

1.**A) Password Security**

Argon2id Hashing Algorithm

Biryukov, Dinu and Khovratovich (2016) argues that argon2id is still the most secure password hashing algorithm, and it was designed well enough to resist GPU and ASIC brute-force attacks since it is memory-hard.

standards and parameters

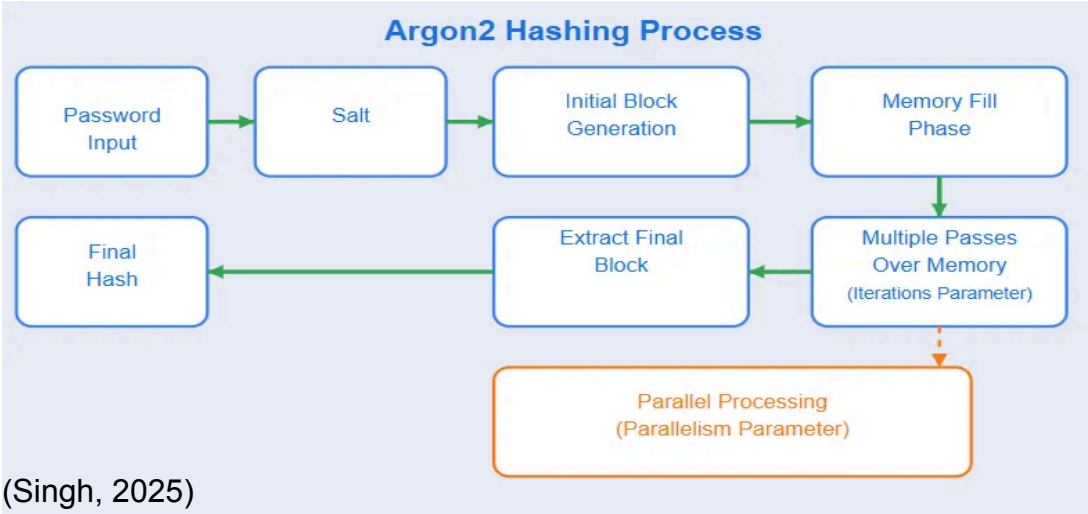
We will implement Argon2id according to RFC 9106 and align with NIST SP 800-63B requirements to store memorised secrets as salted, iterated hashes with tunable cost factors (Biryukov, Dinu and Khovratovich, 2021; NIST, 2020). Each account receives a unique 16–32-byte salt generated by a CSPRNG, and we keep a server-side pepper only in a secrets manager or HSM, not in the database. Argon2id will be calibrated so a single verification takes approximately 200–500 ms on production hardware, using initial parameters of memory 64–128 MB, time cost 3–4, parallelism ≈ CPU cores. We store only algorithms, parameters, salt and passwordHash, and use constant-time comparison to prevent timing leaks (OWASP, 2023a).

Process:

1. Generate for each password a unique, random 16-byte salt.
2. Hash the password by using both Argon2id and also the salt.
3. Do not ever store the plain password; store only the hash.
4. With extra protection, add a global pepper stored securely in a KMS or HSM.

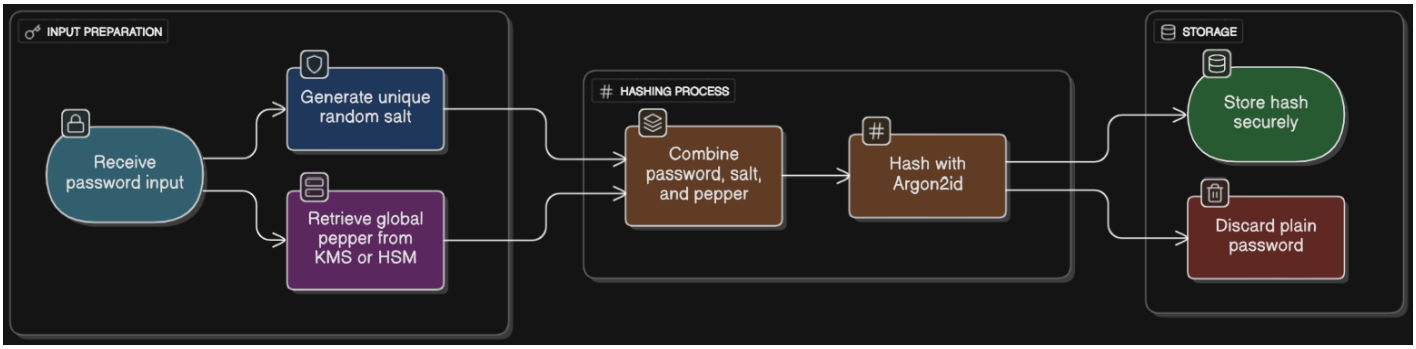
Diagrams to show the hashing process

Figure 1



(Singh, 2025)

Figure 2



(Canvas, n.d)

Quality and breach screening

At registration and reset we enforce a minimum 12–14 characters and screen candidate passwords against known-compromised lists using a k-anonymity service such as Pwned Passwords, as recommended by NIST and OWASP (NIST, 2020; OWASP, 2023a; Hunt, n.d.). We also rate-limit login attempts with progressive back-off and temporary lockouts (OWASP, 2023a).

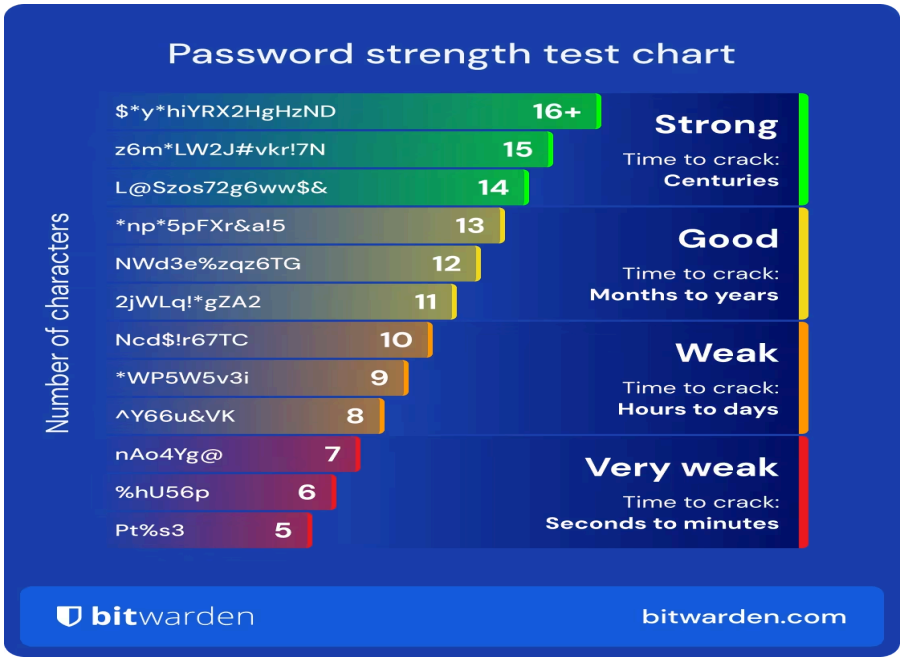
Password Policies

Strong password policies greatly reduce the likelihood of account compromise:

- Passwords must include uppercase, lowercase, numbers as well as symbols, with a minimum of 12 characters (OWASP, 2023).
- Passwords should not include credentials that are common or were previously hacked. Through integration, databases like Have I Been Pwned can filter these credentials.
- In order to prevent brute-force attacks, be sure to implement progressive lockout policies along with exponentially increasing backoff delays after about five failed login attempts.
- A password strength meter helps users make passwords that are secure plus strong.
- Weak passwords frequently cause breaches. Passwords that are reused could also be a major cause. Financial fraud as well as reputational damage may directly result from these weaknesses that exist in banking systems.

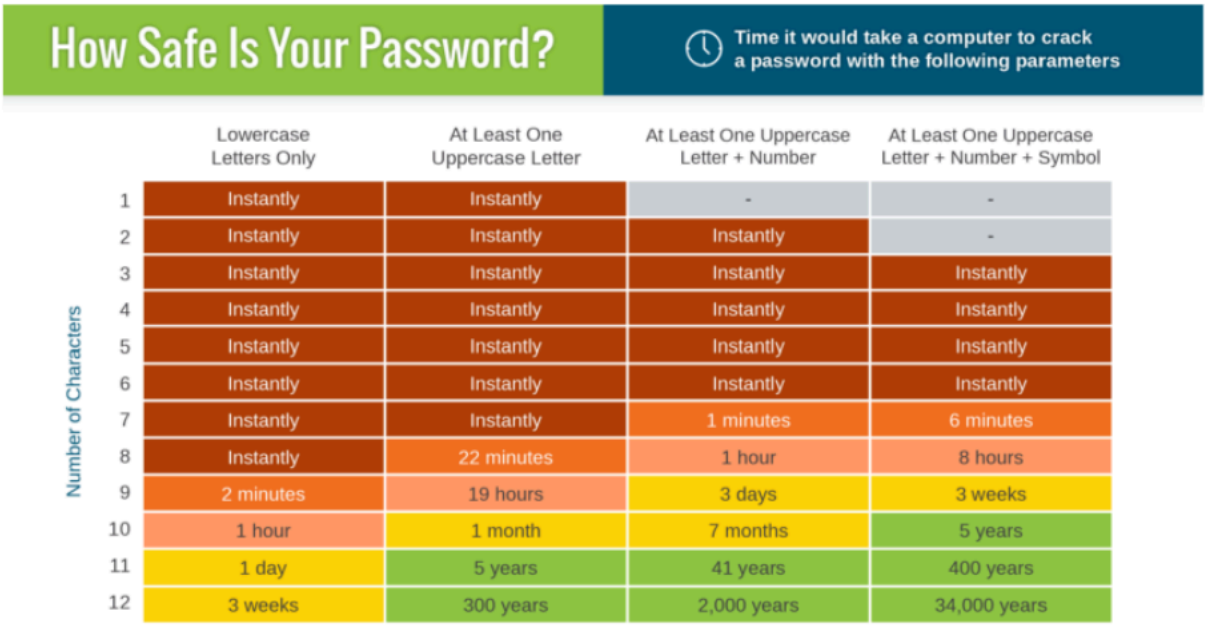
Infographics that show the how strong/safe a password can be

