SHERLOCK

Security Review For Axal

Collaborative Audit Prepared For:	**Axal**
Lead Security Expert(s):	**defsec**
Date Audited:	**July 14 - July 21, 2025**
Final Commit:	**4159bad**

Introduction

Axal (getaxal.com, x.com/getaxal) streamlines earning yield on digital assets by using non-custodial 7702 smart wallets, sponsoring gas/batching transactions, and investing in top DeFi yield strategies.

Core components of the stack include Privy wallets and session signers, an open-sourced Trusted Execution Environment (TEE) that validates and signs all transactions, and a backend monitoring strategies and proposing transactions to the TEE.

This audit reviewed Axal's TEE and conducted a blackbox penetration test of Axal's backend.

Scope

Repository: getaxal/verified-signer

Audited Commit: a94cc7f212861ba06e367a3bf6f612961696a64c

Final Commit: 4159bad9745663dcd839dd9ab2bf13f2967d9ea0

Files:

- common/aws/config.go
- common/aws/regions.go
- common/aws/secret_manager/config.go
- common/aws/secret_manager/secret_manager.go
- common/aws/secret_manager/utils.go
- common/go.mod
- common/go.sum
- common/network/client.go
- common/network/transport.go
- common/vsock/conn.go
- common/vsock/fd.go
- common/vsock/listener.go
- common/vsock/proxy/proxy.go
- common/vsock/socket/accept4.go
- common/vsock/socket/accept.go
- common/vsock/socket/conn.go
- common/vsock/socket/conn_linux.go
- common/vsock/socket/doc.go

- common/vsock/socket/netns_linux.go
- common/vsock/socket/netns_others.go
- common/vsock/socket/setbuffer_linux.go
- common/vsock/socket/setbuffer_others.go
- common/vsock/socket/typ_cloexec_nonblock.go
- common/vsock/socket/typ_none.go
- common/vsock/vsock.go
- common/vsock/vsock_others.go
- enclave/attestation/attestation_data.go
- enclave/attestation/attestation.go
- enclave/cmd/main.go
- enclave/config.go
- enclave/docker_build_prod.sh
- enclave/docker_build.sh
- enclave/go.mod
- enclave/go.sum
- enclave/privy-signer/authorization_signature/authorization_signature.go
- enclave/privy-signer/authorization_signature/signing.go
- enclave/privy-signer/data/message.go
- enclave/privy-signer/data/privy_eth_tx_data.go
- enclave/privy-signer/data/privy_sol_tx_data.go
- enclave/privy-signer/data/privy_user_data.go
- enclave/privy-signer/privy_api_paths.go
- enclave/privy-signer/privy_client.go
- enclave/privy-signer/privy_config.go
- enclave/privy-signer/privy_eth_signing.go
- enclave/privy-signer/privy_sol_signing.go
- enclave/router/attestation_handler.go
- enclave/router/eth_handlers.go
- enclave/router/health_handler.go
- enclave/router/router.go

- enclave/router/sol_handler.go
- enclave/utils.go
- enclave/verifier/verifier.go
- enclave/verifier/whitelist.go
- host/cmd/main.go
- host/go.mod
- host/go.sum
- host/logs.sh
- host/network/proxy.go

Final Commit Hash

4159bad9745663dcd839dd9ab2bf13f2967d9ea0

Findings

Each issue has an assigned severity:

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.
- Low/Info issues are non-exploitable, informational findings that do not pose a security risk or impact the system's integrity. These issues are typically cosmetic or related to compliance requirements, and are not considered a priority for remediation.

Issues Found

High	Medium	Low/Info
1	5	21

Issues Not Fixed and Not Acknowledged

High	Medium	Low/Info
0	0	0

Issue H-1: Integer overflow in ethereum transaction value fields

Source: https://github.com/sherlock-audit/2025-07-axal-tee-server/issues/83

Summary

The Ethereum transaction data structure uses `int64` type for value and gas-related fields, which cannot represent the full range of valid Ethereum uint256 values. This causes integer overflow for transactions exceeding approximately 9.2 ETH or gas prices above `int64` maximum.

Vulnerability Detail

Ethereum blockchain uses uint256 (256-bit unsigned integers) for all monetary values and gas calculations, with amounts denominated in Wei (1 ETH = 10^18 Wei). The current implementation uses Go's int64 type, which has a maximum value of 9,223,372,036,854,775,807 (2^63 - 1).

This creates an overflow condition when:

- Transaction value exceeds ~9.223 ETH
- Gas prices exceed int64 maximum during high network congestion
- Gas limits exceed int64 maximum for complex operations

The overflow occurs silently in Go, resulting in incorrect values being processed without error indication.

Impact

Transactions with values exceeding int64 maximum will overflow, resulting in incorrect amounts being signed and sent.

Code Snippet

```
// Common transaction structure used in both APIs
  type EthTransaction struct {
      ChainID              *int64 `json:"chain_id,omitempty"`
      Data                 string `json:"data,omitempty"`
      From                 string `json:"from,omitempty"`
      GasLimit             *int64 `json:"gas_limit,omitempty"`        // Overflow
       ↪  risk
      GasPrice             *int64 `json:"gas_price,omitempty"`        // Overflow
       ↪  risk
```

```
    MaxFeePerGas         *int64 `json:"max_fee_per_gas,omitempty"`  // Overflow
    ↪  risk
    MaxPriorityFeePerGas *int64 `json:"max_priority_fee_per_gas,omitempty"` //
    ↪  Overflow risk
    Nonce                *int64 `json:"nonce,omitempty"`
    To                   string `json:"to"`
    Type                 *int64 `json:"type,omitempty"`
    Value                *int64 `json:"value,omitempty"`            // Primary
    ↪  overflow risk
}
```

Example overflow scenario:

- Sending 21.7 ETH = 21,734,488,100,000,000,000 Wei
- int64 maximum = 9,223,372,036,854,775,807 Wei
- Overflow result = 3,287,744,026,290,448,384 Wei (incorrect)

Tool Used

Manual Review

Recommendation

Replace `int64` types with appropriate representations (Using `math/big`) for Ethereum `uint256` values.

