HALBORN

MystenLabs - Adapter & Verifier

Layer 1 Security Audit

Prepared by: **Halborn**

Date of Engagement: **August 1st, 2022 - October 14th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	08/01/2022	Jakub Heba
0.2	Document Updates	09/09/2022	Jakub Heba
0.3	Document Updates	09/15/2022	John Saigle
0.4	Document Updates	09/20/2022	Luis Quispe Gonzales
0.5	Draft Version	10/18/2022	Luis Quispe Gonzales
0.6	Draft Review	10/18/2022	Gabi Urrutia
1.0	Remediation Plan	05/08/2023	Luis Quispe Gonzales
1.1	Remediation Plan Review	05/10/2023	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Luis Quispe Gonzales	Halborn	Luis.QuispeGonzales@halborn.com
Jakub Heba	Halborn	Jakub.Heba@halborn.com
John Saigle	Halborn	John.Saigle@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

MystenLabs engaged Halborn to conduct a security audit on their smart contracts beginning on August 1st, 2022 and ending on October 14th, 2022. The security assessment was scoped to the smart contracts provided to the Halborn team.

1.2 AUDIT SUMMARY

The team at Halborn was provided two months and a half for the engagement and assigned full-time security engineers to audit the security of the solution. The security engineers are blockchain and smart-contract security experts with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to:

- Ensure that smart contract functions operate as intended
- Identify potential security issues with the smart contracts

In summary, Halborn identified some improvements to reduce the likelihood and impact of multiple risks, which has been mostly addressed by the MystenLabs team. The main ones are the following:

- Update the code of Serde library to handle situations when additional fields have been included in input messages.
- Verify and restrict the use of ChangeEpoch in batch transactions.
- Set overflow-checks flag to true in cargo.toml. Alternatively, use checked arithmetic operations.
- Instead of relying on an adequate serialization of all fields in a transaction, enforce that is_verified value is false before further processing.
- Use an adequate error handling method instead of using unsafe methods and macros that could crash the nodes.

- Secure the sui-gateway, so that publishing large packages does not block the entire client.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the solidity code and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture and purpose.
- Smart contract manual code review and walkthrough.
- Manual testing by custom scripts and fuzzers.
- Scanning of Rust files for vulnerabilities, security hotspots or bugs.
- Static Analysis of security for scoped contract, and imported functions.
- Testnet deployment.

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.

1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

1. Rust Crates
    (a) Repository: narwhal
        i. Commit ID: 21e039631e3b91f0404c52829dba7f245d191ff7
    (b) Repository: sui
        i. Commit ID: 46f9662ca129867d84e512e7ac954aa33dd00599
    (c) Crates in scope:
        i. sui-adapter
        ii. sui-verifier
        iii. sui-framework
        iv. sui-types/src/crypto.rs
        v. sui-types/src/signature-seed.rs

Out-of-scope: External libraries and financial related attacks.

1.5 CAVEATS

Some vulnerabilities described in this report were found by testing the whole transactions' life cycle, so they could even affect components that are **out-of-scope** for this audit. Nevertheless, it is highly advisable to complement the results of these tests with specific audits for the remaining components: sui-core, sui-node, sui-storage, etc.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
2	0	1	4	7

LIKELIHOOD

IMPACT

(HAL-03)				(HAL-01) (HAL-02)
(HAL-04) (HAL-05) (HAL-06)				
(HAL-08) (HAL-09)	(HAL-07)			
(HAL-10) (HAL-11) (HAL-12) (HAL-13) (HAL-14)				