Figure 1



(Orenstein, 2022)

Figure 2



	Lowercase Letters Only	At Least One Uppercase Letter	At Least One Uppercase Letter + Number	At Least One Uppercase Letter + Number + Symbol
1	Instantly	Instantly	-	-
2	Instantly	Instantly	Instantly	-
3	Instantly	Instantly	Instantly	Instantly
4	Instantly	Instantly	Instantly	Instantly
5	Instantly	Instantly	Instantly	Instantly
6	Instantly	Instantly	Instantly	Instantly
7	Instantly	Instantly	1 minutes	6 minutes
8	Instantly	22 minutes	1 hour	8 hours
9	2 minutes	19 hours	3 days	3 weeks
10	1 hour	1 month	7 months	5 years
11	1 day	5 years	41 years	400 years
12	3 weeks	300 years	2,000 years	34,000 years

(Hightouchtechnologies, 2022)

Multi-Factor Authentication (MFA)

Even with strong passwords, important is forbidden access prevention with MFA:

- One-Time Passwords (OTP) provide a verification layer that is added through email or SMS delivery.
- Authenticator apps such as Google Authenticator generate time-based codes (TOTP).
- Biometric authentication is useful enough for use within mobile banking apps. Facial recognition as well as fingerprints exemplify this.

Adaptive MFA:

- It is triggered automatically at the time when there is detection of unusual behaviour, such as logins coming from an unrecognized device or a foreign country.
- For legitimate users, friction reduces. For high-risk scenarios, there is improved security.

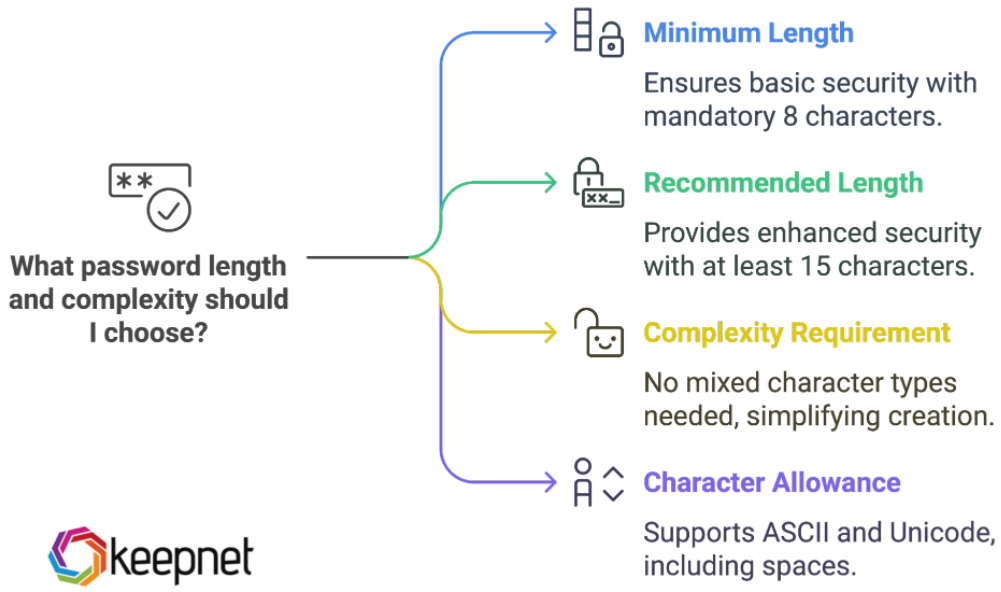
MFA policy

MFA is mandatory for staff and for high-risk customer actions. TOTP or FIDO2 is preferred, with SMS used only where stronger factors are unavailable, consistent with NIST 800-63 and OWASP guidance (NIST, 2020; OWASP, 2023a).

Password Lifecycle and Management

- Risk-based password resets require password resets of users only when a breach or suspicious activity is detected.
- Frequent mandatory resets are not ideal. These resets encourage users to adopt predictable and weak passwords.
- You should provide secure self-service password recovery for when you use identity-verified channels like email or SMS since that reduces reliance upon call centres that are vulnerable to social engineering.

Diagram to show most effective length and complexity:



(Keepnet, 2025)

Brute-force attack detection:

- Implement CAPTCHAs following repeated and failed login attempts now.
- Accounts which show that suspicious login behaviour is occurring should be locked automatically.

Credential stuffing detection:

- Login attempts should be compared against known breached credentials.
- Detects the unusual login velocity through the use of tools. This is exemplified via various logins from nations in brief time.

Audit logging:

- For investigation as well as compliance purposes, track account access, failed attempts, and all password changes.
- For suspicious activities security teams can respond by generating alerts rapidly.

Application to the Banking Portal and SWIFT Flow

- When a customer logs in, the system validates their password versus a securely hashed as well as salted version it stores.
- MFA gets triggered before transactions are allowed.
- In the event that employees access the SWIFT system, they must follow strong password and MFA rules combined with Role-Based Access Control (RBAC) in order to ensure only authorized staff access sensitive systems.
- Immediate alerts plus temporary account suspension happens upon any suspicious login attempt. Also, forced MFA re-verification is a potential result in turn.

2. Securing Input Data

Client-Side Security

- Sensitive fields such as passwords or account numbers should be masked.
- Disable the browser's automated credential filling.
- Input length must have validation for buffer overflow attacks.
- In order to block any automated bots from submitting malicious data, make use of reCAPTCHA.

Server-Side Validation

- Validate the allow-list with respect to trusted values such as codes for banks or currencies.
- Block encoding bypass tricks are blocked from canonicalized inputs.
- JSON schema validation achieves a correct data structure. This correct structure is enforced by validation.

Authoritative validation rules

Validation is server-side authoritative using allow-lists and strict schemas, with canonicalisation before checks as per OWASP (OWASP, 2023b). Examples:

- Currency must be on our ISO allow-list and amounts must meet range and decimal precision rules.
- IBAN must pass country-specific format and checksum per the SWIFT IBAN Registry (SWIFT, n.d.).
- SWIFT/BIC must conform to ISO 9362 structure, that is BIC8 or BIC11 with valid bank, country and location components (ISO, 2014).

Preventing Injection & XSS

- Parameterized queries or ORM frameworks: Always use against SQL Injection.
- SELECT * FROM Users WHERE username = @username
- **XSS Protection:**
    - CSP headers must be implemented soon.
    - Sanitize then escape all user-supplied data before display.

Input validation. Validate on the server using allow-lists and strict schemas. Examples: ISO currency codes; SWIFT/BIC format (8 or 11 uppercase chars, valid bank/country/branch structure); IBAN length and checksum per country; amount ranges and decimal precision. Normalize/canonicalize inputs before checks and reject unexpected fields.

Secure Storage

- Barker (2020) says that all sensitive data must be encrypted at rest with AES-256 and stored securely, following industry best practices
- Keys are stored securely by an HSM or cloud KMS.
- You must restrict access to data that is encrypted through the application of Role-Based Access Control (RBAC).

B) Securing the Data in Transit

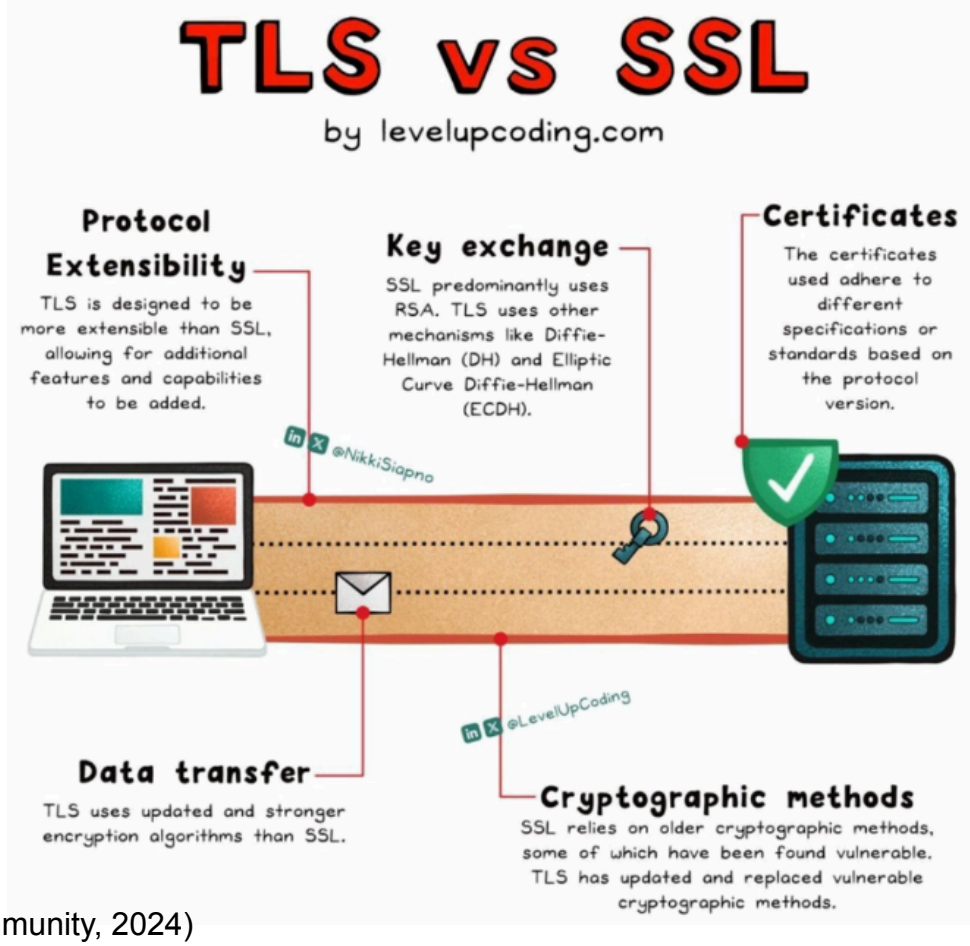
Data traveling between customers, banking systems, and also the SWIFT network must remain protected at each and every stage for this shall prevent interception, and tampering, together with fraud. Online banking is where cybercriminals target high-value transactions because this is critical. Data moving about retains its confidentiality, integrity, and also its authenticity. These habits guarantee that.