Discussion

EkamSinghPandher

Accepted

defsec

The issue is fixed with https://github.com/getaxal/verified-signer/commit/85e30156d67f77b5556a47abb8029063523958e7 .

Issue M-1: Incorrect HTTP status code in attestation endpoints

Source: https://github.com/sherlock-audit/2025-07-axal-tee-server/issues/59

Summary

The attestation endpoints return HTTP 400 (Bad Request) instead of HTTP 200 (OK) for successful operations, which could confuse clients and monitoring systems.

Vulnerability Detail

Both attestation handlers at `verified-signer/enclave/router/attestation_handler.go:47` and `verified-signer/enclave/router/attestation_handler.go:80` return `http.StatusBadRequest` instead of `http.StatusOK` when the attestation operation succeeds.

Impact

Applications may interpret successful attestations as failures.

Code Snippet

```
// verified-signer/enclave/router/attestation_handler.go:47
  resp := attestation.AttestationBytesResponse{
      Attestation: attString,
  }
  c.JSON(http.StatusBadRequest, resp)  // Should be StatusOK

  // verified-signer/enclave/router/attestation_handler.go:80
  resp := attestation.AttestationDocResponse{
      AttestationDoc: *doc,
  }
  c.JSON(http.StatusBadRequest, resp)  // Should be StatusOK
```

Tool Used

Manual Review

Recommendation

Fix the HTTP status codes to return http.StatusOK for successful attestation operations.

Discussion

EkamSinghPandher

This is a bug, accepted. Good catch

defsec

The issue is fixed in the branch :
https://github.com/getaxal/verified-signer/compare/fix/sec-audit-fixes...main

Issue M-2: Missing transaction verification on the signer

Source: https://github.com/sherlock-audit/2025-07-axal-tee-server/issues/61

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

The transaction verifier component exists but is completely bypassed in the signing flow, allowing all transactions to be processed without whitelist verification.

Vulnerability Detail

The project includes a comprehensive transaction verifier with whitelist functionality at `verified-signer/enclave/verifier/verifier.go:29-65`, but none of the signing handlers actually invoke this verification logic. The security control is implemented but never executed.

Impact

All transactions are processed without verification.

Code Snippet

```
// verified-signer/enclave/verifier/verifier.go:29-65
 func (v *Verifier) VerifyEthTxRequest(req data.EthTxRequest) bool {
     // ... comprehensive verification logic exists ...
 }

 // verified-signer/enclave/router/eth_handlers.go:100-108
 func EthTransactionSignTxHandler(c *gin.Context) {
     // ... validation ...
     // MISSING: verifier.VerifyEthTxRequest(transactionSignReq)
     resp, httpErr := privysigner.PrivyCli.EthSignTransaction(&transactionSignReq,
     ↪  ethWallet.WalletID)
     // ... signs without verification ...
 }
```

Tool Used

Manual Review

Recommendation

Call `verifier.VerifyEthTxRequest()` before signing.

Discussion

EkamSinghPandher

Ah this verifier is actually meant for v2 launch, it is disabled at the moment because the exact pools are not yet confirmed. It will be launched in a future launch, the logic has just been initiated, you can take it as if it is not part of this codebase for now.

defsec

Marked as a acknowledged due to the feature is not implemented.

Issue M-3: Race condition enables wallet duplication

Source: https://github.com/sherlock-audit/2025-07-axal-tee-server/issues/62

Summary

User cache implementation contains a race condition that could lead to duplicate wallet creation and application state inconsistencies under concurrent access.

Vulnerability Detail

The `GetUser` method in privy_user_manager.go:17-21 performs non-atomic cache operations by first checking if a key exists using `Has()` and then separately retrieving the value using `Get().Value()`. This creates a race condition where:

1. Multiple concurrent requests for the same uncached userId will all pass the initial cache check
2. The cache entry could expire between the Has() check and Get() call, causing a nil pointer dereference
3. All concurrent requests will proceed to wallet creation, potentially creating duplicate wallets for the same user

The issue occurs because cache operations are split into separate calls rather than using atomic get-and-check operations.

Impact

Multiple concurrent requests can trigger simultaneous wallet creation for the same user.

Code Snippet

https://github.com/sherlock-audit/2025-07-axal-tee-server/blob/868cac8486f4eb7e61f277f4abc7853af109cb3c/verified-signer/enclave/privy-signer/privy_user_manager.go#L17

Tool Used

Manual Review

Recommendation

Replace the separate `Has()` and `Get()` calls with a single atomic operation:

```
if item := cli.userCache.Get(userId); item != nil {
    log.Infof("Cache Hit: %s", userId)
    value := item.Value()
    return &value, nil
}
```

Discussion

EkamSinghPandher

Accepted, good catch.

defsec

The issue has been fixed with https://github.com/getaxal/verified-signer/commit/af74cab1d14c2262e7a7793f97e095539648ec8a.

Issue M-4: [API] Privy User ID in JWT Not Verified Against Request Body

Source: https://github.com/sherlock-audit/2025-07-axal-tee-server/issues/92

Summary

The backend endpoint accepts a `user_privy_id` in the request body without verifying it against the `sub` (subject) claim of the JWT provided in the `Authorization` header. This allows a user to submit requests on behalf of another user by modifying the `user_privy_id` field.

Vulnerability Detail

In the provided request, the backend receives both a signed JWT (as a Bearer token) and a `user_privy_id` in the request body:

```
{
  "privy_token": "<valid_jwt>",
  "token_amount_usdc": "100000000000000000",
  "user_privy_id": "cmegehrg73d6dgg28bbsba"
}
```

However, the backend responds with a `200 OK` status without validating that the `user_privy_id` matches the `sub` claim from the decoded JWT:

```
"sub": "did:privy:cmd90qeh501qiju0m702o7z6q"
```

This gap allows attackers to submit arbitrary `user_privy_id` values, leading to possible unauthorized actions on behalf of other users.