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
INADEQUATE DESERIALIZATION OF TRANSACTIONS LEADS TO A DOS OF NODES	Critical	SOLVED - 10/26/2022
ANY USER CAN CHANGE EPOCHS BY USING BATCH TRANSACTIONS	Critical	SOLVED - 08/24/2022
INADEQUATE HANDLING OF POTENTIAL INTEGER OVERFLOW	Medium	SOLVED - 03/29/2023
INCOMPLETE VALIDITY CHECK ON TRANSACTIONS	Low	SOLVED - 10/26/2022
SOME ASSERTS ARE NOT ENFORCED WHEN PACKAGES ARE COMPILED IN RELEASE MODE	Low	PARTIALLY SOLVED - 05/08/2023
POSSIBLE RUST PANICS DUE TO UNSAFE METHODS AND MACROS USAGE	Low	PARTIALLY SOLVED - 05/08/2023
CLIENTS DENIAL OF SERVICE WHEN PUBLISHING A LARGE PACKAGE	Low	SOLVED - 08/03/2022
DEBUG TRAIT IMPLEMENTED ON SENSITIVE VALUES	Informational	SOLVED - 09/05/2022
SEPARATE RULES FOR GENESIS MODE CAN LEAD TO MEMORY-RELATED ISSUES	Informational	SOLVED - 10/12/2022
UNMAINTAINED DEPENDENCIES	Informational	PARTIALLY SOLVED - 05/08/2023
USE OF CLONE TRAIT MAY REDUCE PERFORMANCE	Informational	ACKNOWLEDGED
USE OF OSRNG MAY REDUCE PERFORMANCE	Informational	PARTIALLY SOLVED - 10/22/2022
UNUSED FUNCTION WITH POTENTIAL RISKY LOGIC	Informational	SOLVED - 10/22/2022
DEVELOPER NOTE INDICATES POTENTIAL ISSUE	Informational	SOLVED - 10/22/2022

FINDINGS & TECH DETAILS

3.1 (HAL-01) INADEQUATE DESERIALIZATION OF TRANSACTIONS LEADS TO A DOS OF NODES - CRITICAL

Description:

When sending a transaction using TransactionEnvelope with additional arbitrary fields with specific values (see table below), the receiving nodes will automatically crash when trying to deserialize the transaction because of an out-of-memory (OOM) issue. As a consequence, a malicious user can create a Denial-of-Service (DoS) of the nodes by just sending a crafted transaction.

For the proof of concept video showing how to exploit this security issue, we are using the is_verified field with **true** as a value, which it is supposed not to be serialized, but an attacker can force it anyway in his client.

It is worth mentioning that any other arbitrary field name could have been used instead of is_verified (e.g.: new_stuff) for the **proof of concept** showed above, as long as it uses one of the following values (non-exhaustive list):

Data type	Value(s)
bool	true
u8	1
u16	1
u32	0, 1
String	""

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

Update the code of Serde serialization / deserialization library to handle situations when additional fields have been included in input messages.

Remediation plan:

SOLVED: The issue was fixed in commit 0d00b40.

3.2 (HAL-02) ANY USER CAN CHANGE EPOCHS BY USING BATCH TRANSACTIONS - CRITICAL

Description:

Despite ChangeEpoch transaction is restricted from being called externally (i.e: by any user), this measure can be bypassed by calling ChangeEpoch in a batch transaction (TransactionKind::Batch). This issue happens because is_system_tx function incorrectly assumes that an incoming transaction is always a single one (TransactionKind::Single).

As a consequence, any user can change the amount of gas charged for storage and computation in the whole network during each epoch. Here is a proof of concept showing how to exploit this security issue:

Proof of Concept:

1. An attacker modifies the move_call function in his **sui client** to create a transaction for calling ChangeEpoch in a batch transaction (TransactionKind::Batch).

```
/*let kind = TransactionKind::Single(
    self.create_move_call_transaction_kind(params, &mut used_object_ids)
        .await?,
);*/

let res: SingleTransactionKind = SingleTransactionKind::ChangeEpoch(ChangeEpoch {
    epoch: 20,
    storage_charge: 0,
    computation_charge: 0,
});

let kind: TransactionKind = TransactionKind::Batch(vec![res]);
```