1. TLS vs SSL

Why is TLS Preferred?

- SSL has been fully replaced by Transport Layer Security (TLS) due to the fact of its even stronger security mechanisms.
- TLS creates a secure encrypted channel between the client (user) and the server (banking system).

Diagram to show differences between TLS and SSL



(Dev community, 2024)

TLS provides three fundamental security guarantees:

- Confidentiality: Data happens to be encrypted in order to prevent attackers from reading sensitive details such as passwords, account numbers, as well as SWIFT codes.
- Data integrity ensures it cannot be tampered with from others. Data transmission is just at the point when this thing happens.
- Authentication verifies the server's identity to stop malicious actors impersonating it.

TLS 1.3 is the protocol's latest version as well as it being what modern banking systems should mandate:

Strong Cipher Suites make use of secure encryption algorithms that exist. They include such algorithms as listed below.

- AES-256-GCM
- ChaCha20-Poly1305

Perfect Forward Secrecy (PFS):

- Earlier conversations stay safe even if a secret key gets exposed later.
- Ephemeral keys are used once generated uniquely during each session.

Reduced Handshake Steps:

In improving speed while in reducing attack surfaces, round trips happen to be fewer during connection establishment.

Removal of Insecure Algorithms:

Fully eliminated are RC4 and MD5 legacy ciphers.

2. Secure Communication Practices

- **HTTPS with TLS 1.3:**

  For prevention of downgrade attacks, all customer interactions must occur over HTTPS. Secure connections get forced thanks to HSTS.

  Strict-Transport-Security: max-age=63072000; includeSubDomains; preload

- **Mutual TLS (mTLS):**

  Mutual TLS adds validation of client certificates since the identities of both sides are verified.

  The bank confirms that the customer is connecting up from within a trusted banking application.

  Only the banking systems that are authorized can submit SWIFT transactions Bank → SWIFT.

  Session hijacking is prevented by it. It also protects people against rogue endpoints.

- **Certificate Pinning:**

  Mobile banking apps should implement certificate pinning to prevent rogue certificate attacks (OWASP, 2023).

Protocols and settings

We will enforce TLS 1.2/1.3 only, with TLS 1.3 preferred and legacy protocols/ciphers disabled. TLS 1.3 provides confidentiality, integrity and authenticated key exchange with simpler, safer cipher negotiation (IETF, 2018). We will allow AEAD cipher suites such as AES-GCM and ChaCha20-Poly1305 and ensure Perfect Forward Secrecy via ephemeral key exchange (OWASP, 2023a; IETF, 2018). On the public site we mandate HSTS to prevent protocol downgrade and mixed content; cookies are marked Secure, HttpOnly, SameSite (OWASP, 2023b; OWASP, 2023c).

For the Android app, we will restrict cleartext traffic and configure certificate pinning for the API domain via a Network Security Config, with backup pins and a documented rotation plan to avoid lockouts during CA changes (Android Developers, 2025; OWASP, 2023d).

mTLS controls

For backend-to-backend calls (bank → SWIFT integration), we will require mutual TLS with managed client certificates, short-lived credentials, OCSP stapling, and scheduled key/cert rotation. Client-certificate auth (mTLS) is the recommended pattern for service-to-service authentication where appropriate (OWASP, 2023e).

3. Protecting Against MITM Attacks

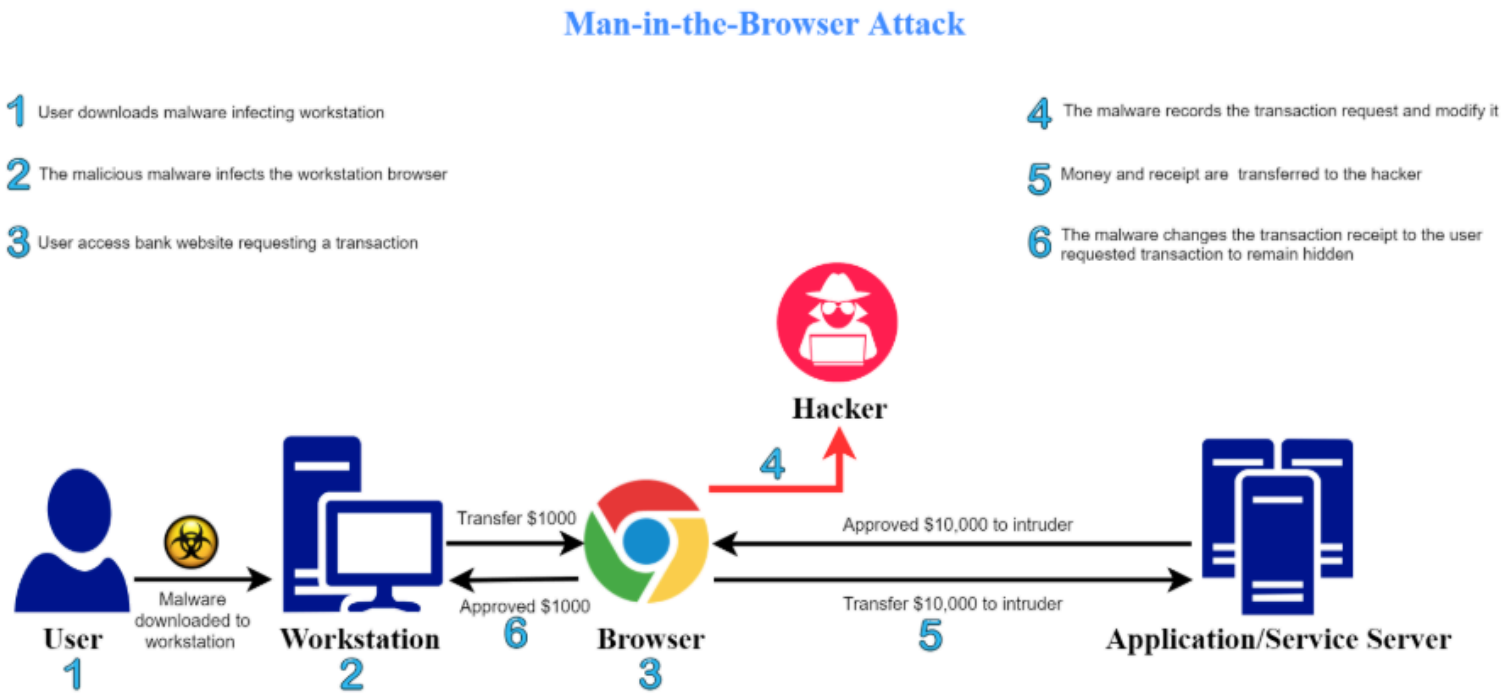
MITM attacks occur when attackers intercept secretly and manipulate the communication between the client and the bank. Online banking can stem from data manipulation, credential theft, and fraudulent transfers.

Mitigation Strategies:

- Digital certificates get validated in real time.
- Prevents attackers from exploiting revoked certificates, or expired certificates, as it reduces latency.
- Be alert and monitor network traffic on a continuous basis for any abnormal patterns. Unusual login attempts or sudden transaction spikes can be examples.

- In order to achieve centralized threat analysis, combine SIEM (Security Information and Event Management) with IDS.
- Internal communications between banking staff as well as servers must occur through VPNs or private networks for ensuring data is not exposed on public internet routes.
- For edge-case vulnerabilities, add encryption (AES-256) as another protective layer beyond TLS. This is recommended within sensitive fields like SWIFT codes or account numbers.
- This prevents browsers and apps from connecting except to HTTPS endpoints having known certificates, so downgrade attacks are impossible.

Diagram that shows how Man-in-the-browser attack works:



(Secret double octopus, 2020)

MITM specifics and evidence

In order to eliminate the need for clients to query the CA and enhance security and performance, we will enable OCSP stapling, which will cause the server to "staple" a signed revocation status during the TLS handshake (IETF, 2013). We will publish a HSTS policy containing max-age, includeSubDomains, and preload directives for public web traffic (OWASP, 2023b). Certificate pinning in conjunction with TLS 1.3 for the mobile application stops rogue intermediaries from being accepted; pinning will be done using backup keys and telemetry will be used to detect failures (Android Developers, 2025; OWASP, 2023d).

Acceptance proof includes a screenshot of the Android Network Security Config pin entries, HSTS presence, OCSP stapling enabled, and TLS scanner output verifying protocol/cipher policy.

4. Key Security Layers

Layer	**Example**
Encryption	TLS 1.3 with AES-256-GCM
Authentication	Mutual TLS certificates
Integrity	Perfect Forward Secrecy
Verification	OCSP stapling, HSTS
App Security	Certificate Pinning

Layer mapping

Encryption and integrity are provided by TLS 1.3 with AEAD ciphers; identity is assured by server certificates and, for service calls, client certificates (mTLS). PFS protects past sessions even if keys are later exposed; OCSP stapling and HSTS harden validation and transport; pinning constrains the trust anchor set on mobile (IETF, 2018; IETF, 2013; OWASP, 2023a; OWASP, 2023b; OWASP, 2023d).

Monitoring artefacts

We will capture TLS handshake telemetry (protocol, cipher, SNI, pin validation results) and alert on anomalies; rotate keys/certs on a defined schedule; and keep attestation that mTLS is enforced on the SWIFT connector (OWASP, 2023a; OWASP, 2023e). Acceptance evidence: SIEM dashboards showing TLS version/cipher distributions, alerts for pinning failures, and change records for certificate rotations.