Impact

An attacker with a valid JWT for their own user could craft a request using someone else's `user_privy_id`, bypassing identity controls and potentially triggering privileged actions (such as deposits, withdrawals, or changes) for other users.

Code Snippet

Tool Used

Manual Review

Recommendation

Always verify that the `user_privy_id` (or any user-identifying field) matches the `sub` claim in the JWT provided. Reject the request if there is any mismatch between:

- `Authorization` header's JWT → `sub` field
- `user_privy_id` or any similar identifier in the request body

Enforce this check on all endpoints that rely on user-specific context to prevent identity spoofing.

Discussion

EkamSinghPandher

Ah understood, we will remove the user_privy_id since that field isnt even used

EkamSinghPandher

Neither of the token or the privy_id is used, this can be set as a medium issue

Issue M-5: [API] Strategy execution can be initiated without sufficient balance

Source: https://github.com/sherlock-audit/2025-07-axal-tee-server/issues/95

Summary

The API endpoint for initiating a strategy deposit does not validate whether the user has a sufficient balance. This allows deposit requests to be processed regardless of the user's actual token holdings.

Vulnerability Detail

The `/api/v1/strategy/deposit` endpoint responds with a success message (`Strategy execution initiated successfully`) even when the `token_amount_usdc` provided exceeds the actual balance of the user's account. This lack of validation could result in failed or stuck transactions downstream in the strategy pipeline.

Impact

Users can initiate strategy deposits without having the required balance.

Code Snippet

N/A – behavior was observed through manual testing.

Tool Used

Manual Review

Recommendation

Implement proper balance checks before executing strategy deposits. The backend should verify whether the user has sufficient USDC before proceeding with the deposit logic and return an appropriate error message if not.

Issue L-1: Resource exhaustion through unbounded goroutines

Source: https://github.com/sherlock-audit/2025-07-axal-tee-server/issues/58

Summary

The host proxy implementation creates unlimited goroutines for each connection without any bounds checking or resource management.

Vulnerability Detail

The proxy service spawns new goroutines for every incoming connection without implementing connection limits, goroutine pools, or resource management. This can lead to resource exhaustion attacks.

Impact

Complete system resource exhaustion.

Code Snippet

```
// host/cmd/main.go:17-23
 func main() {
     // Multiple unbounded proxy goroutines
     go network.InitVsockToTcpProxy(ctx, 50001, 443,
      ↪  "https://secretsmanager."+aws.USEast2.String()+".amazonaws.com")
     go network.InitVsockToTcpProxy(ctx, 50002, 443, "https://api.privy.io")
     go network.InitTcpToVsockProxy(ctx, 8080, 50003)
     go network.InitVsockToTcpProxy(ctx, 50004, 80, "http://169.254.169.254")

     // No resource limits or management
     for {
         time.Sleep(time.Hour)
     }
 }
```

Tool Used

Manual Review

Recommendation

Consider using goroutine pools instead of unlimited goroutine creation.

Discussion

EkamSinghPandher

Will connection pool this(mainly just go network.InitTcpToVsockProxy(ctx, 8080, 50003)). Accepted

defsec

Fixed with https://github.com/getaxal/verified-signer/pull/23/files.

Issue L-2: Cryptographic key material memory exposure in TEE

Source: https://github.com/sherlock-audit/2025-07-axal-tee-server/issues/60

Summary

Memory management issue in the TEE allow cryptographic key material to persist in enclave memory without proper sanitization, enabling key recovery through memory analysis attacks.

Vulnerability Detail

The enclave's cryptographic operations fail to implement secure memory management practices. Private keys and sensitive cryptographic material are processed using standard Go memory allocation without explicit zeroing, allowing keys to persist in memory indefinitely. String operations create immutable copies of base64-encoded private keys that cannot be securely cleared, and SHA-256 hash operations leave sensitive payload data in memory. The absence of secure memory scrubbing during enclave termination means cryptographic secrets remain accessible through memory dumps even after process termination.

Impact

ECDSA P-256 private keys can be extracted from enclave memory.

Code Snippet

```
// enclave/privy-signer/authorization_signature/signing.go:25-29
 hash := sha256.Sum256([]byte(payload))  // Hash data persists in memory
 signature, err := ecdsa.SignASN1(rand.Reader, privateKey, hash[:])
 if err != nil {
     return "", fmt.Errorf("failed to sign payload: %w", err)
 }
 // No explicit memory clearing - hash and signature data remains accessible

 // enclave/privy-signer/authorization_signature/signing.go:42-46
 pkcs8Bytes, err := base64.StdEncoding.DecodeString(pkcs8B64)
 if err != nil {
     return "", fmt.Errorf("failed to decode PKCS8 key: %w", err)
 }
 // Base64 decoding creates persistent copies in memory

 // enclave/privy-signer/authorization_signature/signing.go:42
```

```
pkcs8B64 := privyAuthorizationKey  // String copy of private key material
// Immutable string cannot be securely cleared from memory
```

Tool Used

Manual Review

Recommendation

1. Implement secure memory allocation using memory-mapped regions with mlock() to prevent swapping.
2. Use explicit memory zeroing with memset_s() or equivalent after cryptographic operations.
3. Replace string operations with byte slices for all sensitive data handling.

Discussion

EkamSinghPandher

Hmm, ok for this, AWS claims that the TEE memory is not accessible to anyone, even the host vm. This is why we can hold privy private keys within this enclave and this is the trust assumption we make. However, I will also still implement these changes. Accepted

defsec

Fixed with https://github.com/getaxal/verified-signer/commit/550d440f743ca11dd38568e56c1a91ef40433b71.

Issue L-3: Inconsistent error messages in solana handlers

Source: https://github.com/sherlock-audit/2025-07-axal-tee-server/issues/63

Summary

The Solana transaction handlers contain hardcoded error messages that incorrectly reference "eth wallet" instead of "sol wallet", creating confusion during error handling and potentially impacting incident response procedures.

