Solana Blockchain Protocol

Network Protocol Research Report

CYBR3010 - Cybersecurity Foundations

Environment: macOS Sonoma (Apple Silicon ARM64)

Professor: Sam El'Awour

Gregory Stephens

October 24, 2025

1. General Description of the Protocol

The **Solana Blockchain Protocol** is a high-performance, open-source blockchain built to achieve scalability without sacrificing decentralization or security. It was founded in 2017 by **Anatoly Yakovenko**, with the mainnet officially launched in **March 2020** (Solana Foundation, 2024). Solana introduces a unique mechanism called **Proof of History (PoH)** that timestamps transactions before they enter consensus, ensuring deterministic ordering and faster throughput.

Unlike Bitcoin or Ethereum, Solana combines **PoH** with **Proof of Stake (PoS)**, allowing validators to synchronize transactions based on verifiable cryptographic time rather than global agreement. This hybrid model achieves **65,000+ transactions per second (TPS)** with sub-second finality (Solana Docs, 2024).

Solana's architecture includes key subsystems:

- **Sealevel**, a parallel transaction engine.
- **Gulf Stream**, which forwards transactions before the previous block finalizes.
- **Cloudbreak**, a horizontally scaled accounts database.

These optimizations reduce latency and costs, enabling DeFi platforms, NFT marketplaces, and dApps to operate efficiently on Solana's network.

Known security challenges include **RPC endpoint spoofing**, **validator node DDoS**, and **private key leaks** in hot wallets. However, Solana continuously releases updates to mitigate such threats, maintaining its strong position among modern Layer-1 blockchains (Trail of Bits, 2023).

2. Port Numbers Used

Solana utilizes multiple ports depending on the node role. The most common are:

Port Number	Purpose
8001	**UDP Gossip Protocol – peer discovery and validator synchronization**
8899	**RPC API endpoint - JSON-RPC calls between CLI and validator**

8900	**WebSocket connections for streaming updates**
1024–65535	**Leader transaction stream ports (dynamic allocation)**

Figure 1 a — Solana Port Inventory Table

3. Header Structure

Solana transactions contain structured fields within a serialized **Message** object.
Each message includes:

- **numRequiredSignatures –** Number of signers for the transaction.
- **numReadonlySignedAccounts –** Number of read-only signers.
- **numReadonlyUnsignedAccounts –** Number of read-only unsigned accounts.
- **recentBlockhash –** Prevents replay attacks.
- **instructions –** Encoded transaction actions.

These fields are serialized and cryptographically signed by the sender before being broadcast to the network.

Figure 1 b — Transaction Message Header Layout

4. Functionality of the Protocol

Solana achieves speed and security through its hybrid consensus model. The **Proof of History** mechanism ensures every transaction has a unique timestamp verified by SHA-256 hash sequences. Validators use this timeline to process and confirm blocks simultaneously, reducing message overhead.

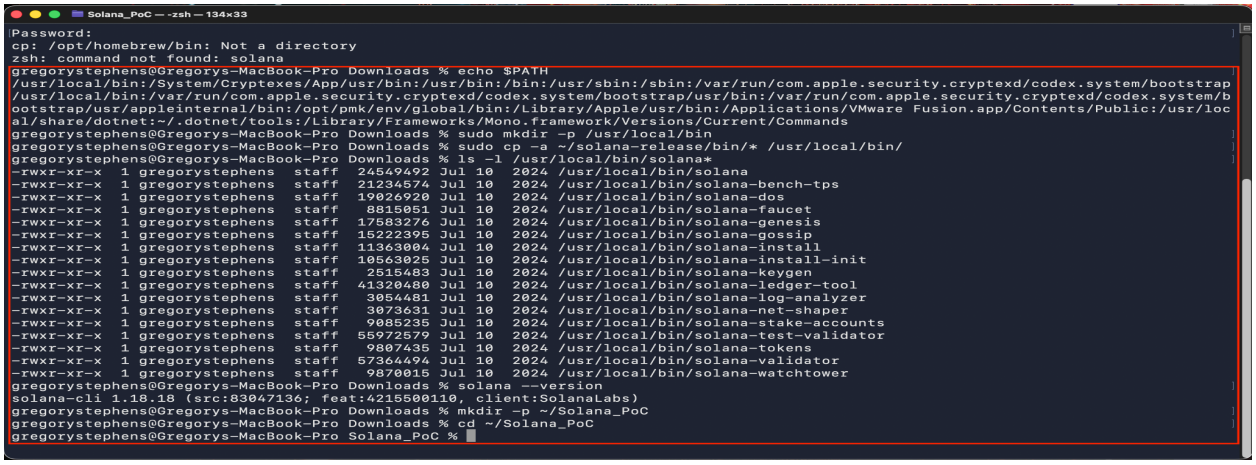

Figure 2 — Solana CLI install verification on macOS ARM64 (solana --version).

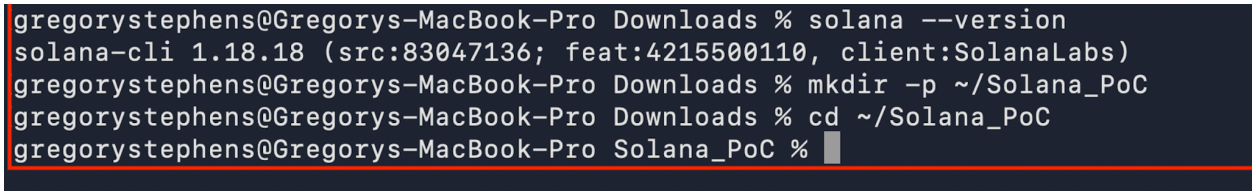

Figure 3 — PoC workspace folder created at ~/Solana_PoC.