2. Once the move_call function is called, the attacker successfully executes the change of epoch info, i.e.: ChangeEpoch transaction.

```
❯ sui client call --package 0xa6cd5ee33d587d6111ca0744b5ab8be9e65ec561 --module m1 --function create --args 100 0xb23a85
e663071615cc3b48328091192b2df59663 --gas-budget 1000
----- Certificate ----
Transaction Hash: 0Q37EXLnOiIzHUzmWHXwCwfNZkuZCW9x6Hq1QHyGvh4=
Transaction Signature: NN0NZcAnNLEkdTjjoPKpZZ94Cz0k/+WsCTqmeLPAPDKIYxexIG3iDMLmKlTzT/rHC1RTDW/gg81M08NnW9OWDQ==@AqIdhtgc
RqH/449PSt3TndogGiFfa161vE7ASY7uMlY=
Signed Authorities Bitmap: RoaringBitmap<[0, 1, 2]>
Transaction Kind : Call
Package ID : 0xa6cd5ee33d587d6111ca0744b5ab8be9e65ec561
Module : m1
Function : create
Arguments : [100, "0xb23a85e663071615cc3b48328091192b2df59663"]
Type Arguments : []
----- Transaction Effects ----
Status : Success
Created Objects:
  - ID: 0xb5000ebf627da464f648f425c10c44010be01018 , Owner: Account Address ( 0xb23a85e663071615cc3b48328091192b2df59663
 )
Mutated Objects:
  - ID: 0x5a5f16cb1edd3233520d94ac4ef77726651334a3 , Owner: Account Address ( 0xb23a85e663071615cc3b48328091192b2df59663
 )
```

Code Location:

The unrestricted execution of ChangeEpoch transaction is controlled in handle_transaction_impl function by ensuring that incoming transaction is not a system one (is_system_tx):

Listing 1: sui-core/src/authority.rs (Lines 298-301)

```
285 async fn handle_transaction_impl(
286  &self,
287  transaction: Transaction,
288 ) -> Result<TransactionInfoResponse, SuiError> {
289  let transaction_digest = *transaction.digest();
290  // Ensure an idempotent answer.
291  if self.database.transaction_exists(&transaction_digest)? {
292   self.metrics.tx_already_processed.inc();
293   let transaction_info = self.make_transaction_info(&
 ↳ transaction_digest).await?;
294   return Ok(transaction_info);
295  }
296
297  // Validators should never sign an external system transaction.
298  fp_ensure!(
299   !transaction.data.kind.is_system_tx(),
300   SuiError::InvalidSystemTransaction
301  );
```

is_system_tx function incorrectly assumes that an incoming transaction is always a single one (TransactionKind::Single), but can be bypassed by calling ChangeEpoch in a batch transaction (TransactionKind::Batch):

Listing 2: sui-types/src/messages.rs (Line 293)

```
290 pub fn is_system_tx(&self) -> bool {
291  matches!(
292    self,
293    TransactionKind::Single(SingleTransactionKind::ChangeEpoch(_))
294  )
295 }
```

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

It is recommended to apply one of the following security measures:

- Update the logic of `is_system_tx` function to verify if the incoming transaction, either single or batch, includes `ChangeEpoch` or not.
- Restrict `ChangeEpoch` from being called in batch transactions.

Remediation plan:

SOLVED: The issue was fixed in commit `e1cfb36`. The `MystenLabs team` solved this issue while the security audit was in progress by restricting certain operations in batch transactions.

3.3 (HAL-03) INADEQUATE HANDLING OF POTENTIAL INTEGER OVERFLOW - MEDIUM

Description:

charge_storage_mutation and deduct_gas functions in **sui-types/src/gas.rs** create situations that could trigger integer overflow issues in the code. As a consequence, unexpected results could derive when handling with gas operations, e.g.: increasing the coin balance of users to almost unlimited amounts.

Despite the likelihood of these events being triggered is very low (only under edge scenarios), it is important to handle them adequately to have a security-in-depth approach.

Code Location:

Code that could create integer overflow issues when handling with gas operations:

Listing 3: sui-types/src/gas.rs (Line 213)

```
195 pub fn charge_storage_mutation(
196  &mut self,
197  old_size: usize,
198  new_size: usize,
199  storage_rebate: GasCarrier,
200 ) -> Result<u64, ExecutionError> {
201  if self.is_unmetered() {
202    return Ok(0);
203  }
204
205  // Computation cost of a mutation is charged based on the sum of
 ↳ the old and new size.
206  // This is because to update an object in the store, we have to
 ↳ erase the old one and
207  // write a new one.
208  let cost = INIT_SUI_COST_TABLE
209      .object_mutation_per_byte_cost
```

```
210         .with_size(old_size + new_size);
211     self.deduct_computation_cost(&cost)?;
212
213     self.storage_rebate += storage_rebate;
214
215     let storage_cost = INIT_SUI_COST_TABLE
216         .storage_per_byte_cost
217         .with_size(new_size);
218     self.deduct_storage_cost(&storage_cost)
219 }
```