5. Continuous Threat Monitoring and Response

Even the most secure systems are required to be continuously monitored in order to detect new dangers:

- Real-Time Alerts immediately notify administrators of suspicious activity.
- Ethical hacking exercises on a regular basis should be conducted. These exercises should simulate MITM with other attacks.
- Discovering vulnerabilities as well as patching vulnerabilities must be quite quick.
- In order to lower the risk of lateral movement by attackers, separate all SWIFT-related systems away from other internal networks.
- AWS Shield or Cloudflare can absorb Distributed Denial of Service (DDoS) attacks aimed toward banking APIs or portals.

Infographic to show cyber security monitoring:

(Aztech, 2024)

C) Hardening the Portal Against Security Threats

Given the sensitive and monetary information needed for an online banking system, security is the topmost factor. The employment of a multi-level defense-in-depth approach including the use of technical controls, secure coding, and active monitoring techniques will make the system more defensible and make it more difficult for assumed threats.

i. Session Jacking

Session jacking occurs where the attacker pretends to be the user by catching or guessing a session token. Through the creation of high-entropy session identifiers and the transmission of them only through encrypted TLS streams, the portal shall minimize such a threat. In an effort to prevent the running of scripts over the client-side, the session cookies shall be established with the Secure and HttpOnly flags. In an effort to prevent cross-site forgeries (CSRF), the portal shall incorporate the use of the SameSite attribute as well. Auto-timeouts and short-lived sessions shall also minimize the exposure (OWASP, 2021).

Short-lived session/access tokens with rotation on privilege change; SameSite=Lax/Strict for state-changing endpoints; server-side session store or signed JWTs with key rotation; CSRF tokens (synchronizer or double-submit) on all unsafe methods; idle and absolute timeouts; replay protection with nonces for high-risk actions; and logout on password/MFA change (OWASP, 2023a; OWASP, 2023b).

ii. Clickjacking

Clickjacking tricks victims into interacting with unseen UI elements loaded within malicious iframes. To prevent the same, the portal will apply the X-Frame-Options header DENY or SAMEORIGIN, preventing content from being framed on untrusted domains at all. As a secondary line of defense, a Content Security Policy (CSP) with frame-ancestors 'none' will be enforced. In situations where necessary, UI integrity checks will be implemented for ensuring that buttons or forms are not altered visually (OWASP, 2021; Gupta & Gupta, 2019).

Enforce X-Frame-Options: DENY (or SAMEORIGIN where framing on first-party is required) and Content-Security-Policy: frame-ancestors 'none' (or an explicit allow-list). Test via a headers scanner and manual iframe attempts on known attacker domains; document any approved framing exceptions (OWASP, 2023c).

iii. SQL Injection Attacks

SQL injection remains a threat if input validation is not necessary. To avoid this, the portal will utilise parameterised queries and prepared statements instead of dynamic concatenation of SQL. Input fields will be whitelisted and cleaned before processing. Database access will follow the principle of least privilege such that application accounts are unable to run unauthorised procedures. A Web Application Firewall (WAF) will provide an additional layer of real-time filtering (Halfond, Viegas & Orso, 2006).

Prepared statements/ORM for every query; no string concatenation; least-privilege DB roles with separate read/write users; input allow-lists and strict schema validation; WAF rules for common payloads; and automated SAST/DAST in CI to block regressions (OWASP, 2023d).

iv. Cross-Site Scripting (XSS)

XSS permits attackers to inject malicious scripts into a trusted application. The portal will have output encoding in place to guarantee that any user-supplied content is safely rendered. A CSP will restrict script execution to just trusted sources, thereby stopping inline script execution. Input sanitization libraries shall be used to eliminate malicious payloads. Furthermore, session cookies will be marked as HttpOnly to ensure that scripts cannot access sensitive session data (OWASP, 2021).

Context-aware output encoding (HTML, attribute, URL, JS contexts), template auto-escaping by default, sanitize any rich-text fields server-side, CSP forbidding inline/eval and pinning to trusted script origins, Subresource Integrity for third-party scripts, and HttpOnly cookies so scripts cannot read session tokens (OWASP, 2023e).

v. Man-in-the-Middle (MITM) Attacks.

The effectiveness of MITM attacks is based on the weaknesses of the communication link between clients and the server. In the portal, all data in motion will be protected using encryption with the TLS 1.3 standard, which will also make sure that the data is reliable and complete. Downgrading of protocols will be avoided via HSTS (HTTP Strict Transport Security) mechanisms. Moreover, certificates will be pinned and configured on mobile and web clients to block use of counterfeit certificates. Key rotation and certificate management will further strengthen resilience (Rescorla, 2018).

TLS 1.2/1.3 only with AEAD ciphers and Perfect Forward Secrecy; enforce HSTS on the public site; certificate pinning on the mobile app; mutual TLS for backend and SWIFT connectivity; OCSP stapling for revocation; and scheduled certificate/key rotation with change control (IETF, 2018; IETF, 2013; OWASP, 2023f; OWASP, 2023g).

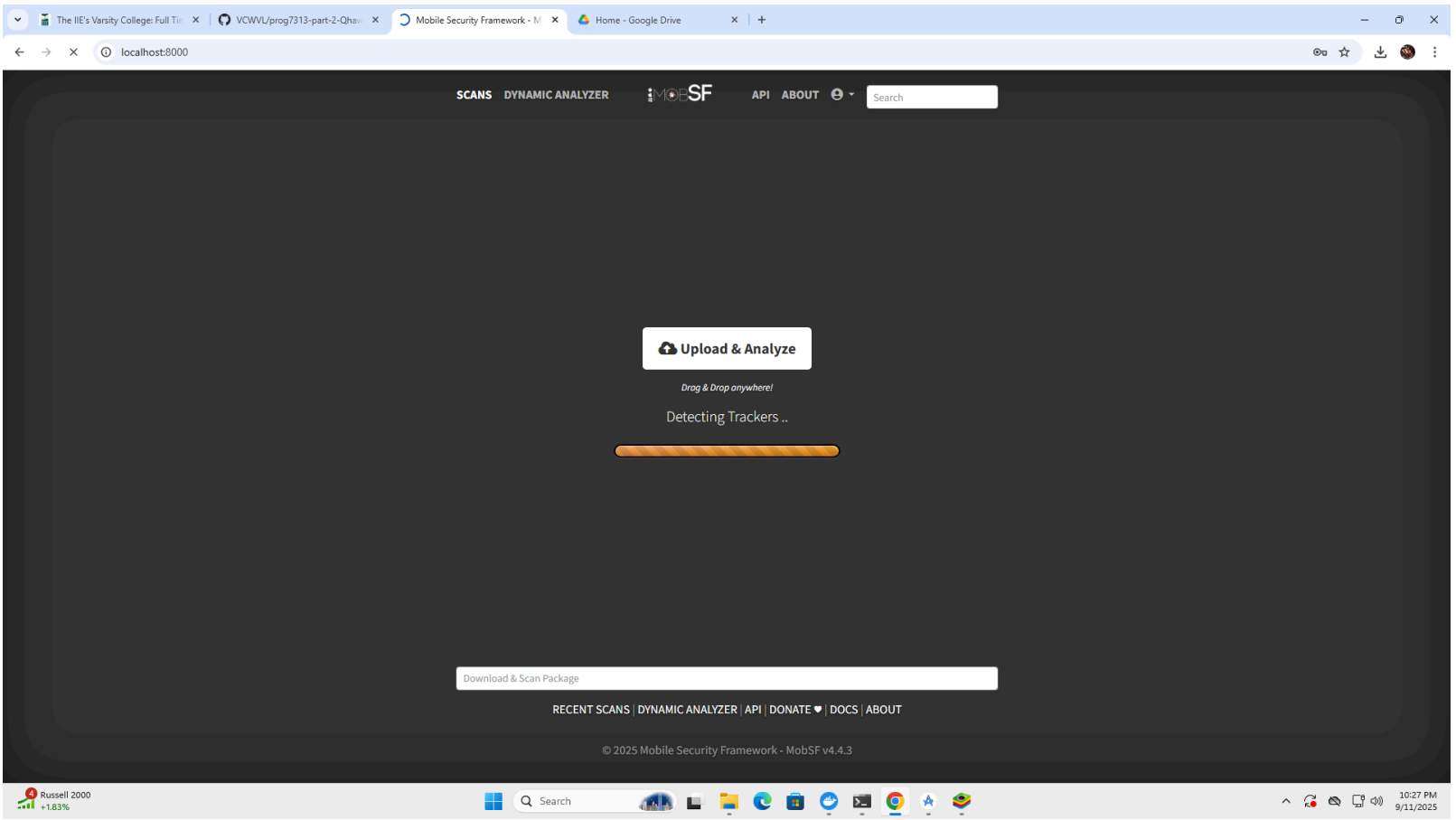
vi. DDoS Attacks

Attempt to attack by flooding the target system with requests. The portal will apply rate limiting and throttling on sensitive endpoints like login and transaction services. To mitigate traffic surges, load balancing and Content Delivery Networks (CDNs) will distribute and absorb the traffic. In addition, the system will use cloud-based DDoS protection (e.g. AWS Shield or Cloudflare). Effortless traffic supervision, Anomaly Based IDS, and real-time IP filtering will provide instant response (Zargar, Joshi, and Tipper, 2013).