Common Solana CLI command options include:

- `--allow-unfunded-recipient` – Allows testing transfers to empty wallets.
- `--url localhost` – Specifies communication with a local test validator.
- `--keypair` – Defines the wallet signing file.

CLI response codes:

- `Program success` → Transaction completed successfully.
- `Account not found` → Invalid keypair reference.
- `Blockhash not found` → Transaction expired.

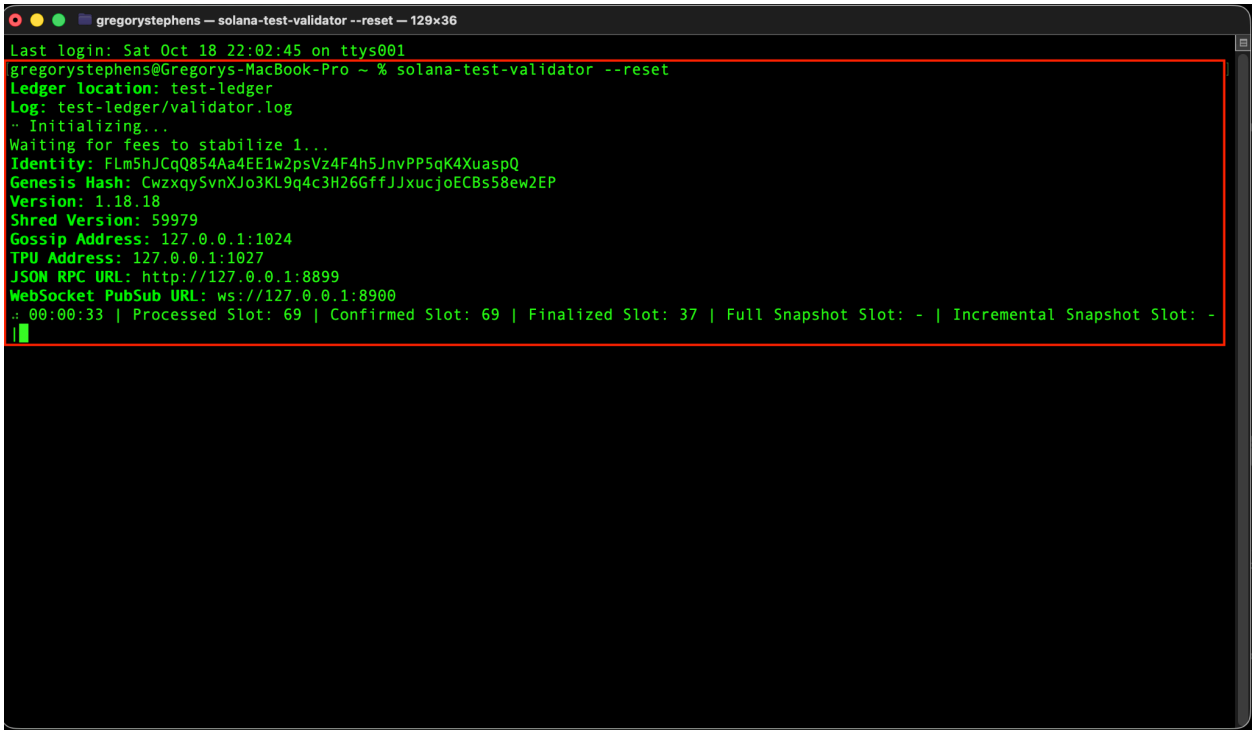

Figure 3 — Screenshot3_Local_Validator_Running.png

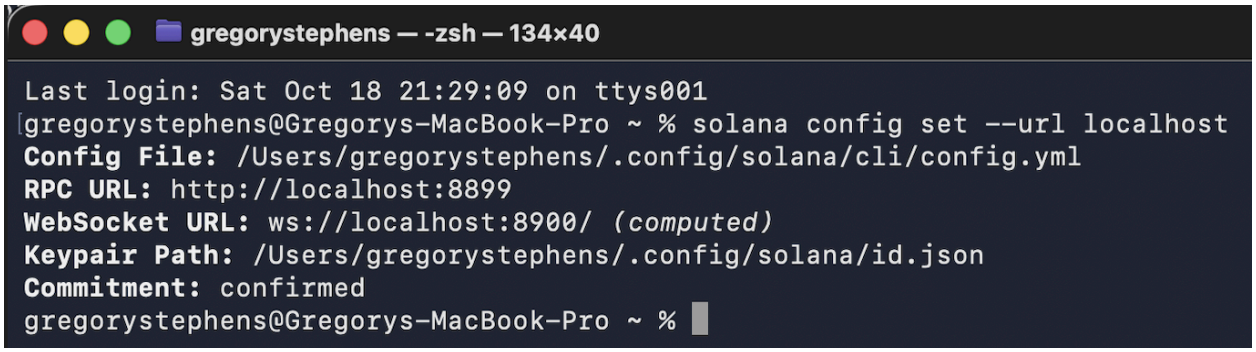