Listing 4: sui-types/src/gas.rs (Line 360)

```
352 pub fn deduct_gas(gas_object: &mut Object, deduct_amount: u64,
 ↳ rebate_amount: u64) {
353     // The object must be a gas coin as we have checked in
 ↳ transaction handle phase.
354     let gas_coin = GasCoin::try_from(&*gas_object).unwrap();
355     let balance = gas_coin.value();
356     debug_assert!(balance >= deduct_amount);
357     let new_gas_coin = GasCoin::new(
358         *gas_coin.id(),
359         gas_object.version(),
360         balance + rebate_amount - deduct_amount,
361     );
362     let move_object = gas_object.data.try_as_move_mut().unwrap();
363     move_object.update_contents_and_increment_version(bcs::to_bytes(&
 ↳ new_gas_coin).unwrap());
364 }
```

Risk Level:

Likelihood - 1
Impact - 5

Recommendation:

Set overflow-checks flag to **true** in profile.release in **cargo.toml**. If there exist limitations in the method explained above, it is recommended to use checked arithmetic operations (checked_*) instead because they

will return a **None** value in case an integer overflow happens, which could be handled appropriately in the code.

Finally, it is important to note that assert! and assert_eq! macros are not advisable to use because they could produce a Denial-Of-Service (DoS) in the nodes.

Remediation plan:

SOLVED: The issue was fixed in commit 6008325 for charge_storage_mutation function and in commit 66ac209 for deduct_gas function.

3.4 (HAL-04) INCOMPLETE VALIDITY CHECK ON TRANSACTIONS - LOW

Description:

transaction and handle_certificate functions in **authority_server.rs** do not enforce that is_verified value is **false** before calling the verify function because they rely on having an adequate serialization / deserialization of TransactionEnvelope.

However, from a security-in-depth approach, this validity check should be applied on transactions in case an edge scenario negatively affects the serialization / deserialization process, e.g.: (HAL-01) INADEQUATE DESERIALIZATION OF TRANSACTIONS LEADS TO A DOS OF NODES.

Code Location:

transaction function does not enforce that is_verified value is **false** before calling the verify function:

Listing 5: sui-core/src/authority_server.rs (Lines 252-254,256)

```
246 async fn transaction(
247  &self,
248  request: tonic::Request<Transaction>,
249 ) -> Result<tonic::Response<TransactionInfoResponse>, tonic::
 ↳ Status> {
250  let mut transaction = request.into_inner();
251
252  transaction
253   .verify()
254   .map_err(|e| tonic::Status::invalid_argument(e.to_string()))?;
255  //TODO This is really really bad, we should have different types
 ↳ for signature-verified transactions
256  transaction.is_verified = true;
257
258  let tx_digest = transaction.digest();
```

handle_certificate function does not enforce that is_verified value is **false** before calling the verify function:

Listing 6: sui-core/src/authority_server.rs (Lines 283-285,287)

```
277 async fn handle_certificate(
278  &self,
279  request: tonic::Request<CertifiedTransaction>,
280 ) -> Result<tonic::Response<TransactionInfoResponse>, tonic::
 ↳ Status> {
281  let mut certificate = request.into_inner();
282  // 1) Verify certificate
283  certificate
284   .verify(&self.state.committee.load())
285   .map_err(|e| tonic::Status::invalid_argument(e.to_string()))?;
286  //TODO This is really really bad, we should have different types
 ↳ for signature verified transactions
287  certificate.is_verified = true;
288
289  // 2) Check idempotency
290  let digest = certificate.digest();
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

It is recommended that transaction and handle_certificate functions in **authority_server.rs** should enforce that is_verified value is **false** before calling the verify function (L#253, 284).

Remediation plan:

SOLVED: The issue was fixed in commit 0d00b40.

3.5 (HAL-05) SOME ASSERTS ARE NOT ENFORCED WHEN PACKAGES ARE COMPILED IN RELEASE MODE - LOW

Description:

The debug_assert! and debug_assert_eq! macros are used around the code-base. They are very useful in a testing environment to identify potential errors in the logic, but those asserts are not enforced when used in production code (i.e.: packages compiled in release mode), which could create unexpected situations during the code execution, e.g.: bypassing a consistency check.

Code Location:

Listing 7: Affected resources

