Momentum Safe

Momentum Safe

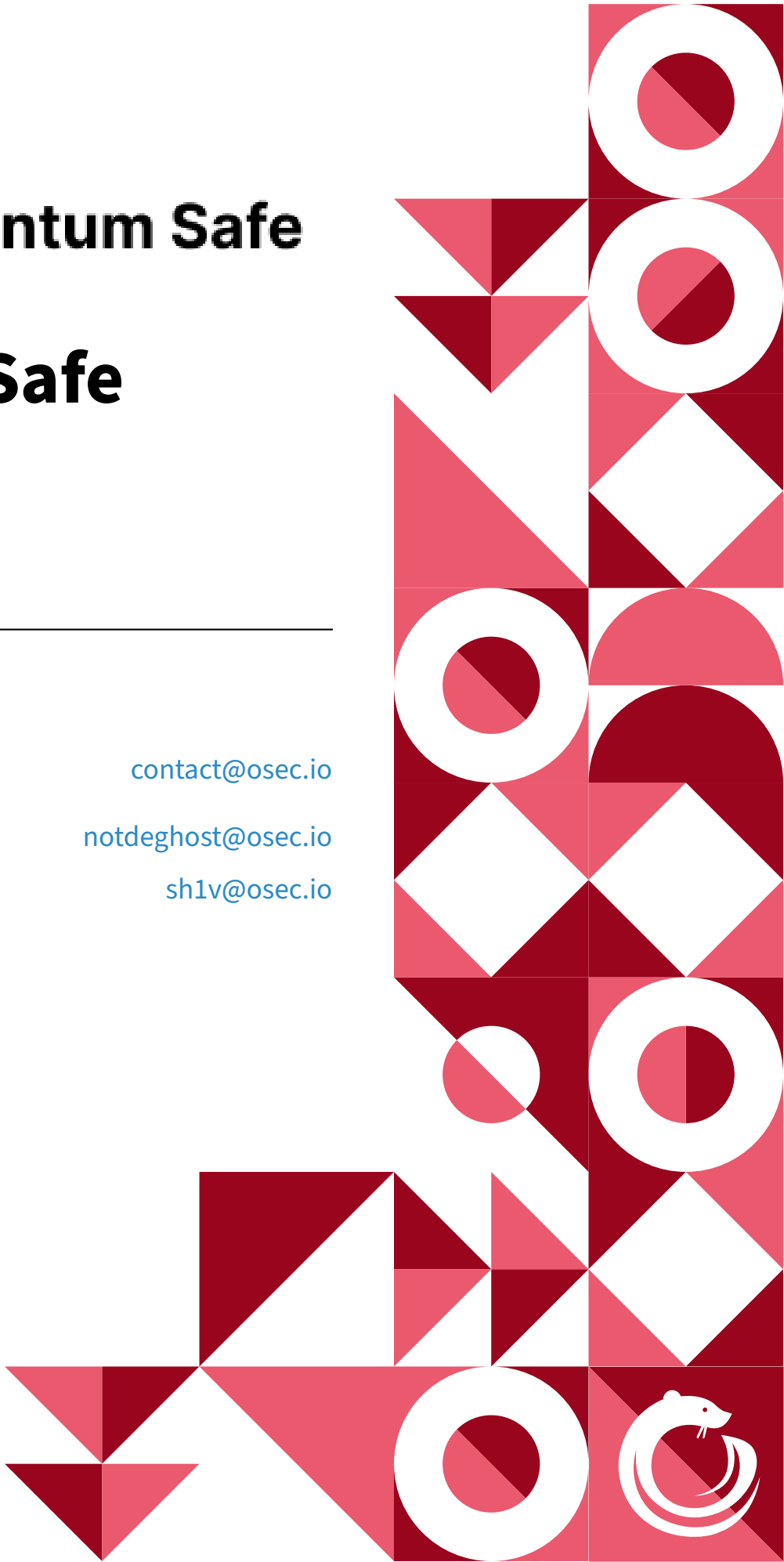
Audit

Presented by:

OtterSec contact@osec.io

Robert Chen notdeghost@osec.io

Shiva Shankar sh1v@osec.io



Contents

Appendices

01 | Executive Summary

Overview

Momentum Safe engaged OtterSec to perform an assessment of the `momentum-safe-mvp` program. This assessment was conducted between September 30th and October 7th, 2022.