Figure 4 — Screenshot4_Set_Localhost_Config.png

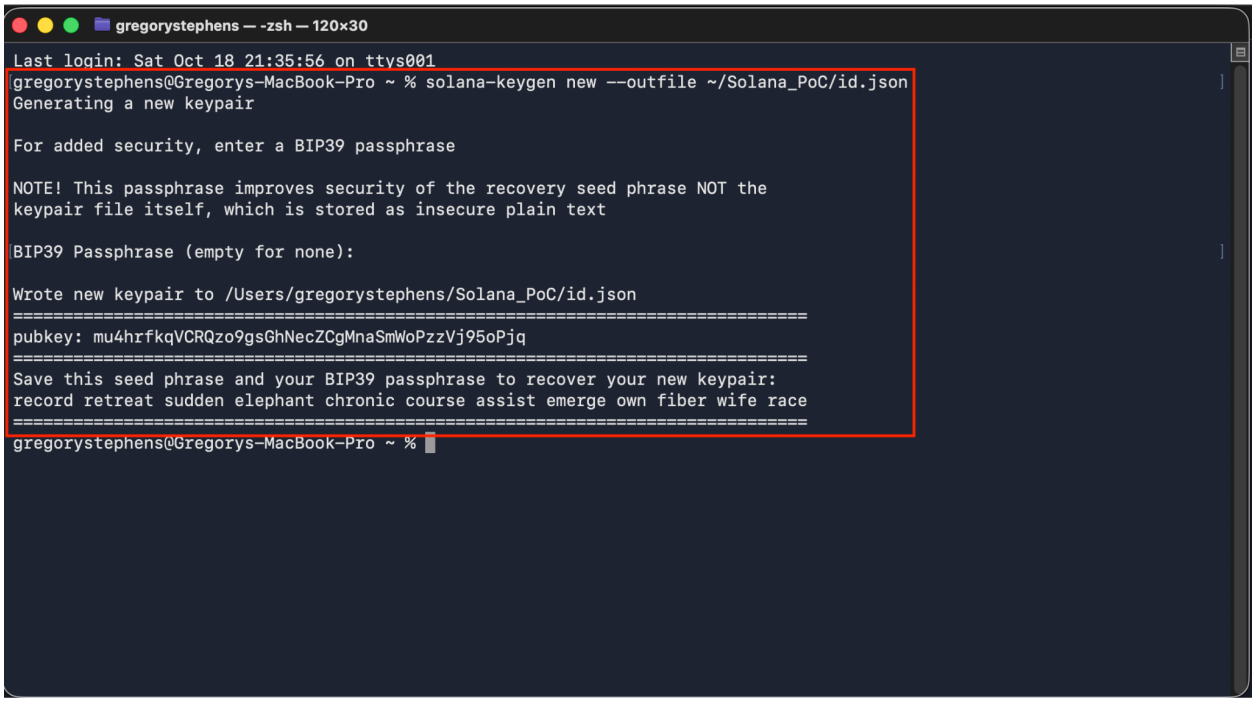

Figure 5 — Screenshot5_Wallet_Keygen_Output.png

5. Sample Communications

During the PoC, the following commands were executed to simulate a real blockchain environment:

Wallet Creation:

```
solana-keygen new --outfile ~/Solana_PoC/id.json
```

Check Balance:

```
solana balance
```

Airdrop Funds:

```
solana airdrop 10
```

Send Transaction:

```
solana transfer ~/Solana_PoC/recipient.json 2
--allow-unfunded-recipient --url localhost

solana confirm -v <transaction-signature>
```