```
debug_assert!:
============
sui-adapter/src/adapter.rs: L#163,170,189,194,209,210,221,815
sui-adapter/src/bytecode_rewriter.rs: L#130,134
sui-adapter/src/in_memory_storage.rs: L#96,98
sui-adapter/src/temporary_store.rs: L#255,265,276,297,314,334
sui-verifier/src/private_generics.rs: L#106,111,145

debug_assert_eq!:
===============
sui-adapter/src/adapter.rs: L#682
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

It is recommended to use an adequate error handling method instead of the debug_assert! and debug_assert_eq! macros. On the other hand, using the equivalent assert! and assert_eq! macros is not advisable because they could produce a Denial-Of-Service (DoS) in the nodes.

Remediation plan:

PARTIALLY SOLVED: To the current commit (8999273) at the time of writing the final report, the issue remains for the following resources belonging to sui-adapter and sui-verifier:

- sui-adapter/src/error.rs
- sui-adapter/src/execution_engine.rs
- sui-adapter/src/programmable_transactions/context.rs
- sui-verifier/src/id_leak_verifier.rs
- sui-verifier/src/private_generics.rs

3.6 (HAL-06) POSSIBLE RUST PANICS DUE TO UNSAFE METHODS AND MACROS USAGE - LOW

Description:

Rust helpers methods are used to trigger an error message when it occurs, so that the developer or client receives a detailed description of what went wrong. In test and development environments it is very helpful, but in the case of production systems, such behavior of the node may cause its failure or the death of the thread on which the process is operating, i.e.: a Denial-of-Service (DoS).

Code Location:

Instances of unsafe functions and macros that have been detected:

Listing 8: Affected resources

```
Plain panic!'s:
==============
sui-adapter/src/temporary_store.rs L#321,341
sui-verifier/src/id_leak_verifier.rs L#79,350
sui-types/src/messages.rs L#972,980

Unreachable!'s:
==============
sui-adapter/src/adapter.rs L#460,545,668
sui-types/src/committee.rs L#203
sui-types/src/messages_checkpoint.rs L#157
sui-types/src/object.rs L#190

Unwrap's:
========
sui-adapter/src/adapter.rs
L#180,181,188,291,338,394,453,457,479,528,532,658,947,948
sui-adapter/src/genesis.rs L#30
sui-adapter/src/in_memory_storage.rs L#42
sui-adapter/src/temporary_store.rs L#110
```

```
sui-verifier/src/global_storage_access_verifier.rs L#23
sui-verifier/src/id_leak_verifier.rs L#200,238,261,264,268,272,
287,316,317,322,323,365,372,401,409,412,416,424,425,426
sui-verifier/src/private_generics.rs L#84

Expect:
======
sui-adapter/src/adapter.rs L#331,425,429,445
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

It is recommended to avoid the use of panic!, unreachable!, unwrap or expect macros and functions in production environments as it could crash the node or process on which the specific action is operating. Potential errors or invalid paths in the contract code should be handled by the corresponding error messages, or the specific <Error> types.

Remediation plan:

PARTIALLY SOLVED: To the current commit (8999273) at the time of writing the final report, the issue remains for the following resources belonging to sui-adapter and sui-verifier:

- sui-adapter/src/execution_engine.rs
- sui-adapter/src/lib.rs
- sui-adapter/src/programmable_transactions/context.rs
- sui-adapter/src/programmable_transactions/execution.rs
- sui-adapter/src/programmable_transactions/linkage_view.rs
- sui-verifier/src/id_leak_verifier.rs

3.7 (HAL-07) CLIENTS DENIAL OF SERVICE WHEN PUBLISHING A LARGE PACKAGE - LOW

Description:

When publishing a large package containing a very long codebase, Sui gateway does not allow any other transaction/operations to be made on storage, which actively blocks the client from write & execute communication with the network.

Code Location:

Affected function highlighted in the code snippet:

Listing 9: sui-core/src/authority/authority_store.rs (Line 134)