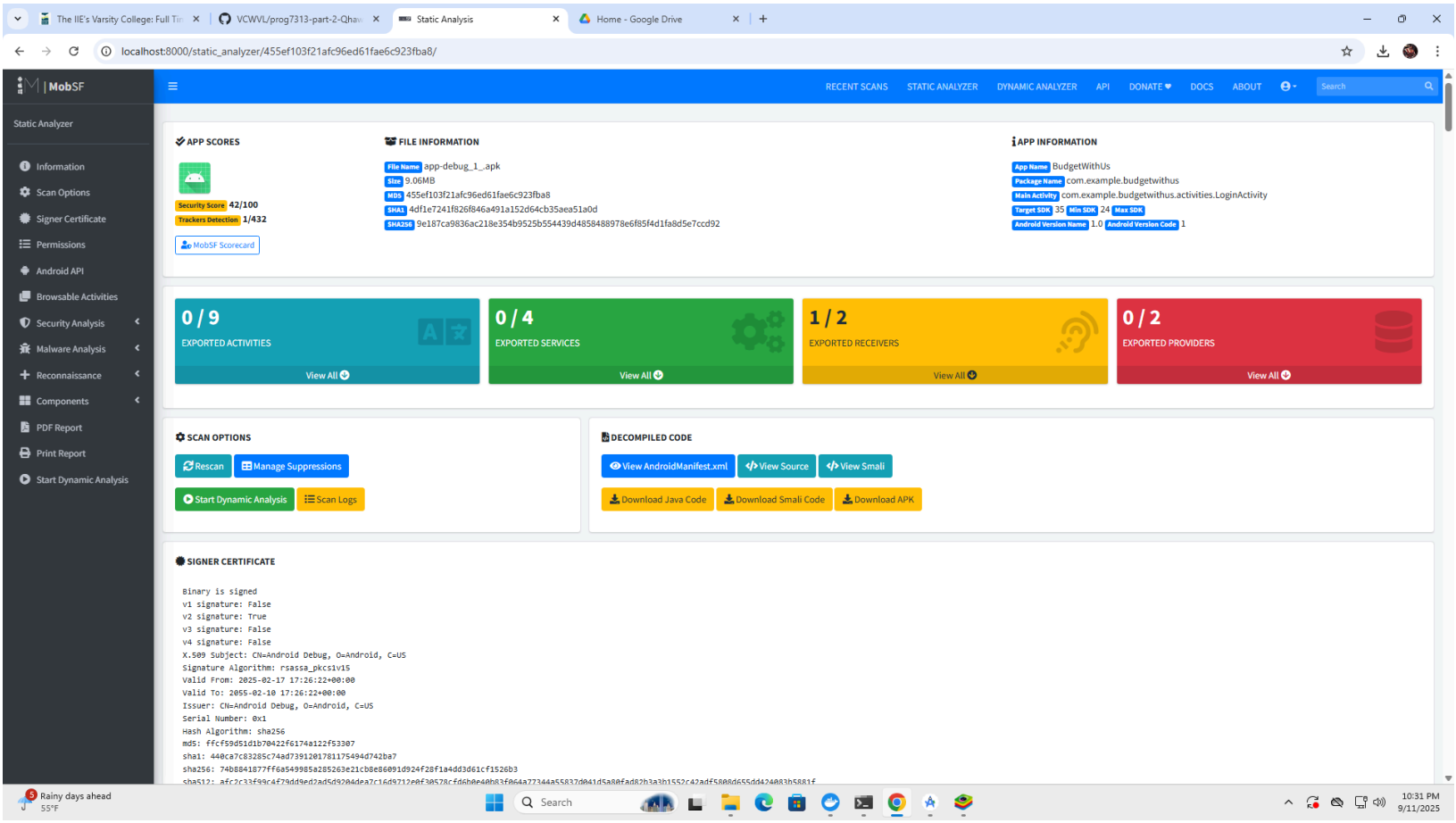
Rate limits (token-bucket/SLAs) on login and payment APIs; bot management and WAF at the edge; CDN caching for static/anon traffic; autoscaling and connection limits for burst absorption; SYN cookies/L4 protections; and enable a managed service such as AWS Shield Advanced for L3/L4/L7 attacks. Monitor with SIEM dashboards and protect SWIFT connectors behind strict egress allow-lists (AWS, 2025; OWASP, 2023h).

2. Evidence of mobsf implementation with screenshots

1.) Analyzing budget app with mobsf

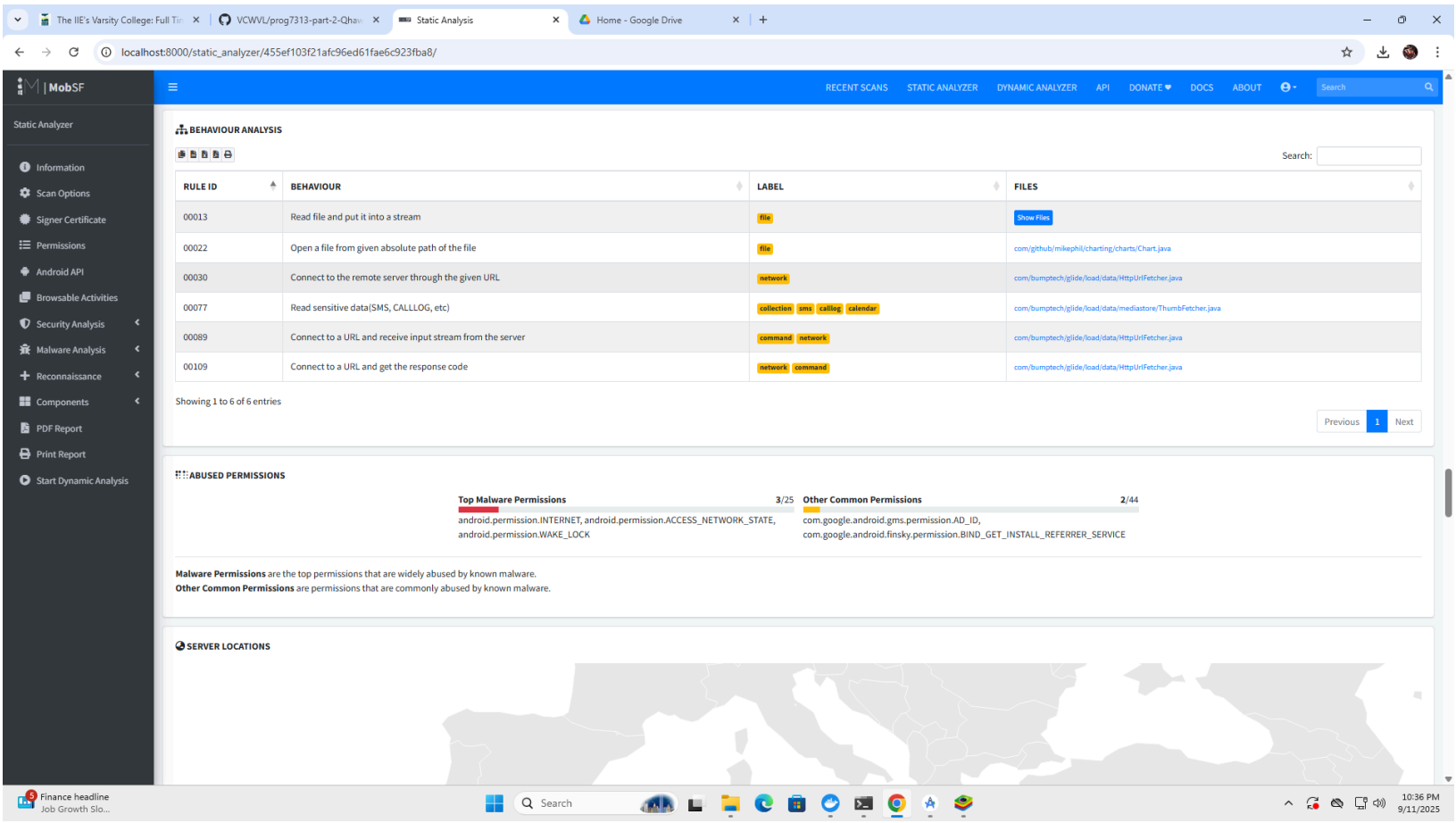


2.) Security score for budget tracking app



http://localhost:8000/static_analyzer/455ef103f21afc96ed61fae6c923fba8/

3.) Mobsf showing abused permissions



Mobsf static analysis report also submitted separately

Short ChatGPT Report: Evaluation of MobSF for Mobile Application Security Testing

App Tested: BudgetWithUs (Version 1.0)
Tool Used: Mobile Security Framework (MobSF v4.4.3)
Scan Date: 11 September 2025
Security Score: 42/100 (Medium Risk)
Grade: B

1. Purpose of the Test

The BudgetWithUs Android application (APK) was analyzed using MobSF to assess potential security risks before deployment. The objective was to determine whether MobSF is a suitable tool for identifying vulnerabilities in our mobile applications.

2. Key Findings

High-Risk Issues

- Debug Certificate in Use: The APK is signed with a debug certificate. Production apps must be signed with a release certificate.
- Debugging Enabled: android:debuggable=true was detected, which allows attackers to hook debuggers and reverse engineer the app.
- Support for Old Android Versions: Minimum SDK = 24 (Android 7.0), which exposes the app to known unpatched vulnerabilities.

Medium/Warning Issues

- Allow Backup Enabled: android:allowBackup=true could allow app data to be copied off the device via ADB.
- Exported Broadcast Receiver: May be accessible by other apps if permission checks are weak.
- Hardcoded Secrets Detected: Firebase database URL and Google API keys were found within the code.

- External Storage Access: The app can read/write to external storage, exposing sensitive data to other apps.

Informational Issues

- Sensitive Logging: Certain components may log sensitive information, violating OWASP MSTG guidelines.

- Firebase Remote Config Disabled: Currently secure, but indicates reliance on static configs.

Positive Findings

- No known malware domains were detected.

- Only one tracker was identified (Firebase Analytics).

- The application is signed, and certificates were validated.

3. Implications

If left unresolved, these issues could lead to:

- Unauthorized access to sensitive data (due to backups and external storage).

- Reverse engineering or tampering (debuggable flag, debug certificate).

- Exposure of API keys and Firebase endpoints, which can be exploited for unauthorized access.

4. Assessment of MobSF as a Tool

MobSF successfully identified multiple security risks in a short time, including misconfigurations, hardcoded secrets, and insecure permissions. Its automated static

analysis aligns with OWASP Mobile Security Testing Guide (MSTG) and provides actionable recommendations.

Strengths of MobSF:

- Automated scanning with detailed reports.
- Detects both code-level issues and manifest misconfigurations.
- Highlights hardcoded keys and insecure permissions.