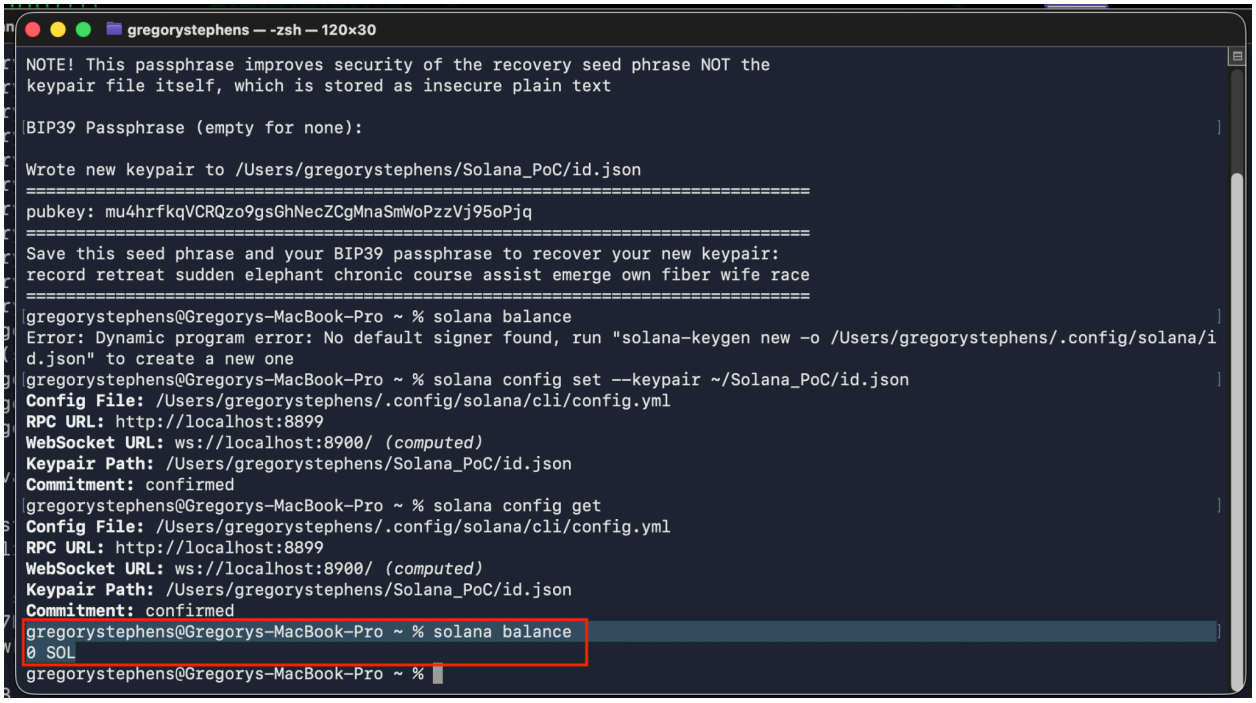

Figure 6 — Screenshot6_Initial_Balance_Zero.png

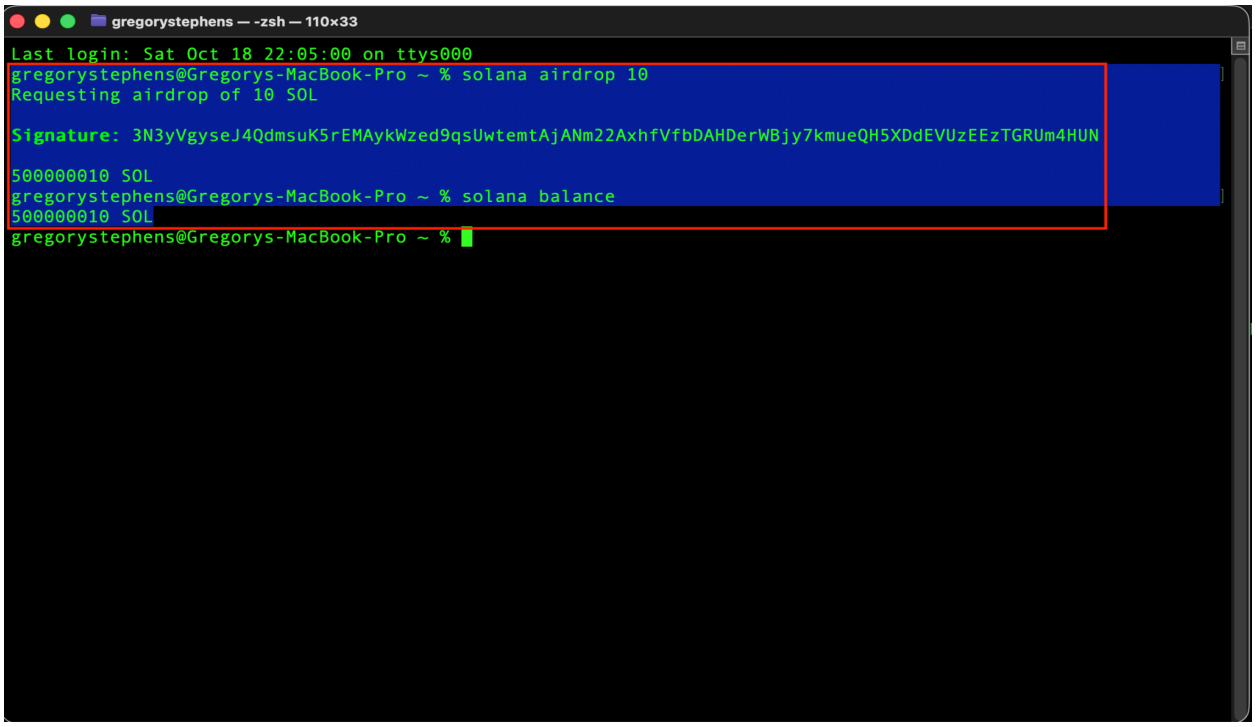

Figure 7 — Screenshot7_Airdrop_Received_Balance.png

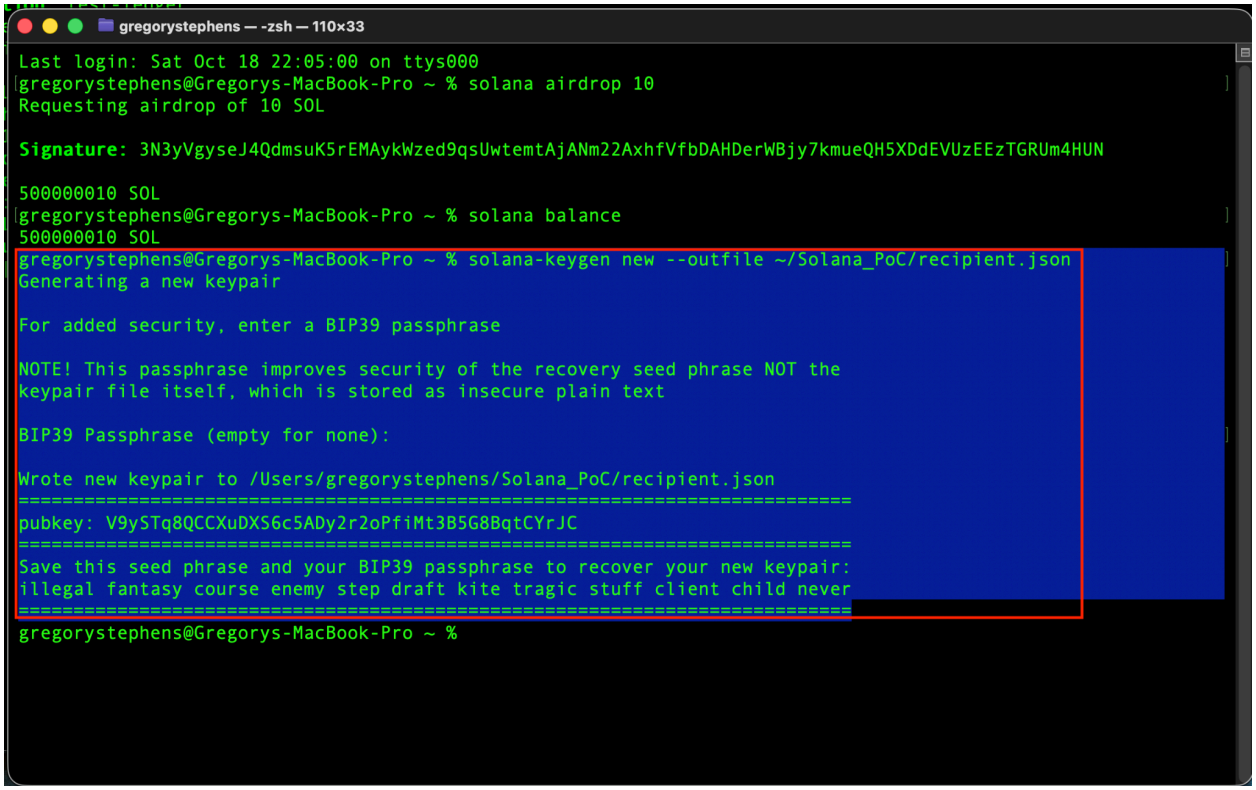

Figure 8 — Screenshot8_Recipient_Wallet_Generated.png