Vulnerability Detail

Three locations in the Solana handler code contain error messages that reference Ethereum wallets instead of Solana wallets:

1. Line 118: In the `SolSignTxHandler` function, the error log incorrectly states "delegated eth wallet"
2. Lines 180-182: In the `SolSignAndSendTxHandler` function, both the error log and response message incorrectly reference "delegated eth wallet"

These inconsistencies occur when a user attempts Solana operations but doesn't have the required delegated wallet configured.

Impact

Security logs contain incorrect information that could complicate forensic analysis.

Code Snippet

File: verified-signer/enclave/router/sol_handler.go

```
// Line 118 - Incorrect "eth wallet" reference in Solana handler
log.Errorf("Solana signTransaction API error user %s does not have a delegated eth
↪  wallet", privyUserId)

// Lines 180-182 - Incorrect "eth wallet" references in Solana handler
log.Errorf("Sol signAndSend API error user %s does not have a delegated eth
↪  wallet", privyUserId)
resp := privydata.Message{
    Message: "user does not have an delegated eth wallet",
}
```

Tool Used

Manual Review

Recommendation

Update all error messages in Solana handlers to correctly reference Solana wallets.

Discussion

EkamSinghPandher

Yeah this is a error accepted.

defsec

Fixed with https://github.com/getaxal/verified-signer/commit/c29f1ae0df97ec89d23eea33c8c7a1b59777ccde.

Issue L-4: Insecure log file permissions

Source: https://github.com/sherlock-audit/2025-07-axal-tee-server/issues/64

Summary

Deployment scripts create log files without restricting access permissions, potentially exposing sensitive TEE runtime information, cryptographic operations, and configuration details to unauthorized users on the host system.

Vulnerability Detail

The deployment scripts (run.sh, run_prod.sh, docker_build.sh) create multiple log files but don't set restrictive permissions:

- $LOG_DIR/build.log - Contains Docker build output
- $LOG_DIR/console.log - Contains enclave console output
- $LOG_DIR/enclave.log - Contains deployment logs

These files may contain sensitive information such as:

- AWS credential references
- Enclave configuration details
- Error messages with internal system information
- Timing information that could aid side-channel attacks

Impact

Sensitive runtime data exposed to other users/processes on host.

Code Snippet

Files: Multiple deployment scripts

```
# run.sh - No permission restrictions
BUILD_LOG="$LOG_DIR/build.log"
CONSOLE_LOG="$LOG_DIR/console.log"
echo "=== Nitro Enclave Console Log - $(date) ===" > "$CONSOLE_LOG"

# run_prod.sh - No permission restrictions
ENCLAVE_LOG="$LOG_DIR/enclave.log"
echo "=== Production Enclave Deployment - $(date) ===" >> "$ENCLAVE_LOG"

# docker_build.sh - No permission restrictions
```

```
BUILD_LOG="$LOG_DIR/build.log"
echo "=== Docker Build Log - $(date) ===" >> "$BUILD_LOG"
```

Tool Used

Manual Review

Recommendation

Implement secure log file creation with restricted permissions:

```
# Create log directory with restricted permissions
LOG_DIR="./log"
mkdir -p "$LOG_DIR"
chmod 750 "$LOG_DIR"  # Owner rwx, group rx, other none

# Create log files with restricted permissions
BUILD_LOG="$LOG_DIR/build.log"
CONSOLE_LOG="$LOG_DIR/console.log"
ENCLAVE_LOG="$LOG_DIR/enclave.log"

# Set restrictive permissions on log files
touch "$BUILD_LOG" "$CONSOLE_LOG" "$ENCLAVE_LOG"
chmod 640 "$BUILD_LOG" "$CONSOLE_LOG" "$ENCLAVE_LOG"  # Owner rw, group r, other
↪  none
chown root:nitro-enclave "$BUILD_LOG" "$CONSOLE_LOG" "$ENCLAVE_LOG" 2>/dev/null ||
↪  true
```

Discussion

EkamSinghPandher

Will double check the build logs to see if we print any sensitive info, in prod, the TEE actually does not produce any logs. We only enable it with the add logger for dev environments so there is actually no console.log in prod.

EkamSinghPandher

Yeah it seems there is no sensitive info in the build logs, just docker image name and eif file location.

defsec

Fixed with https://github.com/getaxal/verified-signer/commit/02df0d402f029a8fa2f8b2c15f5a893ba53d65b1.

Issue L-5: Cross chain replay attack via unrestricted network validation

Source: https://github.com/sherlock-audit/2025-07-axal-tee-server/issues/66

Summary

Missing chain ID validation allows attackers to execute transactions on any Ethereum network, enabling cross-chain replay attacks and unintended network execution.

Vulnerability Detail

The `ValidateTxRequest()` function in `EthSendTransactionRequest` only validates that CAIP2 is not empty (line 130-132) but performs no validation of:

- CAIP2 format (eip155:chainId structure)
- Allowed chain IDs (could accept eip155:1 for mainnet when expecting testnet)
- Numeric chain ID validation
- Cross-chain protection

Impact

Cross-chain replay attacks between different Ethereum networks.

Code Snippet

https://github.com/sherlock-audit/2025-07-axal-tee-server/blob/868cac8486f4eb7e61f277f4abc7853af109cb3c/verified-signer/enclave/privy-signer/data/privy_eth_tx_data.go#L130-L131

Tool Used

Manual Review

Recommendation

1. Parse and validate CAIP2 format: eip155:.
2. Maintain allowlist of permitted chain IDs.
3. Validate numeric chain ID matches expected network.
4. Reject transactions for unauthorized networks.

Discussion

EkamSinghPandher

Accepted, will add more complex caip2 parsing.

defsec

Fixed with https://github.com/getaxal/verified-signer/commit/b94e415e1df492c18d9855207c5fe168f76f1091.