Limitations:

- Generates many warnings from third-party libraries (e.g., Glide, MPAndroidChart), which may require filtering.
- Static analysis only; dynamic runtime issues require complementary testing.

5. Recommendation

I recommend adopting MobSF as part of our mobile security pipeline. While not a complete solution, it provides valuable early detection of risks during development. For best results, it should be complemented by:

- Dynamic analysis (runtime testing).
- Manual code review for sensitive areas (API keys, authentication logic).
- Integration into CI/CD to catch issues before deployment.

Hashing and Salting Strategy for the Banking Portal

Goal. Never store plaintext passwords. Store only a slow, key-stretched hash with a unique per-user salt. Keep any server-side secret separate from the database to limit damage if the DB is leaked.

Algorithm choice. Use Argon2id as the primary password hashing algorithm because it is memory hard and resists GPU attacks. If Argon2id is not available in the stack, use PBKDF2 with HMAC-SHA-256 as a fallback.

Salts. For every account, generate a random 16 to 32 byte salt with a cryptographically secure RNG. Store the salt alongside the hash. Do not reuse salts across users or password resets.

Pepper. Add a **server-side pepper** of at least 32 bytes. Store the pepper only in a secure secrets store or HSM, not in code or the database. Rotate the pepper on a defined schedule and after any suspected exposure.

Parameters. Calibrate on production hardware. For Argon2id, start with memory cost 64–128 MB, time cost 3–4, and parallelism equal to available CPU cores, then tune until a single hash takes about 200–500 ms on the server. Re-benchmark after hardware or version changes.

What we store. Save algorithm, version or parameter set, salt, and passwordHash. Do not store the pepper. Keep a rehashRequired flag so users are upgraded to newer parameters on next login.

Verification flow. On login, fetch the user's salt and parameters, append the server pepper to the submitted password, derive the hash using Argon2id or PBKDF2, and compare with a constant time comparison. On password change or reset, create a new salt and rehash with current parameters.

Password policy and guards. Enforce strong password rules at registration and reset, add rate limiting and temporary lockouts for repeated failures, and require MFA for console or staff access. All traffic must be over TLS.

Reset and remember-me tokens. Generate random tokens, store only a hash of each token, set short expiries, and enforce single use. Sign or HMAC tokens with a server key that is kept in the same secrets store as the pepper.

Why does this matter for our app? The mobile client should never hash or store passwords. All hashing must happen on the backend. Using salted, memory hard hashing with a server-side pepper and proper operational controls limits the blast radius of any database exposure and meets the rubric requirement for hashing and salting.

Conclusion: MobSF proved effective in identifying medium to high-risk vulnerabilities in the BudgetWithUs app. It is a strong candidate for integration into our mobile application security toolkit.

Do I support the use of the tool mobsf or not?

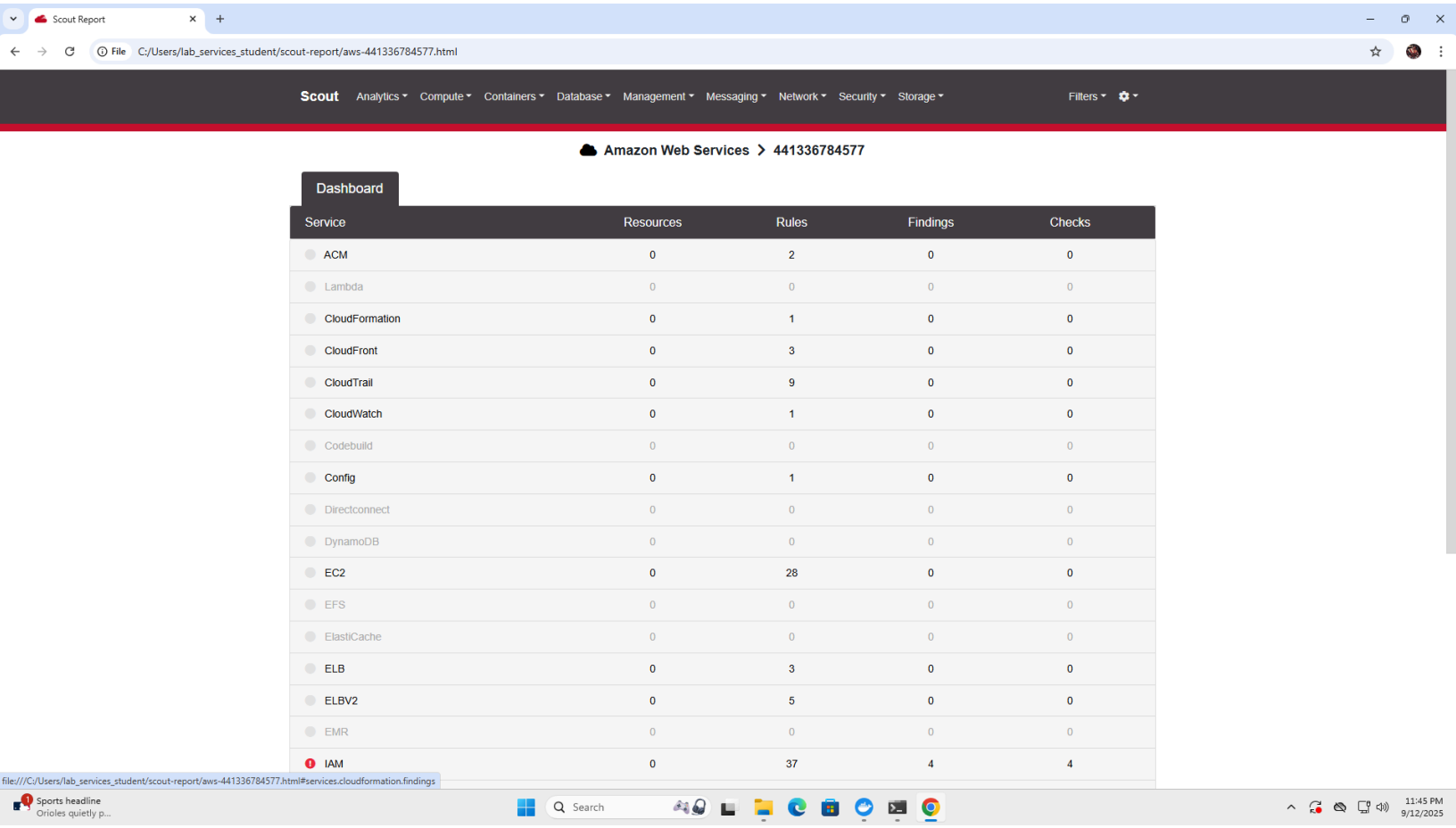
I support adopting MobSF as part of our mobile security pipeline. In a few minutes it surfaced real risks in our APK, such as a debuggable build, a debug certificate, allowBackup, an exported receiver, possible hard coded secrets, and external storage patterns. These are the kinds of misconfigurations that can lead to data exposure or tampering in a banking app. The tool maps findings to OWASP MASVS and MSTG, generates clear PDF and JSON evidence for CR packs, and is easy to run with Docker, which means we can make it a repeatable check on every build and wire it into CI or CD so releases fail when high risk issues appear.

MobSF is not a silver bullet; it is strongest at static analysis and can produce noise from third party libraries, so we still need dynamic testing, code review, and periodic penetration tests for high risk features. These limitations are manageable. Maintain a short ignore list for known benign items, pair scans with emulator or device testing, and keep manual reviews for authentication, cryptography, and data flows.

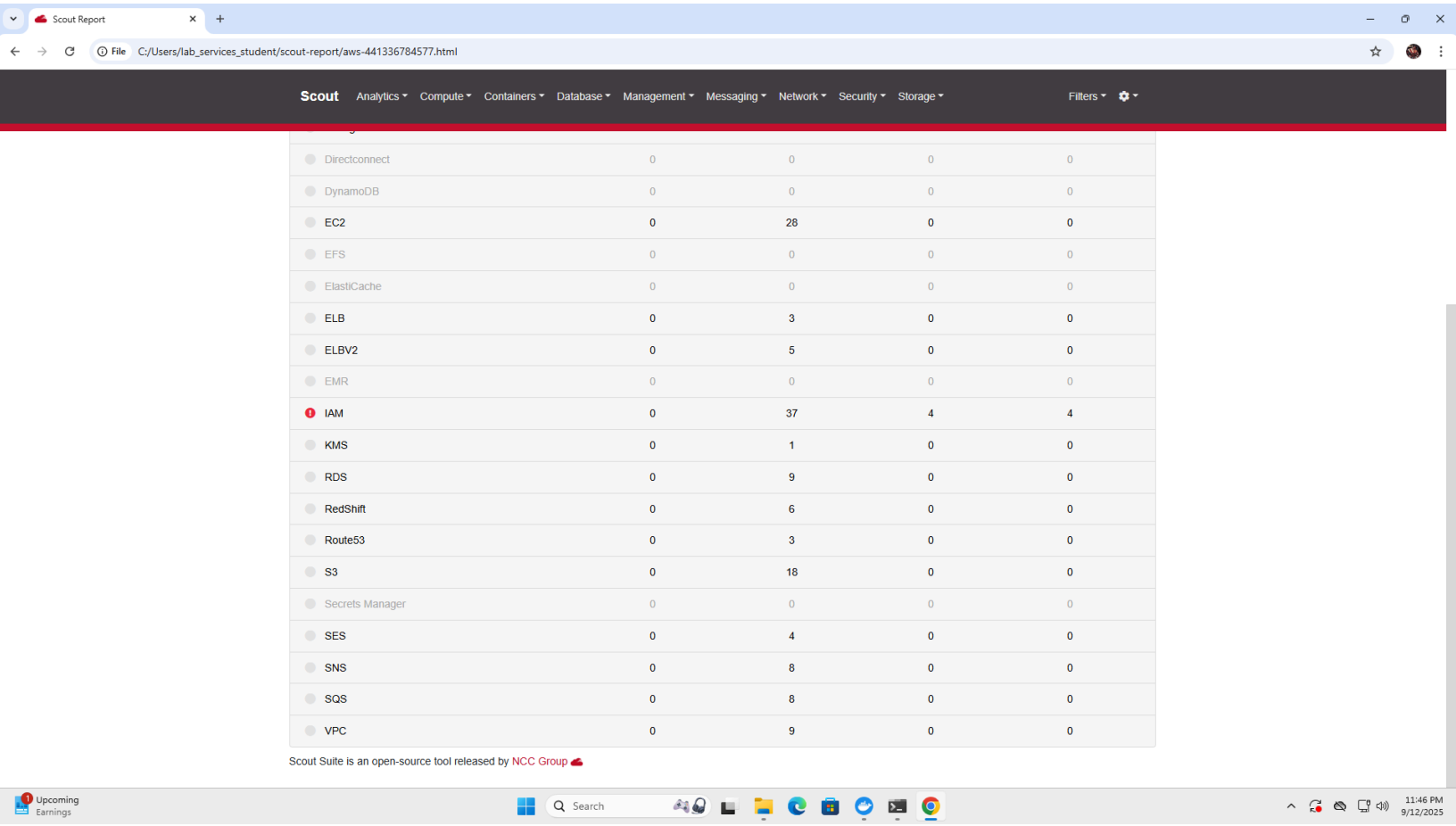
Overall, MobSF provides fast, actionable signal with very low setup cost. I recommend adopting it as a first line automated control. For example, require non debuggable, release signed builds, set allowBackup to false, remove hard coded secrets, and set a minimum score threshold. Use it alongside dynamic testing and targeted manual review for a balanced, bank grade AppSec workflow.