```
132 impl<S: Eq + Serialize + for<'de> Deserialize<'de>> SuiDataStore<S
 ↳ > {
133     /// Open an authority store by directory path
134     pub fn open<P: AsRef<Path>>(path: P, db_options: Option<
 ↳ Options>) -> Self {
135         let (options, point_lookup) = default_db_options(
 ↳ db_options, None);
136
137         let db = {
138             let path = &path;
139             let db_options = Some(options.clone());
140             let opt_cfs: &[(&str, &rocksdb::Options)] = &[
141                 ("objects", &point_lookup),
142                 ("transactions", &point_lookup),
143                 ("owner_index", &options),
144                 ("certificates", &point_lookup),
145                 ("pending_execution", &options),
146                 ("parent_sync", &options),
147                 ("effects", &point_lookup),
148                 ("sequenced", &options),
149                 ("schedule", &options),
150                 ("executed_sequence", &options),
```

```
                ("batches", &options),
                ("last_consensus_index", &options),
                ("epochs", &options),
            ];
            typed_store::rocks::open_cf_opts(path, db_options,
 ↳ opt_cfs)
        }
        .expect("Cannot open DB.");

        let executed_sequence =
            DBMap::reopen(&db, Some("executed_sequence")).expect("
 ↳ Cannot open CF.");
```

Sample script used for generation of very long package:

Listing 10

```
part1 = """module test::m1 {
    use sui::tx_context::{Self, TxContext};
    use sui::transfer;
    use sui::id::VersionedID;
    struct Object has key, store {
        id: VersionedID,
        value: vector<u64>,
    }
    """
part2 = "public entry fun create"
part3 = """(recipient: address, ctx: &mut TxContext) {
        let value = vector<u64>[];
        let e = 0;
        while (e < 10000) {

            e = e+1;
            std::vector::push_back(&mut value, e);

        };
        transfer::transfer(
            Object { id: tx_context::new_id(ctx), value },
            recipient
        )
    }
    """
part4 = "}"
```

```
temp = ""

print(part1)
for i in range(1,50000):
    temp = part2 + str(i)+part3
    print(temp)
print(part4)
```

Error message generated by Sui-gateway:

Listing 11

```
2022-08-08T12:28:56.079826Z ERROR telemetry_subscribers: panicked
↳ at 'Cannot open DB.: RocksDBError("IO error: While lock file: /
↳ Users/test/.sui/sui_config/client_db/LOCK: Resource temporarily
↳ unavailable")', /Users/test/.cargo/git/checkouts/_empty-
↳ fe9b3228fef81f1d/46f9662/crates/sui-core/src/authority/
↳ authority_store.rs:157:10 panic.file="/Users/test/.cargo/git/
↳ checkouts/_empty-fe9b3228fef81f1d/46f9662/crates/sui-core/src/
↳ authority/authority_store.rs" panic.line=157 panic.column=10
2022-08-08T14:54:34.682757Z ERROR typed_store::rocks: error=
↳ rocksdb error: IO error: While lock file: /Users/test/.sui/
↳ sui_config/client_db/LOCK: Resource temporarily unavailable
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Since the vulnerability applies only to the same machine from which the package is deployed, it is considered a self Denial-of-Service. Nevertheless, it is recommended to introduce multi-threading, so the process of uploading and processing the package does not interfere with the normal operation of the entire client.

Remediation plan:

SOLVED: The issue was fixed in commit f32877f.

3.8 (HAL-08) DEBUG TRAIT IMPLEMENTED ON SENSITIVE VALUES - INFORMATIONAL

Description:

The Debug trait is implemented for structs that contain sensitive information. Implementing this trait may cause sensitive variables to be logged while debugging, which increases the risk of a leak of sensitive information.