Issue L-6: Error variable shadowing in transaction send handler

Source: https://github.com/sherlock-audit/2025-07-axal-tee-server/issues/75

Summary

The `EthTransactionSendTxHandler` function incorrectly logs the validation error instead of the actual Privy API error when transaction sending fails, causing debugging confusion and inconsistent error reporting.

Vulnerability Detail

In enclave/router/eth_handlers.go:190, the error logging uses the wrong variable:

```
resp, httpErr := privysigner.PrivyCli.EthSendTransaction(&transactionSendReq,
↪  ethWallet.WalletID)
if httpErr != nil {
    log.Errorf("Eth transaction send API error user %s could not send tx with err:
    ↪  %v", privyUserId, err)
    c.JSON(httpErr.Code, httpErr.Message)
    return
}
```

The variable err at this point contains the last validation error from line 162, not the actual Privy API error. This differs from the consistent pattern used in other handlers (lines 66, 128, 252) which correctly log `httpErr.Message.Message`.

Impact

Developers investigating transaction failures will see validation errors instead of actual API errors.

Code Snippet

https://github.com/sherlock-audit/2025-07-axal-tee-server/blob/868cac8486f4eb7e61f277f4abc7853af109cb3c/verified-signer/enclave/router/eth_handlers.go#L190-L191

Tool Used

Manual Review

Recommendation

Change line 190 to use the correct error variable:

- `log.Errorf("Eth transaction send API error user %s could not send tx with err: %v", privyUserId, httpErr.Message.Message)`

This ensures consistent error logging across all Ethereum handlers and provides accurate debugging information.

Discussion

defsec

Same issue for Sol :

```
File: enclave/router/sol_handler.go:190
- Same issue: wrong error variable logged
```

defsec

Hi @EkamSinghPandher , I'm not sure If you missed that one, but also wanted to inform you about that one. Thank you!

defsec

Fixed with https://github.com/getaxal/verified-signer/commit/17e5fd498fb0b1d5b1d6afdb9906899ae8b86c54.

Issue L-7: Dependency verification disabled on the Dockerfile

Source: https://github.com/sherlock-audit/2025-07-axal-tee-server/issues/76

Summary

The Docker build configuration disables Go's dependency verification system by setting `GOSUMDB=off`, which completely disables checksum validation for all Go modules

Vulnerability Detail

The current Dockerfile configuration globally disables the Go checksum database verification:

```
ENV GOSUMDB=off
```

This setting was likely added to avoid checksum validation errors for private modules (github.com/getaxal/*), but it incorrectly disables verification for all modules, including public ones.

When GOSUMDB=off is set:

- No checksum verification occurs for any downloaded modules
- The build process becomes vulnerable to dependency substitution attacks
- Man-in-the-middle attacks can inject malicious code during dependency downloads
- There's no protection against compromised module registries

Impact

No verification that downloaded modules match their expected checksums.

Code Snippet

Current vulnerable configuration:

```
ENV GOPRIVATE=github.com/getaxal/*
ENV GOPROXY=direct
ENV GOSUMDB=off
```

Tool Used

Manual Review

Recommendation

Replace the global **GOSUMDB=off** setting with **GONOSUMDB** that only bypasses checksum verification for private modules:

1. Remove the global disable: Delete **ENV GOSUMDB=off**
2. Use GONOSUMDB for private modules: **Add ENV GONOSUMDB=github.com/getaxal/***
3. Keep GOPRIVATE: Maintain **ENV GOPRIVATE=github.com/getaxal/*** for proxy bypass

Discussion

EkamSinghPandher

Accepted

defsec

Fixed with https://github.com/getaxal/verified-signer/commit/9ec47c8261b6ad2095f875f7a52a55be6a9fcf7e.

Issue L-8: Dockerfile running as root user

Source: https://github.com/sherlock-audit/2025-07-axal-tee-server/issues/77

Summary

The Dockerfile is configured to run the application as the root user, which poses a significant security vulnerability. Running containers as root provides unnecessary privileges and increases the attack surface if the container is compromised.

Vulnerability Detail

The Dockerfile creates a runtime image from `alpine:latest` and sets the working directory to `/root/`, which implies the application will run as the root user. This is a security anti-pattern that violates the principle of least privilege.

Impact

Running as root can lead to:

- **Privilege Escalation**: If the application is compromised, attackers gain root access to the container
- **Host System Compromise**: Root access in containers can potentially escape to the host system
- **Data Breach**: Root privileges allow access to all files and processes within the container
- **Compliance Violations**: Many security standards (CIS, NIST) require containers to run as non-root users
- **Kubernetes Security**: Pod Security Standards and admission controllers may reject root containers

Code Snippet

```
# Stage 2: Runtime image
FROM alpine:latest

WORKDIR /root/  #   Sets working directory to root user's home

# Install CA certificates for HTTPS requests
RUN apk add --no-cache ca-certificates

# Copy the binary from builder stage
```

```
COPY --from=builder /app/main .
RUN chmod +x ./main

# Copy config file
COPY config.yaml /root/config.yaml  #  Places files in root-owned directory

# Run the application
CMD ["/root/main", "-config", "/root/config.yaml"]  #  Runs as root
```

Tool Used

Manual Review

Recommendation

Consider running container as a non-root user.

Discussion

EkamSinghPandher

Accepted, will change it

defsec

Fixed with https://github.com/getaxal/verified-signer/commit/f49672a87f27a51d51e103d5b6f5e9c09b9057fc.

Issue L-9: Missing LRU eviction policy in user cache implementation

Source: https://github.com/sherlock-audit/2025-07-axal-tee-server/issues/78

Summary

The user cache implementation in the Privy client uses a fixed 30-minute TTL configuration without capacity limits or LRU eviction policy. This creates an ineffective caching strategy that can lead to memory bloat and poor cache performance, especially under high load conditions with many unique users.