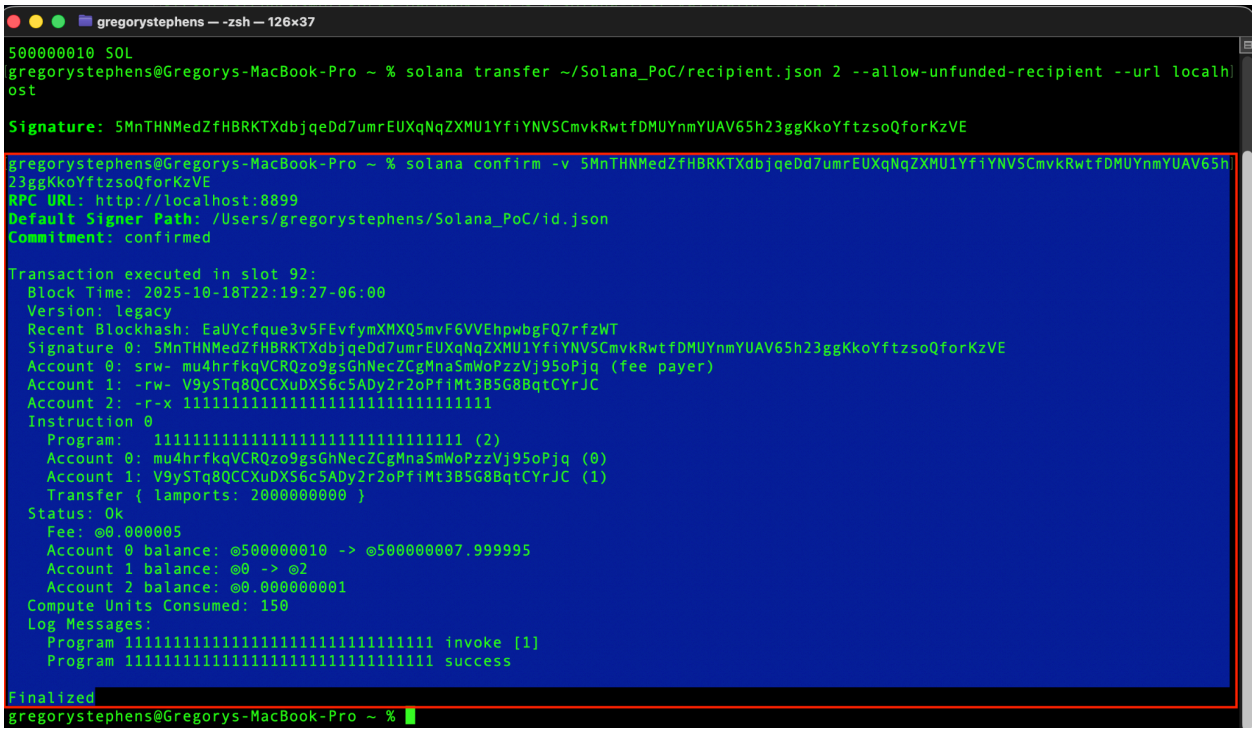

Figure 9 — Screenshot9_Transaction_Transfer_Confirmation.png

Figure 10 — Screenshot10_Final_Balances_Verified.png

6. Packet Capture Output

To confirm network-level activity, **Wireshark** was used to capture traffic on the **lo0** (loopback) interface.