Code Location:

Listing 12: narwhal/crypto/src/ed25519.rs

```
14 #[derive(Debug)]
15 pub struct Ed25519PrivateKey(pub ed25519_dalek::SecretKey);
```

Listing 13: narwhal/crypto/src/bls12381.rs

```
40 #[derive(Default, Debug)]
41 pub struct BLS12381PrivateKey {
42     pub privkey: blst::SecretKey,
43     pub bytes: OnceCell<[u8; BLS_PRIVATE_KEY_LENGTH]>,
44 }
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

Remove the Debug trait in production code.

Remediation Plan:

SOLVED: The Debug trait has been replaced with a SilentDebug trait that does not print the struct contents. It can be viewed in the fastcrypto-derive library.

3.9 (HAL-09) SEPARATE RULES FOR GENESIS MODE CAN LEAD TO MEMORY-RELATED ISSUES - INFORMATIONAL

Description:

During genesis mode, memory is initialized without verification of modules and objects size, which may lead to memory overflow errors. Since genesis mode is used only when the chain is running, this is not a big threat to its functioning, but can cause problems in the form of unexpected errors and crashes during startup.

Code Location:

Instances of unsafe functions or macros that have been detected:

Listing 14: sui-adapter/src/in_memory_storage.rs

```
58 impl InMemoryStorage {
59     pub fn new(objects: Vec<Object>) -> Self {
60         let mut persistent = BTreeMap::new();
61         for o in objects {
62             persistent.insert(o.id(), o);
63         }
64         Self { persistent }
65     }
66
67     pub fn get_object(&self, id: &ObjectID) -> Option<&Object> {
68         self.persistent.get(id)
69     }
70
71     pub fn get_objects(&self, objects: &[ObjectID]) -> Vec<Option
↳ <&Object>> {
72         let mut result = Vec::new();
73         for id in objects {
74             result.push(self.get_object(id));
75         }
```

```
76          result
77      }
78
79      pub fn insert_object(&mut self, object: Object) {
80          self.persistent.insert(object.id(), object);
81      }
82
83      pub fn objects(&self) -> &BTreeMap<ObjectID, Object> {
84          &self.persistent
85      }
86
87      pub fn into_inner(self) -> BTreeMap<ObjectID, Object> {
88          self.persistent
89      }
90
91      pub fn finish(
92          &mut self,
93          written: BTreeMap<ObjectID, (ObjectRef, Object)>,
94          deleted: BTreeMap<ObjectID, (SequenceNumber, DeleteKind)>,
95      ) {
96          debug_assert!(written.keys().all(|id| !deleted.
↳ contains_key(id)));
97          for (_id, (_, new_object)) in written {
98              debug_assert!(new_object.id() == _id);
99              self.insert_object(new_object);
100         }
101         for (id, _) in deleted {
102             let obj_opt = self.persistent.remove(&id);
103             assert!(obj_opt.is_some())
104         }
105     }
106 }
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

It is recommended to standardize the process of creating objects in order to obtain, both, the readability of the code and to protect it against unexpected security issues related to memory management.

Remediation plan:

SOLVED: The issue was fixed in commit 4dcea7f.

3.10 (HAL-10) UNMAINTAINED DEPENDENCIES - INFORMATIONAL

Description:

Halborn used automated security scanners to assist with detection of well-known security issues and vulnerabilities. Among the tools used was cargo audit, a security scanner for vulnerabilities reported to the RustSec Advisory Database. All vulnerabilities published in https://crates.io are stored in a repository named **The RustSec Advisory Database**. cargo audit is a human-readable version of the advisory database which performs a scanning on **Cargo.lock**.

Code Location:

Vulnerable and unsupported dependencies are listed below:

ID	package	Short Description
RUSTSEC-2022-0055	axum-core	No default limit put on request bodies. Should be upgraded to >=0.2.8, <0.3.0-rc.1 OR >=0.3.0-rc.2
RUSTSEC-2020-0159	chrono	Potential segfault in ‘localtime_r‘ invocations. Upgrade to >=0.4.20.
RUSTSEC-2022-0046	rocksdb	Out-of-bounds read when opening multiple column families with TTL. Should be upgraded to >=0.19.0
RUSTSEC-2020-0071	time	Potential segfault in the time crate. Should be upgraded to >=0.2.23
RUSTSEC-2021-0139	ansi_term	unmaintained
RUSTSEC-2021-0141	dotenv	Unmaintained
RUSTSEC-2020-0095	difference	Unmaintained
RUSTSEC-2021-0127	serde_cbor	Unmaintained

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to avoid using dependencies and packages that are no longer supported by developers or have publicly known security flaws, even when they are not currently exploitable.