Vulnerability Detail

The current cache configuration only specifies a TTL without any capacity management:

```
cache := ttlcache.New(
    ttlcache.WithTTL[string, data.PrivyUser](30 * time.Minute),
)
```

This implementation has several performance issues:

1. No Capacity Limits: The cache can grow indefinitely, potentially consuming excessive memory.
2. No LRU Eviction: Without capacity limits, the jellydator/ttlcache v3 library doesn't enable LRU eviction policy.

Impact

Unbounded cache growth can lead to memory exhaustion in high-traffic scenarios.

Code Snippet

```
cache := ttlcache.New(
      ttlcache.WithTTL[string, data.PrivyUser](30 * time.Minute),
  )
```

Tool Used

Manual Review

Recommendation

Implement a proper cache strategy with capacity limits and LRU eviction.

Discussion

EkamSinghPandher

Accepted, will add a capacity

defsec

Fixed with https://github.com/getaxal/verified-signer/commit/f7d4505c95142826358813fc9110f73a86811668.

Issue L-10: Dependency version conflicts in go modules

Source: https://github.com/sherlock-audit/2025-07-axal-tee-server/issues/79

Summary

The multi-module Go project has inconsistent dependency versions across its modules, including mismatched versions of **golang.org/x/sync** (`v0.14.0 vs v0.15.0`) and internal module dependencies (`common module at v0.1.1 vs v0.1.6`). These version conflicts can lead to build failures, runtime incompatibilities, and unpredictable behavior.

Vulnerability Detail

The project consists of three Go modules with version inconsistencies:

1. golang.org/x/sync version mismatch: - common/go.mod: v0.14.0 (direct dependency) - host/go.mod: v0.14.0 (indirect) - enclave/go.mod: v0.15.0 (indirect)
2. Internal module version mismatch: - host/go.mod: requires github.com/getaxal/verified-signer/common v0.1.1 - enclave/go.mod: requires github.com/getaxal/verified-signer/common v0.1.6

Impact

Go's module system may fail to resolve conflicting dependencies.

Code Snippet

// common/go.mod

```
module github.com/getaxal/verified-signer/common
go 1.24
require (
    golang.org/x/sync v0.14.0
)
```

// host/go.mod

```
module github.com/getaxal/verified-signer/host
go 1.24
require (
    github.com/getaxal/verified-signer/common v0.1.1
)
require (
```

```
    golang.org/x/sync v0.14.0 // indirect
)
```

// enclave/go.mod

```
module github.com/getaxal/verified-signer/enclave
go 1.24
require (
    github.com/getaxal/verified-signer/common v0.1.6
)
require (
    golang.org/x/sync v0.15.0 // indirect
)
```

Tool Used

Manual Review

Recommendation

Standardize dependency versions across all modules.

Discussion

EkamSinghPandher

Will fix, accepted.

defsec

The issue is with using relative imports. Tree : https://github.com/getaxal/verified-signer/tree/fix/audit

Issue L-11: Inconsistent error handling in main application function

Source: https://github.com/sherlock-audit/2025-07-axal-tee-server/issues/81

Summary

The main function in `enclave/cmd/main.go` demonstrates inconsistent error handling patterns where critical configuration loading errors result in silent failures using return statements instead of proper application termination.

Vulnerability Detail

The application has inconsistent error handling for startup failures:

```
Lines 28-30 (Port Config Error):
if err != nil {
    log.Errorf("Could not fetch Port config due to err: %v", err)
    return  // Silent exit
}

Lines 37-39 (Environment Config Error):
if err != nil {
    log.Errorf("Could not fetch Env config due to err: %v", err)
    return  // Silent exit
}

Line 44 (Privy Client Error - Handled Correctly):
if err != nil {
    log.Fatalf("Error creating privy cli: %v", err)  // Proper fatal error
}
```

The application exits with status code 0 (success) even when critical errors occur.

Impact

Configuration errors cause the service to exit without proper error indication.

Code Snippet

https://github.com/sherlock-audit/2025-07-axal-tee-server/blob/868cac8486f4eb7e61f277f4abc7853af109cb3c/verified-signer/enclave/cmd/main.go#L29-L30

```
func main() {
    log.Info("Initiating enclave for Axal Verified Signer")

    // Define command line flag for config path
    configPath := flag.String("config", "config.yaml", "Path to configuration file")
    flag.Parse()

    // Setup network port management config
    portCfg, err := enclave.LoadPortConfig(*configPath)

    if err != nil {
        log.Errorf("Could not fetch Port config due to err: %v", err)
        return
    }

    PortsConfig = portCfg

    envCfg, err := enclave.LoadEnvConfig(*configPath)

    if err != nil {
        log.Errorf("Could not fetch Env config due to err: %v", err)
        return
    }

    err = privysigner.InitNewPrivyClient(*configPath, PortsConfig, envCfg)

    if err != nil {
        log.Fatalf("Error creating privy cli: %v", err)
    }

    router.InitRouter(PortsConfig.RouterVsockPort)
}
```

Tool Used

Manual Review

Recommendation

Replace all startup errors with log.Fatalf() for consistent behavior:

```
if err != nil {
    log.Fatalf("Could not fetch Port config due to err: %v", err)
}
```

Discussion

EkamSinghPandher

Accepted

defsec

Fixed with : https://github.com/getaxal/verified-signer/commit/625baa1b780ef7e2f695e91cac946cd4ee7b7bef

Issue L-12: Insufficient ethereum address validation in whitelist verification

Source: https://github.com/sherlock-audit/2025-07-axal-tee-server/issues/85

Summary

The whitelist verification system performs only basic string matching without validating Ethereum address format, checksums, or normalization.

Vulnerability Detail

The whitelist validation performs only a simple map lookup without any address format validation:

```
func (wl *WhiteList) IsWhitelisted(address string) bool {
    return (wl.addressList)[address]  // Simple string match only
}
```

This implementation lacks several critical validations:

1. No Format Validation: Doesn't verify the address is a valid 42-character hex string starting with ”0x”
2. No EIP-55 Checksum Validation: Doesn't validate mixed-case checksum encoding per EIP-55

Impact

Invalid addresses passing validation may cause transaction failures downstream.