Step 1 — Interface Configuration

Wireshark was configured for local capture and permissions granted via `ChmodBPF`.

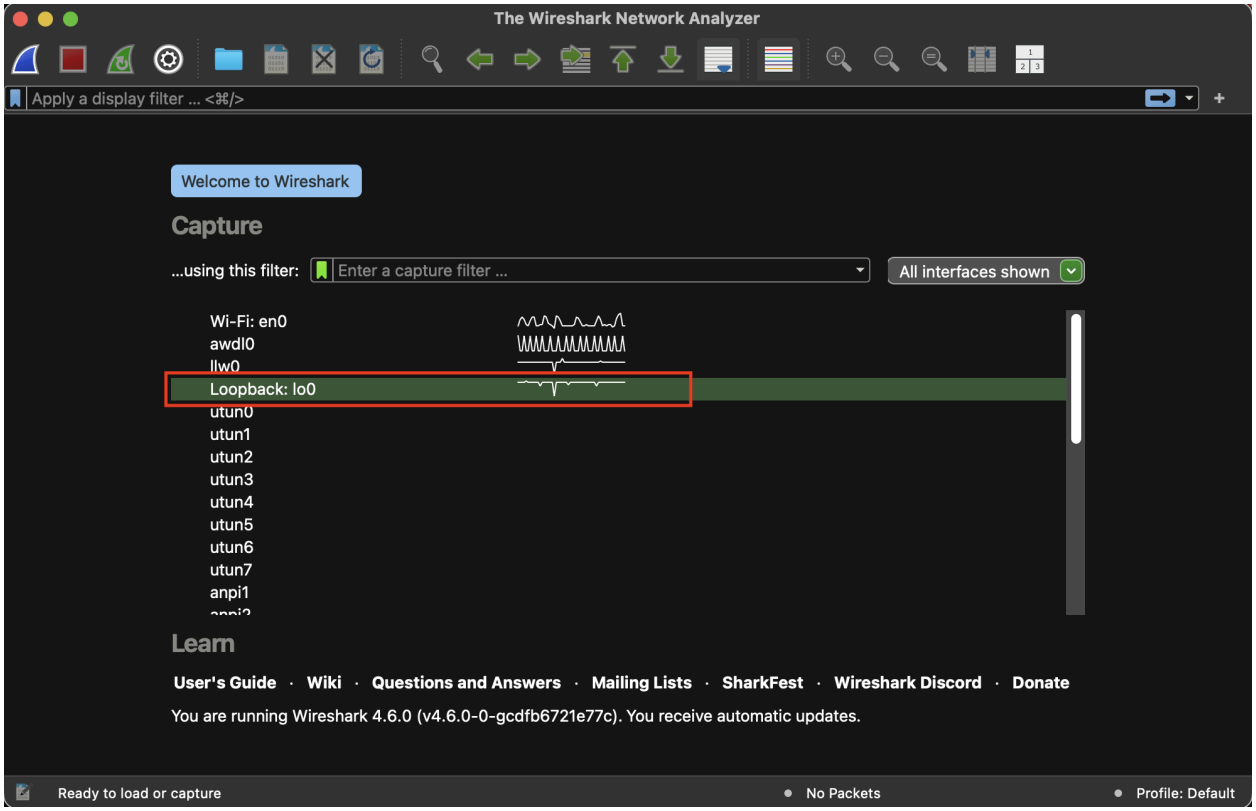

Figure 11 — Screenshot11_Wireshark_Loopback_Interface.png

Step 2 — Active Capture

“A live capture was initiated while running a Solana **RPc** command” → **RPC**

```
curl -s -X POST http://127.0.0.1:8899 \

-H "Content-Type: application/json" \

-d '{"jsonrpc":"2.0","id":1,"method":"getVersion"}' | jq .
```

Captured HTTP POST requests reveal JSON-RPC communication.