Remediation plan:

PARTIALLY SOLVED: To the current commit (8999273) at the time of writing the final report, the issue remains for the following packages: time, ansi-term and difference.

3.11 (HAL-11) USE OF CLONE TRAIT MAY REDUCE PERFORMANCE - INFORMATIONAL

Description:

The Clone trait is implemented for various structs. Using this trait can reduce code performance as it involves creating a redundant copy of data. It can also increase complexity, as cloned data will not be synchronized with the original data.

Code Location:

The affected files are:

- narwhal/crypto/src/bls12381.rs
- narwhal/crypto/src/ed25519.rs
- sui/crates/sui-types/src/crypto.rs

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Remove uses of Clone trait if they are not necessary or if satisfying the **borrow-checker** is complicated enough to justify this trade-off.

Remediation Plan:

ACKNOWLEDGED: The Mysten Labs team acknowledged this finding.

3.12 (HAL-12) USE OF OSRNG MAY REDUCE PERFORMANCE - INFORMATIONAL

Description:

The struct `rand::rngs::OsRng` is used to generate random numbers in some cases. The documentation for this crate indicates that this struct can be expected to perform worse than other methods.

Code Location:

The affected files are:

- narwhal/crypto/src/bls12381.rs
- sui/crates/sui-types/src/crypto.rs

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider using the `StdRng` struct instead, or the `rand_chacha` crate.

Remediation Plan:

PARTIALLY SOLVED: `OsRng` is no longer used in the `bls12381` implementation.

3.13 (HAL-13) UNUSED FUNCTION WITH POTENTIAL RISKY LOGIC - INFORMATIONAL

Description:

get_resource function in **sui-adapter/src/temporary_store.rs** is never called in the audited codebase. However, if in a later version of the code this function is called and object.data does not match Data::Move(_), it could trigger a crash of nodes (or one of their threads) when calling the unimplemented! macro.

Code Location:

Listing 15: sui-adapter/src/temporary_store.rs (Lines 411-414)

```
387 fn get_resource(
388  &self,
389  address: &AccountAddress,
390  struct_tag: &StructTag,
391 ) -> Result<Option<Vec<u8>>, Self::Error> {
392  let object = match self.read_object(&ObjectID::from(*address)) {
393      Some(x) => x,
394      None => match self.read_object(&ObjectID::from(*address)) {
395          None => return Ok(None),
396          Some(x) => {
397              if !x.is_immutable() {
398                  fp_bail!(SuiError::ExecutionInvariantViolation);
399              }
400              x
401          }
402      },
403  };
404
405  match &object.data {
406      Data::Move(m) => {
407          assert!(struct_tag == &m.type_, "Invariant violation: ill-
 ↳ typed object in storage or bad object request from caller\
408          ");
```

```
409             Ok(Some(m.contents().to_vec()))
410         }
411         other => unimplemented!(
412             "Bad object lookup: expected Move object, but got {:?}",
413             other
414         ),
415     }
416 }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to remove the function if there are no plans to use it in later versions of the codebase. Otherwise, use an adequate error handling method instead of the unimplemented! macro.

Remediation Plan:

SOLVED: The unimplemented! macro has been removed.

3.14 (HAL-14) DEVELOPER NOTE INDICATES POTENTIAL ISSUE - INFORMATIONAL

Description:

A developer comment in the codebase indicates a potential security issue when verifying a batch of BLS signatures, likely the **Rogue Public Key Attack**.

Code Location:

Listing 16: narwhal/crypto/src/bls12381.rs

```
164 // TODO: fix this, the identical message opens up a rogue key
 ↳ attack
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Ensure that, if the code is vulnerable to this attack vector, that this task is tracked and completed before deployed to a production environment. Consider introducing unit tests that simulate this attack to ensure that the code is not vulnerable.

Remediation Plan:

SOLVED: The TODO has been removed from the codebase.

THANK YOU FOR CHOOSING

// HALBORN