Code Snippet

```
func (v *Verifier) VerifyEthTxRequest(req data.EthTxRequest) bool {
    switch req.GetMethod() {
    case "eth_signTransaction":
        tx := req.GetTransaction()
        if tx == nil {
            return false
        }

        if !v.verifiedAddresses.IsWhitelisted(tx.To) {  // No address validation
            return false
        }
        return true
```

```
    }
}

func (wl *WhiteList) IsWhitelisted(address string) bool {
    return (wl.addressList)[address]  // Raw string comparison
}
```

Tool Used

Manual Review

Recommendation

Implement comprehensive Ethereum address validation and normalization.

Discussion

EkamSinghPandher

accepted

defsec

Fixed with https://github.com/getaxal/verified-signer/commit/68212b400aef64419b55174b8eb625fb1333154e.

Issue L-13: [API] CORS Misconfiguration Allows Credentials with Wildcard Origin (*)

Source: https://github.com/sherlock-audit/2025-07-axal-tee-server/issues/87

Summary

A CORS misconfiguration was observed in the backend response, where `Access-Control-Allow-Origin: *` is used together with `Access-Control-Allow-Credentials: true`. This violates the Fetch specification and can introduce serious security issues, including unauthorized access to sensitive resources via malicious cross-origin requests.

Vulnerability Detail

The backend server (`yield-backend-staging-v1.getaxal.com`) responds to cross-origin requests with the following CORS headers:

```
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
```

This combination is invalid and dangerous. When credentials are allowed (`Allow-Credentials: true`), the `Access-Control-Allow-Origin` header **must not be** `*`, and instead should explicitly reflect the origin.

Allowing `*` with credentials opens up the possibility for an attacker to send authenticated requests to the backend from a malicious origin (`Origin: malicious.com`), which could lead to **Cross-Site Request Forgery (CSRF)** or unauthorized data access.

Impact

Cookies or authorization headers may be sent along with cross-origin requests from malicious domains.

Code Snippet

Request:

```
Origin: malicious.com
Referer: https://yield-backend-staging-v1.getaxal.com/api/v1/health/ping
```

Response:

```
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
```

Tool Used

Manual Review

Recommendation

Update the server's CORS policy to **dynamically reflect the** `Origin` **header only for allowed domains** when sending `Access-Control-Allow-Credentials: true`. Do **not** use the wildcard `*` in such cases.

Discussion

EkamSinghPandher

Will fix, accepted

Issue L-14: [API] Server not behind cloudflare or any reverse proxy

Source: https://github.com/sherlock-audit/2025-07-axal-tee-server/issues/88

Summary

The staging server (`yield-backend-staging-v1.getaxal.com`) appears to be exposed directly to the internet without protection from Cloudflare or another reverse proxy.

Vulnerability Detail

The staging server (`yield-backend-staging-v1.getaxal.com`) appears to be exposed directly to the internet without protection from Cloudflare or another reverse proxy.

Impact

- Increased risk of **DDoS attacks**, **IP-based enumeration**, or **direct scanning**
- Lack of **WAF** (Web Application Firewall) and **rate limiting**

Code Snippet

Tool Used

Manual Review

Recommendation

Route all traffic through a security-focused reverse proxy like **Cloudflare**, **AWS CloudFront**, or **Fastly** to harden edge-layer protection and obfuscate infrastructure details.

Discussion

EkamSinghPandher

Will implement this

defsec

Acknowledged according to comment.

Issue L-15: [API] Overly Permissive HTTP Methods Exposure

Source: https://github.com/sherlock-audit/2025-07-axal-tee-server/issues/89

Summary

All HTTP methods (GET, POST, PUT, DELETE, OPTIONS) are allowed on the endpoint.

Vulnerability Detail

The server exposes all major HTTP methods : Access-Control-Allow-Methods: GET, POST, PUT, DELETE, OPTIONS.

Impact

All HTTP methods are available, expanding attack surface.

Code Snippet

```
HTTP/2 404 Not Found
Date: Fri, 18 Jul 2025 09:53:33 GMT
Content-Type: text/plain
Content-Length: 18
Access-Control-Allow-Credentials: true
Access-Control-Allow-Headers: Authorization, Content-Type, X-Privy-Token
Access-Control-Allow-Methods: GET, POST, PUT, DELETE, OPTIONS
Access-Control-Allow-Origin: *
Access-Control-Expose-Headers: Content-Length

404 page not found
```

Tool Used

Manual Review

Recommendation

Only allow necessary HTTP methods for the specific endpoint.

Discussion

EkamSinghPandher

Will restrict them

Issue L-16: [API] Missing security headers

Source: https://github.com/sherlock-audit/2025-07-axal-tee-server/issues/90

Summary

The website https://yield-backend-staging-v1.getaxal.com/ is missing security headers.

Vulnerability Detail

The server response lacks security headers that are industry standard for protecting web applications:

- **Strict-Transport-Security**: Missing HSTS header to enforce HTTPS
- **Content-Security-Policy**: No CSP header to prevent XSS attacks
- **X-Frame-Options**: Missing protection against clickjacking attacks
- **X-Content-Type-Options**: No protection against MIME sniffing
- **Referrer-Policy**: Missing control over referrer information disclosure
- **Permissions-Policy**: No control over browser features and APIs

Impact

- **XSS vulnerabilities**: Without CSP, malicious scripts can execute
- **Clickjacking attacks**: Missing X-Frame-Options allows site framing
- **MIME sniffing attacks**: Browser may execute malicious content
- **Information disclosure**: Referrer information may leak sensitive data
- **Feature abuse**: No restrictions on browser APIs and features
- **HTTPS downgrade**: No HSTS enforcement

Code Snippet

https://yield-backend-staging-v1.getaxal.com

Tool Used

Manual Review

Recommendation

Implement the following security headers:

- `Strict-Transport-Security: max-age=31536000; includeSubDomains`
- `Content-Security-Policy: default-src 'self'`
- `X-Frame-Options: SAMEORIGIN`
- `X-Content-Type-Options: nosniff`
- `Referrer-Policy: strict-origin-when-cross-origin`
- `Permissions-Policy: geolocation=(), microphone=(), camera=()`

Discussion

EkamSinghPandher

Accepted, will enable

Issue L-17: [API] MFA Not Enforced on Login and Private Key Export via Privy

Source: https://github.com/sherlock-audit/2025-07-axal-tee-server/issues/91

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

The application does not enforce Multi-Factor Authentication (MFA) during critical user actions such as logging in and exporting the private key or recovery phrase when using Privy for authentication.

