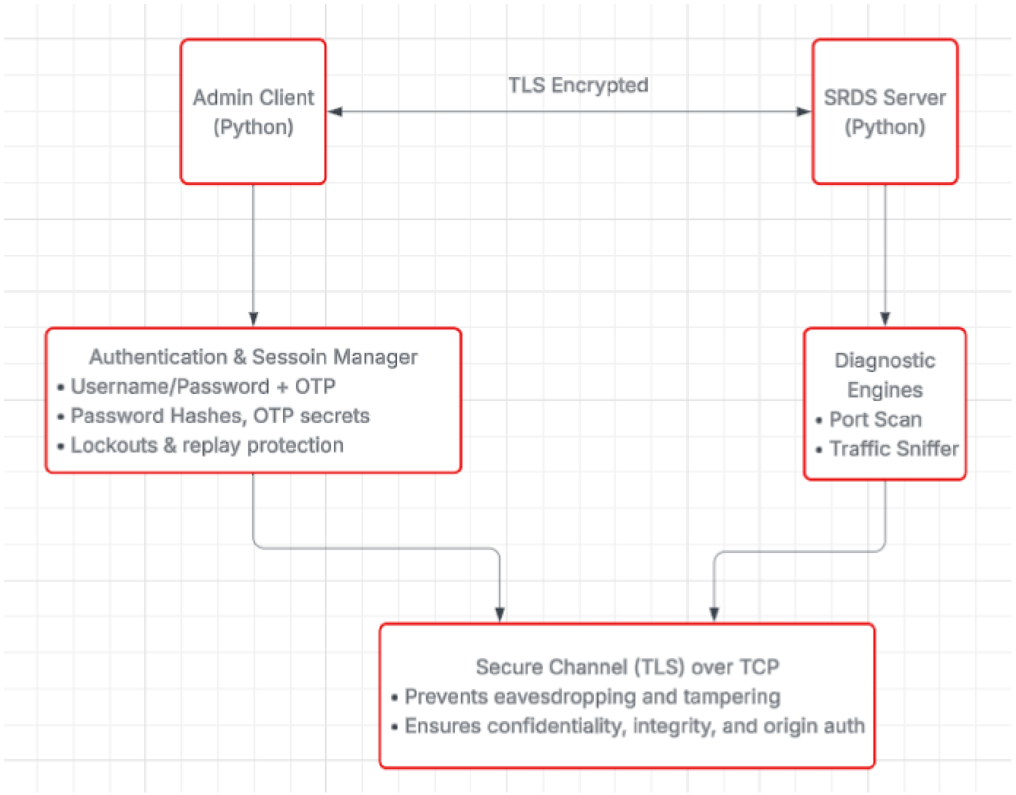
FSCT 8561 – Midterm

By: Jose Bangate, A01271709

System Design & Threat Modelling

Part A: Architecture Design


Admin Client (Python)
TLS Encrypted
SRDS Server (Python)
Authentication & Sessoin Manager
• Username/Password + OTP
• Password Hashes, OTP secrets
• Lockouts & replay protection
Diagnostic Engines
• Port Scan
• Traffic Sniffer
Secure Channel (TLS) over TCP
• Prevents eavesdropping and tampering
• Ensures confidentiality, integrity, and origin auth

Client & Server Components:

To ensure confidentiality and integrity, the client in the SRDS architecture is a Python socket application that initiates a secure connection to the server via a TLS-encrypted TCP channel. The client prompts credentials from the administrator, submits a hashed password and a time-based one-time password (OTP), and only delivers diagnostic commands once authentication has been completed. A corresponding Python service on the server side handles a session for every client, receives incoming remote diagnostic operations like network scans and packet inspection, validates OTPs, enforces lockout policies for repeated failures, and checks password hashes. This secure channel is used for all client-server communication, and the server keeps track of who is authenticated and what commands they are allowed to execute.

The Gateway Logic:

The SRDS uses a direct secure connection between administrator clients and the server instead of adding a gateway or proxy between the client and the server. By keeping the diagnostic pathway simple, this solution reduces complexity and does away with the requirement for extra infrastructure components that would need to be trusted and secured. There is no need for external gateways or middleboxes to mediate traffic because all clients are authorized administrative tools, and communications take place within an encrypted TLS session. Potential attackers are prevented from introducing themselves into an extra layer of intermediate that could serve as an attack surface by this direct paradigm, which guarantees that diagnostic commands and results only move between the authenticated client and the server.

Justification:

Since the authentication process and subsequent command exchange necessitate tracking client information over numerous encounters, a stateful communication paradigm was selected for SRDS. Maintaining this session state allows the server to implement policies like replay protection. Lockouts and rate-limited activities on a per-session basis. Authentication is a multi-step process that includes initial credential verification and OTP validation. Security control enforcement would be made more difficult by a stateless method, which handles each request separately without retaining context. This would require external session tokens or frequent reauthentication. Thus, within the secure diagnostic service, the stateful approach facilitates both safe authentication and effective session management.