B. Evidence of ScoutSuite implementation with screenshots

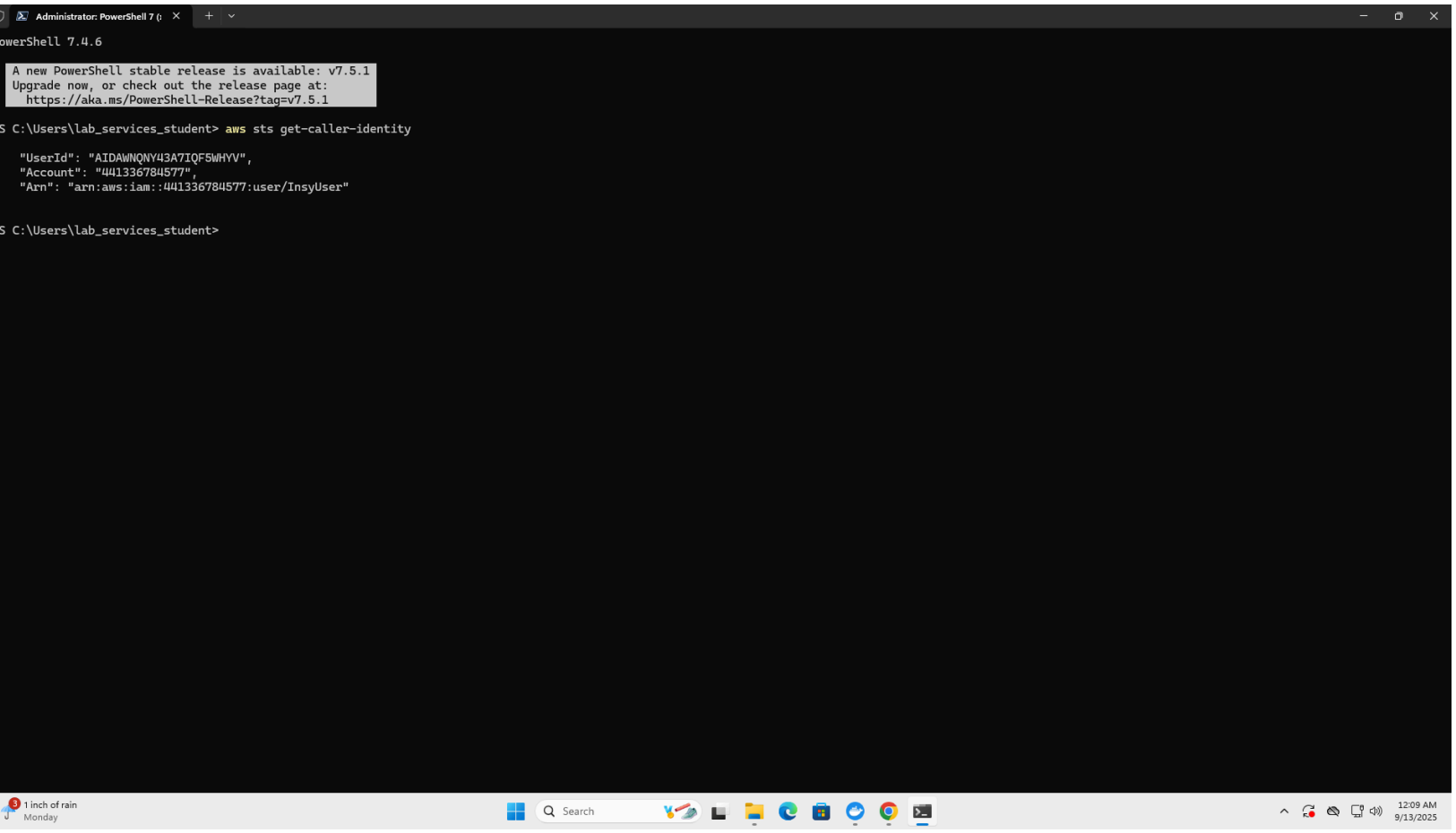
Scout report dashboard



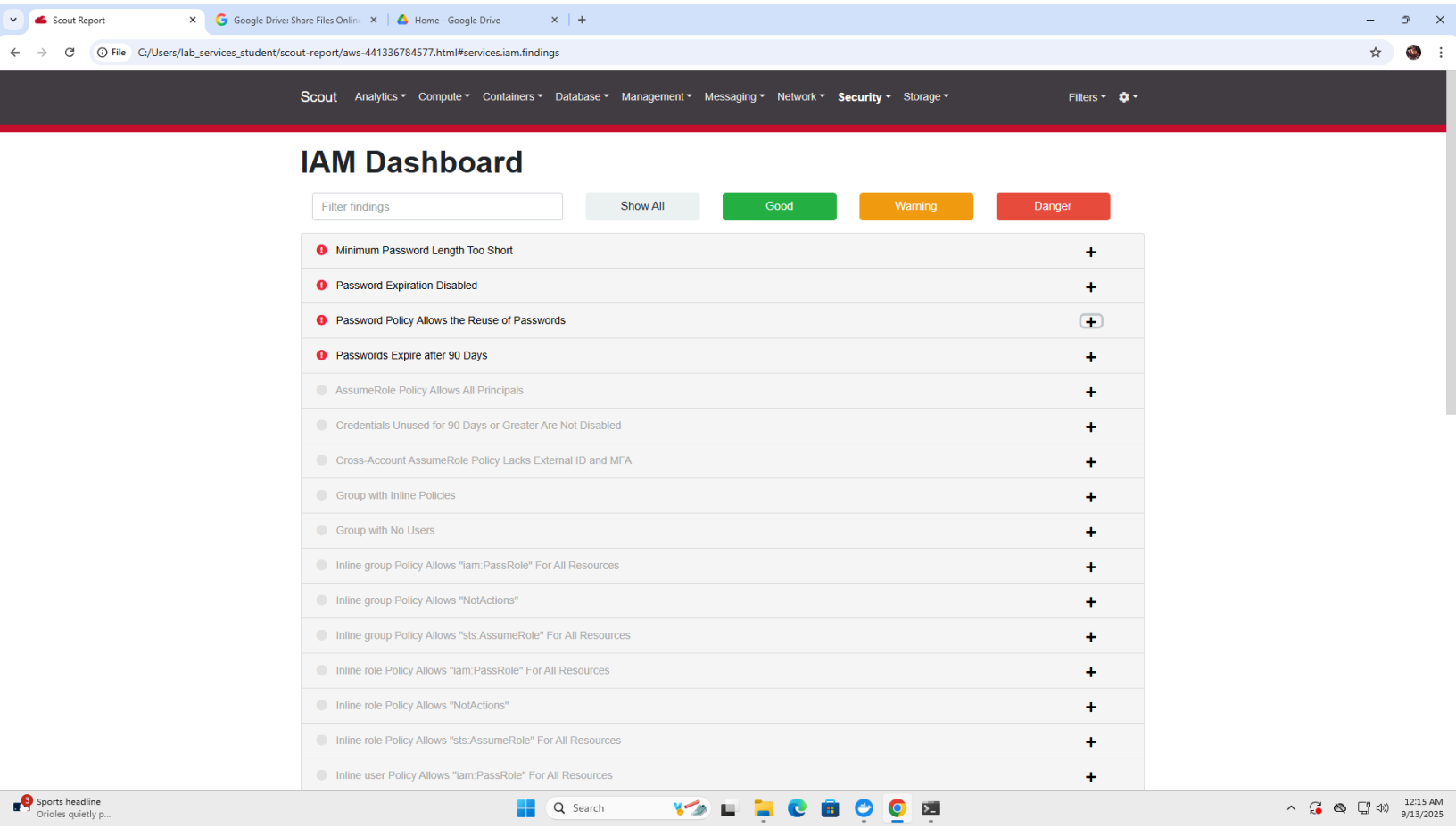
Bottom of scout report dashboard



Powershell command that shows that an aws user was created



IAM Dashboard



ChatGPT ScoutSuite Mini-Report — AWS Account 441336784577

1) Scope & Objective

Run ScoutSuite against the AWS lab account to identify misconfigurations and produce a concise remediation plan suitable for CR sign-off.

2) How We Ran It (Reproducible)

Runner: Docker
Command:

docker run -it --rm -v "${env:USERPROFILE}\.aws:/root/.aws:ro" -v "${PWD}\scout-report:/root/scout-report" rossja/ncc-scoutsuite:latest scout aws --no-browser --report-dir /root/scout-report

Output: HTML report in ./scout-report/ (file name includes `aws` and timestamp).