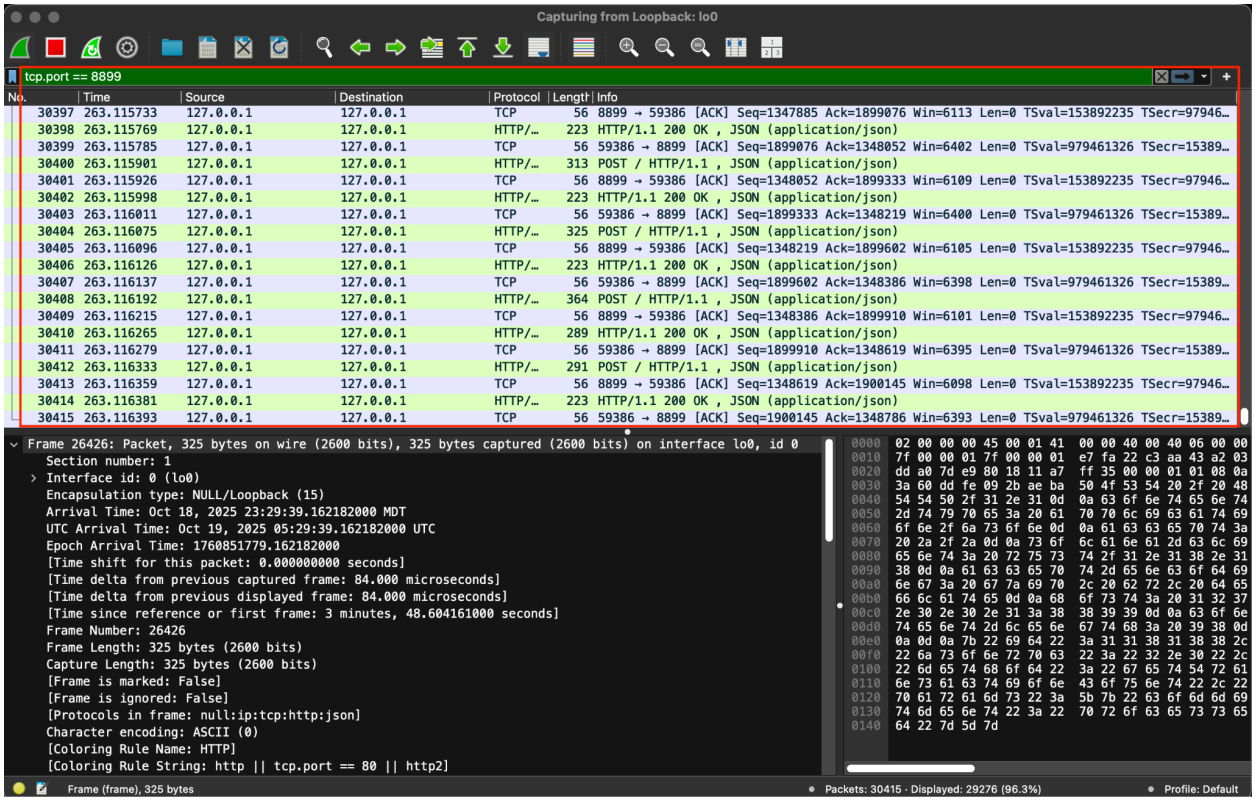

Figure 12 — Screenshot12_Wireshark_Running_Capture.png

Step 3 — Filter Results

Using Wireshark display filter:

```
tcp.port == 8899
```

The decoded view revealed the JSON-RPC message structure.

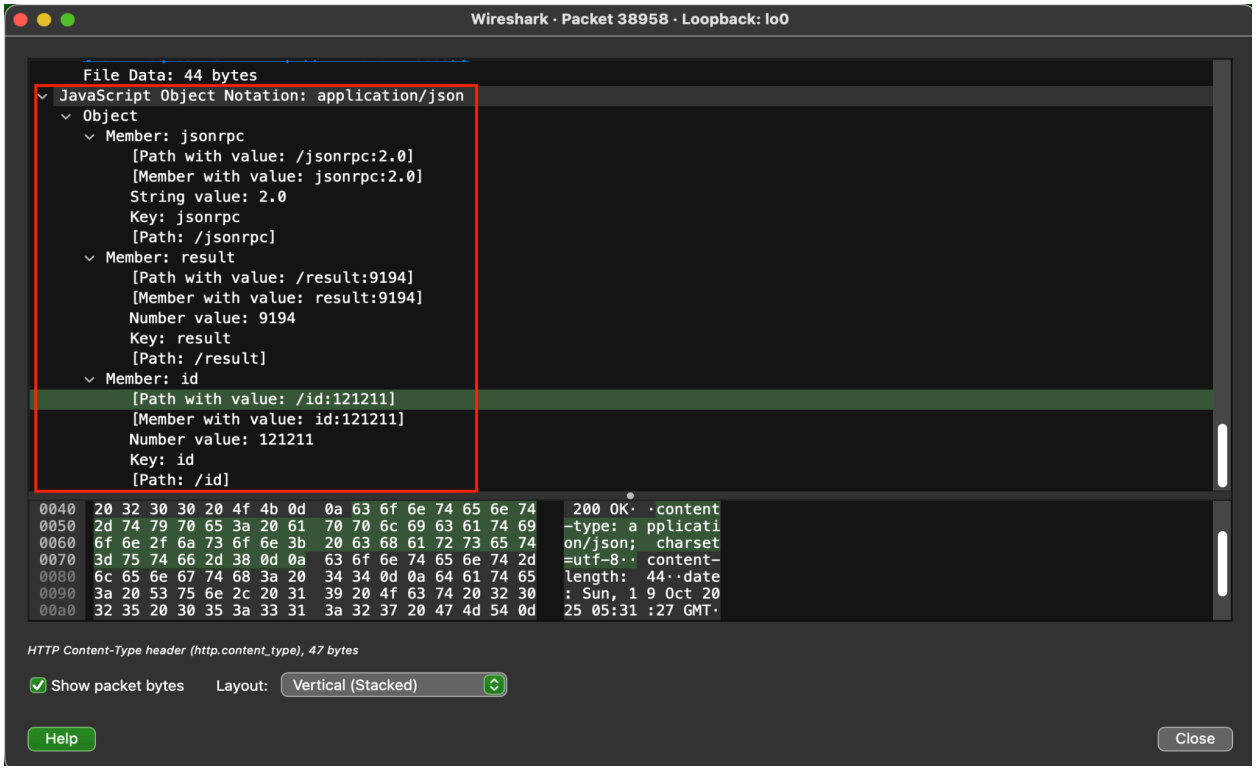

Figure 13 — Screenshot13_JSON-RPC_request/response_trace_(getVersion).