Part B: Adversarial Threat Model

Asset	Threat	Attack Vector	Impact	Mitigation
Admin Credentials	Replay Attack	Sniffing socket traffic	Unauthorized access	Use Challenge-response + nonce, enforce encryption
Client-Server Cannel	Main-in-the-middle	Unencrypted socket traffic interception	Credential theft, command tampering	TLS/SSL encryption with certificate verification

Diagnostic Scanning Service	Unauthorized scanning commands	Forged client requests	Network misuse, denial of service	Authenticate the client before accepting scan commands
MFA (OTP secrets)	OTP Secret Disclosure	Sniffing or local storage compromise	OTP bypass and account takeover	Store OTP secrets securely, avoid plaintext exposure
Packet Payloads	Sensitive Data Leakage	Unencrypted protocols (HTTP) observed in traffic	Exposure of user data	Use HTTP and encrypted payloads

Part C: Secure Protocol Design

Message Format:

The Application-layer message format is JSON rather than TLS. Every message exchanged between the client and the server is a JSON object that is framed with newline (\n) delimiters and encoded as UTF-8.

Authentication Handshake (Challenge-Response):

To avoid replay attacks and confirm client identification, the authentication handshake employs a secure challenge-response technique that never transmits plaintext secrets.

Adversarial Defense:

The protocol uses several techniques to stop captured packets from being replayed:

- Nonce & Timestamp:
  A nonce and a new timestamp are used in each handshake to guarantee that every attempt is distinct. A valid response must fall inside a specific time frame (e.g. +/-30 seconds) and match the server's challenge nonce. Since the nonce changes every time, reusing a previous hash will not work.

- TLS Encryption:
  An attacker cannot access, alter, or replay ciphertext at the transport layer because all JSON messages are transmitted over TLS, which ensures confidentiality and integrity.

- Challenge-Response Without Plaintext:
  The client never transmits the TOTP secret or the raw password. Rather, it transmits a hash that combines server challenge data with the password hash. Since the

nonce/timestamp will be invalid on a replay attempt, this hash cannot be used again, even if it is captured by an attacker.

Part D: Component-Level Design

MFA Module:

The MFA module uses a saved salted SHA-256 hash to authenticate the user after receiving a username and password hash from the client. It also processes a TOTP value created by PyOTP using the user's special secret after verifying the password. A straightforward authentication result showing success or failure is the output. Enforcing lockouts following repeated failures, verifying TOTP within a brief window to tolerate slight clock drift, hashing passwords to ensure that plaintext is never stored or transmitted, and only using an encrypted transport protocol (TLS) to prevent credentials or OTPs from being intercepted are some examples of security controls.

Recon Module:

The Recon Module uses Python-nmap to conduct TCP scans against the designated network node after receiving diagnostic scan requests from authenticated clients, including a target host address and port range. Its output includes error warnings if the scan is unsuccessful or unauthorized, as well as structured results that list open ports and services found. Rate limiting prohibits exploitation of scanning capabilities against internal or external networks, input validation stops malformed or harmful targets, and key security controls guarantee that only authenticated and authorized sessions can initiate scans. Furthermore, recon results are cleaned to prevent unauthorized clients from learning about sensitive internal information.

Detection Module:

The Detection Module uses Scapy to record and examine live or PCAP traffic according to predefined criteria or filters provided by the authenticated client. Protocol counts, packet summaries, and alerts when questionable patterns (such as an excessive number of packets from a single source) are detected are among the outputs. Inputs include packet filter rules, interface names, or PCAP filenames. This module's security controls make sure that captured packet payloads containing potentially sensitive content are processed in memory rather than being stored in unprotected logs, limit the amount of captured data to prevent resource exhaustion, and restrict packet sniffing to authorized operational environments.

Negative Space Decision:

The deliberate failure to save complete packet payloads for encrypted traffic, such HTTPS streams, is one kind of data collection omission. The system minimizes the amount of sensitive data at rest, reducing the attack surface and the risk associated with storing entire packet captures by minimizing the retention of encrypted content that cannot be effectively evaluated without decryption.

Part E: Minimal Implementation

SEE GITHUB LINK FOR MY CODE IMPLEMENTATION

Part F: Security Analysis & Reflection

1. The server's TCP stack itself will be the first thing to fail if an attacker uses SYN flood to target your system, not your application logic (authentication or scanning).
2. The TOTP secret that is kept on the client side is the possession factor in my MFA design. If an attacker manages to obtain the secret, they can take advantage of this. For instance, the attacker "possesses" the second factor and can create valid TOTP codes endlessly if the client's device is compromised or the OTP secret leaks (for instance, by being retrieved from setup information or kept insecurely).
3. If my TLS-encrypted communication travels via a translation gateway (like a WAP proxy), the vulnerability occurs at the gateway where TLS is ended rather than at the client or final server.