3) Summary Dashboard (High-Level)

- Most AWS services show 0 resources / 0 findings (fresh/clean account).
- IAM: 4 findings across 37 checks (account-level hygiene).

4) Key Findings (IAM)

Copy the *exact* rule names from the report: Security → IAM → Findings.

1. [F1: — likely Root account MFA not enabled]
   Risk: High — Unprotected root access can lead to total account compromise.
   Fix: Enable MFA on the root user; remove any root access keys.

2. [F2: — likely Weak or Missing Password Policy]
   Risk: Medium — Increases brute-force and credential-stuffing risk.
   Fix: Set strict policy: length ≥ 14, complexity (ULNS), prevent reuse (≥24), optional rotation ≤ 90 days.

3. [F3: — likely MFA not enforced for IAM console users]
   Risk: Medium — Stolen credentials can be used without a second factor.

Fix: Enforce MFA for all console users; add conditional checks (e.g., aws:MultiFactorAuthPresent).

4. [F4: — likely Access Key Hygiene/Rotation]
   Risk: Medium — Long-lived keys increase blast radius if leaked.
   Fix: Remove unused keys; rotate ≤ 90 days; prefer roles and short-lived credentials.

5) Prioritized Remediation Plan (Do Now → Next)

Do Now (24–48h):

1. Enable root MFA; verify no root access keys exist.
2. Enforce account-wide password policy (ULNS, ≥14 chars, reuse prevention).
3. Enroll MFA for all IAM users with console access.

Next (This Week):

4. Audit access keys; remove stale; rotate active; migrate to role-based auth.
5. Add CI guardrails: periodic ScoutSuite run; send deltas to Slack/Email.

6) Conclusion

The account is largely empty (minimal attack surface), but IAM hygiene needs attention. Applying the above remediations will likely reduce findings to zero on re-scan and meet baseline expectations for lab/prototype environments.

Do I support the use of the tool ScoutSuite or not?

I support using ScoutSuite in our workflow. It was quick to run with Docker, produced a clear HTML report, and immediately highlighted account level risks in our AWS environment. In our scan the tool focused us on IAM hygiene with four findings, which is exactly where a new or lightly used account is most vulnerable. That kind of fast signal is valuable for both students and teams who need evidence they can act on without deep cloud expertise.

ScoutSuite is best as a snapshot auditor rather than a full security platform. It does not replace continuous monitoring, detective controls, or alerting. You still need native AWS services such as Security Hub, Config, CloudTrail, and IAM Access Analyzer, along with least privilege policies and MFA. It also relies on the permissions of the credentials you provide, so weak permissions can hide issues, and overly broad permissions are a separate risk that must be managed.

Overall, the benefits outweigh the limits. I recommend adopting ScoutSuite as a lightweight, repeatable check after environment changes and before assessments, storing each HTML report for trend tracking. Pair it with strong IAM policies, MFA enforcement, and native AWS guardrails, and you will get a clear, practical view of misconfigurations with minimal setup effort.

Reference list

Android Developers (2025) *Network security configuration*. [Online] Available at: https://developer.android.com/privacy-and-security/security-config (Accessed: 15 September 2025).

AWS (2022) *AWS Key Management Service best practices*. [Online] Available at: https://docs.aws.amazon.com/kms/latest/developerguide/best-practices.html (Accessed: 15 September 2025).

AWS (2025) *How AWS Shield and Shield Advanced work*. [Online] Available at: https://docs.aws.amazon.com/waf/latest/developerguide/ddos-overview.html (Accessed: 15 September 2025).

Barker, E. (2020) *Recommendation for Key Management: Part 1 – General (SP 800-57 Part 1 Rev. 5)*. National Institute of Standards and Technology. [Online] Available at: https://csrc.nist.gov/pubs/sp/800/57/pt1/r5/final (Accessed: 11 September 2025).

Biryukov, A., Dinu, D. and Khovratovich, D. (2016) *Argon2: The memory-hard function for password hashing and other applications*. University of Luxembourg. [Online] Available at: https://www.cryptolux.org/index.php/Argon2 (Accessed: 11 September 2025).

Biryukov, A., Dinu, D. and Khovratovich, D. (2021) *Argon2 memory-hard function for password hashing (RFC 9106)*. Internet Research Task Force. [Online] Available at: https://www.rfc-editor.org/rfc/rfc9106 (Accessed: 15 September 2025).

Cloudflare (2023) *What is OCSP stapling?* [Online] Available at: https://www.cloudflare.com/learning/ssl/what-is-ocsp-stapling/ (Accessed: 11 September 2025).

Gamache, M. and Wall, W. (2023) *Certificate and public key pinning*. OWASP. [Online] Available at: https://owasp.org/www-community/controls/Certificate_and_Public_Key_Pinning (Accessed: 13 September 2025).

Gupta, B.B. and Gupta, R. (2019) 'Clickjacking attack detection and prevention techniques: A comprehensive survey', *Journal of Information Security and Applications*, 46, pp. 72–84.

Halfond, W.G.J., Viegas, J. and Orso, A. (2006) 'A classification of SQL-injection attacks and countermeasures', *Proceedings of the IEEE International Symposium on Secure Software Engineering*, pp. 13–15.

Hunt, T. (n.d.) *Pwned Passwords*. Have I Been Pwned. [Online] Available at: https://haveibeenpwned.com/Passwords (Accessed: 15 September 2025).

IETF (2013) *RFC 6961: The Transport Layer Security (TLS) Multiple Certificate Status Request Extension (OCSP stapling)*. Internet Engineering Task Force. [Online] Available at: https://datatracker.ietf.org/doc/html/rfc6961 (Accessed: 15 September 2025).

ISO (2014) *ISO 9362: Financial services — Banking telecommunication messages — Business identifier code (BIC)*. International Organization for Standardization. [Online] Available at: https://www.iso.org/standard/60390.html (Accessed: 15 September 2025).

NIST (2020) *Digital Identity Guidelines: Authentication and Lifecycle Management (SP 800-63B)*. National Institute of Standards and Technology. [Online] Available at: https://pages.nist.gov/800-63-3/sp800-63b.html (Accessed: 15 September 2025).

Orenstein, G. (2022) 'How long should my password be?', *Bitwarden Blog*. [Online] Available at: https://bitwarden.com/blog/how-long-should-my-password-be/ (Accessed: 15 September 2025).

OWASP (2021) *OWASP Top 10: The ten most critical web application security risks*. Open Web Application Security Project. [Online] Available at: https://owasp.org/Top10/ (Accessed: 11 September 2025).

OWASP (2023a) *Session Management Cheat Sheet*. OWASP Foundation. [Online] Available at: https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html (Accessed: 15 September 2025).

OWASP (2023b) *Cross-Site Request Forgery Prevention Cheat Sheet*. OWASP Foundation. [Online] Available at: https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html (Accessed: 15 September 2025).

OWASP (2023c) *Clickjacking Defense Cheat Sheet*. OWASP Foundation. [Online] Available at: https://cheatsheetseries.owasp.org/cheatsheets/Clickjacking_Defense_Cheat_Sheet.html (Accessed: 15 September 2025).

OWASP (2023d) *SQL Injection Prevention Cheat Sheet*. OWASP Foundation. [Online] Available at: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html (Accessed: 15 September 2025).

OWASP (2023e) *Cross-Site Scripting Prevention Cheat Sheet*. OWASP Foundation. [Online] Available at: https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html (Accessed: 15 September 2025).

OWASP (2023f) *Transport Layer Security Cheat Sheet*. OWASP Foundation. [Online] Available at: https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Security_Cheat_Sheet.html (Accessed: 15 September 2025).

OWASP (2023g) *HTTP Strict Transport Security Cheat Sheet*. OWASP Foundation. [Online] Available at:

https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Strict_Transport_Security_Cheat_Sheet.html (Accessed: 15 September 2025).

OWASP (2023h) *HTTP Security Response Headers Cheat Sheet*. OWASP Foundation. [Online] Available at: https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Headers_Cheat_Sheet.html (Accessed: 15 September 2025).

OWASP (2023i) *Pinning Cheat Sheet*. OWASP Foundation. [Online] Available at: https://cheatsheetseries.owasp.org/cheatsheets/Pinning_Cheat_Sheet.html (Accessed: 15 September 2025).

OWASP (2023j) *Web Service Security Cheat Sheet*. OWASP Foundation. [Online] Available at: https://cheatsheetseries.owasp.org/cheatsheets/Web_Service_Security_Cheat_Sheet.html (Accessed: 15 September 2025).

OWASP (2023k) *Denial of Service Cheat Sheet*. OWASP Foundation. [Online] Available at: https://cheatsheetseries.owasp.org (Accessed: 15 September 2025).

Rescorla, E. (2018) 'The Transport Layer Security (TLS) Protocol Version 1.3', *RFC 8446*. Internet Engineering Task Force. [Online] Available at: https://www.rfc-editor.org/rfc/rfc8446 (Accessed: 11 September 2025).

Singh, A. (2025) 'How password hashing works: PBKDF2, Argon2 & more', *Medium*. [Online] Available at: https://medium.com/@aannkkiittaa/how-password-hashing-works-pbkdf2-argon2-more-95cee0cd7c4a (Accessed: 15 September 2025).

SWIFT (n.d.) *IBAN Registry*. Society for Worldwide Interbank Financial Telecommunication. [Online] Available at: https://www.swift.com/standards/data-standards/iban (Accessed: 15 September 2025).

Zargar, S.T., Joshi, J. and Tipper, D. (2013) 'A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks', *IEEE Communications Surveys & Tutorials*, 15(4), pp. 2046–2069.

AI usage

AI was used for writing and research assistant to: clarify setup steps for MobSF and ScoutSuite; draft and refine security sections (input security, data-in-transit, hardening); format in-text citations and a reference list;

Link to chat: https://chatgpt.com/share/68c86674-248c-800e-8133-4e537bdd56bb