Vulnerability Detail

While Privy is integrated for user authentication, there is no additional layer of MFA enforced during sensitive operations such as login or private key/phrase export.

Impact

Lack of MFA on critical actions increases the risk of unauthorized access and key leakage. If an attacker gains access to a user's credentials (e.g., through phishing or credential stuffing), they can log in and export the private key or recovery phrase without any additional verification.

Code Snippet

https://axal-yield-frontend-git-staging-axal.vercel.app

Tool Used

Manual Review

Recommendation

Integrate an additional layer of MFA for:

- Login process (at minimum, after password verification)
- Exporting private key or recovery phrase

Privy allows adding MFA via Passkey/Authenticator; enable and enforce it on these sensitive operations.

Discussion

EkamSinghPandher

We spoke to privy, they unfortunately dont have MFA for logins, will push them to implement it

defsec

Acknowledged.

Issue L-18: [API] Privy token stored in cookie without HttpOnly flag

Source: https://github.com/sherlock-audit/2025-07-axal-tee-server/issues/96

Summary

The `privy-token` cookie for the domain `axal-yield-frontend-git-staging-axal.vercel.app` is stored **without** the `HttpOnly` flag.

Vulnerability Detail

- `HttpOnly: false` as seen in browser dev tools
- Cookie Name: `privy-token`
- Domain: `axal-yield-frontend-git-staging-axal.vercel.app`
- Path: `/`
- SameSite: `Strict`
- Secure: `true`

Impact

Without the `HttpOnly` flag can be accessed via JavaScript in the browser. If an attacker exploits a stored or reflected XSS vulnerability, they can extract the user's authentication token and perform unauthorized actions on their behalf.

Code Snippet

N/A

Tool Used

Manual Review

Recommendation

Set the `HttpOnly` flag on all authentication-related cookies, including `privy-token`, to prevent client-side access.

Issue L-19: [API] Leaked API key & missing rate limiting on third-party api

Source: https://github.com/sherlock-audit/2025-07-axal-tee-server/issues/98

Summary

The frontend is making unauthenticated requests directly to the Dune API (`api.sim.dune.com`) using a hardcoded API key. This exposes the `X-Sim-Api-Key` to end users, allowing them to use the key for arbitrary requests outside of the intended application context.

Vulnerability Detail

A hardcoded API key (`X-Sim-Api-Key: sim_oasaX6kgxxxxx`) is exposed in requests sent directly from the browser to `https://api.sim.dune.com`. Since this key is embedded in the frontend and accessible via browser developer tools, it can be extracted and abused by malicious actors.

Impact

Without user-level throttling, a single user (or attacker) can consume the entire quota.

Code Snippet

The following request reveals the exposed key:

```
GET /v1/evm/balances/xxxx?chain_ids=all
Host: api.sim.dune.com
X-Sim-Api-Key: sim_oasaX6kgxxxxx
Origin: https://axal-yield-frontend-git-staging-axal.vercel.app
```

Tool Used

Manual Review

Recommendation

- **Proxy the API requests through a backend service**, which appends the `X-Sim-Api-Key` server-side. This prevents the key from being exposed on the client.
- **Implement user-based rate limiting** and session binding using access tokens.
- **Rotate the exposed API key** immediately and review any potential misuse.

Discussion

devd-99

We removed this dependency, it was a temporary workaround. Now we're getting this data from the backend without Dune

defsec

Hi @devd-99 can I test your vercel page? Thank you so much!

Issue L-20: [API] Privy-Generated wallet interacts with external protocols without enforced policies

Source: https://github.com/sherlock-audit/2025-07-axal-tee-server/issues/99

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

The Axal Prime web application allows Privy-generated wallets to interact with multiple external DeFi protocols. However, there is no enforcement of access policies or restrictions based on allowlisted protocol interactions.

Vulnerability Detail

The application enables users to earn yield through Tier 1 lending protocols by utilizing Privy-generated wallets. These wallets are capable of interacting with external DeFi platforms such as Moonwell, Seamless, Spark, and Euler. However, there are no access control policies implemented to restrict or validate which protocols the Privy wallet is permitted to engage with.

Impact

Interaction with unverified or malicious protocols.

Code Snippet

No direct code reference available, observed via frontend behavior and transaction tracing.

Tool Used

Manual Review

Recommendation

Implement Privy wallet policy enforcement on the backend or via middleware to allow only interactions with an approved list of protocols. Additionally, validate destination contracts and enforce these rules through signature-based or permissioned smart contract layers.

Discussion

defsec

Marked as an acknowledged.

Issue L-21: Lack of mTLS authentication in communication

Source: https://github.com/sherlock-audit/2025-07-axal-tee-server/issues/100

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

The current TLS setup used in the client-server communication ensures encryption but lacks mutual authentication.

Vulnerability Detail

The platform currently uses TLS to secure communication between client and server endpoints. However, there is no evidence of mutual TLS (mTLS) being implemented.

Impact

The server cannot guarantee the authenticity of the connecting client.

Code Snippet

N/A

Tool Used

Manual Review

Recommendation

Implement mutual TLS (mTLS) to enforce two-way authentication between host and TEE.

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.