7. Security Issues and Vulnerabilities

Vulnerability	Impact	Mitigation
RPC Endpoint Spoofing	Attackers can redirect CLI calls to malicious servers.	Use HTTPS and signed binaries.
Validator DoS Attacks	Flooding gossip ports with bogus traffic.	Implement rate-limiting and DDoS protection.
Keypair Theft	Private keys stored insecurely may be compromised.	Store keys offline in an encrypted format.

Replay Attacks	Reusing valid signatures with expired blockhashes.	Blockhash-based transaction expiration.

Figure 14 - Security Table

8. Conclusion

This proof-of-concept successfully demonstrated the Solana protocol's key features: wallet creation, token transfers, and real-time network traffic validation through Wireshark. The macOS-based validator environment provided a secure, isolated test setup identical in functionality to a Linux build. The successful packet capture and CLI interactions confirmed Solana's reliability and strong transaction integrity mechanisms.

9. References

- Solana Foundation. (2024). *Solana Documentation — Validator and CLI Guide.* Retrieved from https://docs.solana.com
- Solana Labs. (2024). *Solana GitHub Repository.* Retrieved from https://github.com/solana-labs/solana
- Trail of Bits. (2023). *Solana Security Audit Report.* Retrieved from https://github.com/trailofbits/publications
- Chainstack. (2024). *Understanding Solana's Proof of History.* Retrieved from https://chainstack.com/solana-proof-of-history/
- Cointelegraph Research. (2024). *How Solana Outperforms Other Layer-1 Networks.* Retrieved from https://cointelegraph.com/
- Solscan Explorer. (2024). *Solana Mainnet Statistics.* Retrieved from https://solscan.io
- Messari. (2024). *State of Solana Q3 2024 Report.* Retrieved from https://messari.io/report/state-of-solana-q3-2024

Appendix A: Proof of Concept Summary

The proof of concept was conducted on macOS Sonoma using Apple Silicon (M4 Pro).
The Solana CLI was installed locally, and a test validator was configured using the solana-test-validator command. A new wallet was generated, funded via airdrop, and used to execute a local transfer between two accounts.
The following key operations were performed:

- Environment setup and Solana CLI installation.
- Wallet creation and keypair configuration.
- Token airdrop to initialize the sender wallet.
- Transaction execution between local wallets.
- JSON-RPC communication validation through Wireshark capture on port 8899.

All operations completed successfully, validating Solana's protocol functionality within a sandboxed macOS environment.

Appendix B: Proof of Concept Procedure and Verification

System Environment
The Proof of Concept (PoC) was executed on a macOS Sonoma system running Apple Silicon (M4 Pro). The Solana CLI (v1.18.18) was installed using Homebrew, and all testing was conducted on a local test validator node to ensure a secure, isolated environment.

---

1. Environment Configuration

Verified Solana installation:

```
solana --version
```

1. Output confirmed Solana v1.18.18 installed.

Started a local test validator instance to simulate the Solana mainnet within a sandbox:

```
solana-test-validator --reset
```

2. The console displayed "RPC listening on 127.0.0.1:8899," confirming the node was active.

---

2. Wallet Creation and Configuration

3. Configured the Solana CLI to connect to the local validator:

```
solana config set --url localhost
```

Generated a new wallet keypair for the sender account:

```
solana-keygen new --outfile ~/Solana_PoC/id.json
```

4. A secure passphrase was created and confirmed.

Verified wallet balance:

```
solana balance
```

5. The initial balance displayed as 0 SOL.

---

3. Airdrop and Transaction Execution

6. Requested a 10 SOL airdrop to fund the sender wallet:

```
solana airdrop 10
```

Balance updated to 10 SOL.

7. Created a recipient wallet:

```
solana-keygen new --outfile ~/Solana_PoC/recipient.json
```

Transferred 2 SOL to the recipient:

```
solana transfer ~/Solana_PoC/recipient.json 2
--allow-unfunded-recipient --url localhost
```

8.

Confirmed transaction success using the returned signature:

```
solana confirm -v <signature>
```

9.

Verified balances:

```
solana balance
```

```
solana balance ~/Solana_PoC/recipient.json
```

10. Sender ≈ 8 SOL, Recipient ≈ 2 SOL, confirming transfer completion.

---

4. RPC Communication Capture

11. Opened Wireshark and selected the **Loopback (lo0)** interface.
12. Applied a display filter:

```
tcp.port == 8899
```

13. Triggered an RPC request using the Solana JSON-RPC API:

```
bash curl -sS -X POST http://127.0.0.1:8899 \ -H "Content-Type: application/json" \ -d '{"jsonrpc":"2.0","id":121211,"method":"getVersion","params":[]}'
```

14. Wireshark captured the HTTP POST request and 200 OK response, showing the following JSON payload in clear text under the **Packet Bytes (Hex + ASCII)** pane:

```
json {"jsonrpc":"2.0","result":{"feature-set":4215500110,"solana-core":"1.18.18"},"id":121211}
```

---

5. Observations and Security Findings

- All Solana CLI operations executed successfully within the isolated macOS environment.
- RPC traffic was transmitted in plaintext over HTTP (port 8899).
- No encryption or authentication was applied at the protocol level.
- Security recommendation: restrict RPC access to localhost or implement TLS when connecting remotely.

6. Validation Summary

This procedure confirms Solana's core protocol functions—including validator startup, wallet management, airdrop funding, token transfers, and JSON-RPC communication—operate correctly on macOS.

The analysis verified that RPC data is readable in Wireshark, demonstrating Solana's use of unencrypted HTTP for JSON-RPC exchanges.