Critical vulnerabilities were communicated to the team prior to the delivery of the report to speed up remediation. After delivering our audit report, we worked closely with the team over to streamline patches and confirm remediation. We delivered the final confirmation of the patches October 8th, 2022.

We also note that this report represents the second audit for `momentum-safe-mvp`.

Key Findings

Over the course of this audit engagement, we produced 10 findings total.

In particular, we identified a number of discrepancies between the Aptos and Momentum Safe implementations (OS-MSF-ADV-00, OS-MSF-ADV-01, OS-MSF-ADV-03, OS-MSF-ADV-04). We also identified a gas griefing issue which could force a victim to consume asymptotically more gas (OS-MSF-ADV-02).

We also made general recommendations around clarifying semantic meanings with the framework (OS-MSF-SUG-02), miscellaneous error code issues (OS-MSF-SUG-01), and a discussion of how to apply formal verification (OS-MSF-VER-00, OS-MSF-VER-01).

As part of our work with Aptos Labs, we also reported a framework issue that affected Momentum Safe. Resolved in #4787, we were able to collide resource accounts with a `MultiEd25519` address, allowing for the takeover of the multisig.

Overall, the Momentum Safe team was responsive and a pleasure to work with.

02 | Scope

The source code was delivered to us in a git repository at github.com/Momentum-Safe/momentum-safe-mvp. This audit was performed against commit `a5517d0`.

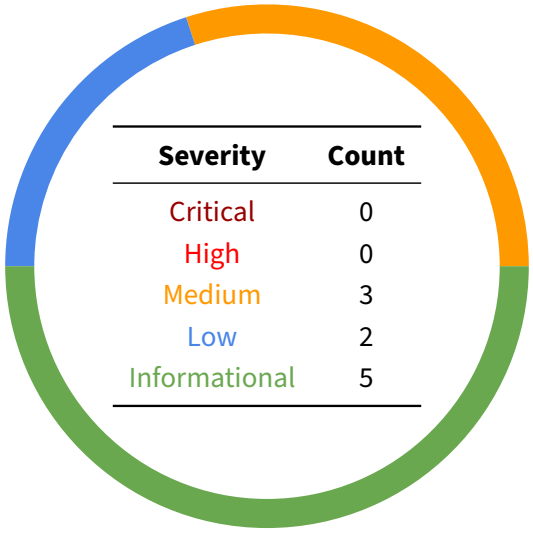
A brief description of the programs is as follows.

Name	Description
momentum-safe-mvp	Aptos multisig wallet building on top of native `MultiEd25519` signatures

03 | Findings

Overall, we report 10 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings don't have an immediate impact but will help mitigate future vulnerabilities.



Severity	Count
Critical	0
High	0
Medium	3
Low	2
Informational	5

04 | Vulnerabilities

Here we present a technical analysis of the vulnerabilities we identified during our audit. These vulnerabilities have *immediate* security implications, and we recommend remediation as soon as possible.

Rating criteria can be found in Appendix A.

ID	Severity	Status	Description
OS-MSF-ADV-00	Medium	Resolved	`type_uleb128` must be validated that the transaction actually is an `EntryFunction`.
OS-MSF-ADV-01	Medium	Resolved	Validate the transaction chain ID to mitigate against griefing.
OS-MSF-ADV-02	Medium	Resolved	`confirm_pending_msafe` gas griefing due to asymptotic differences in gas consumption
OS-MSF-ADV-03	Low	Resolved	The threshold for authenticating a transaction should always be greater than zero.
OS-MSF-ADV-04	Low	Resolved	The max owner limit is off by one since an additional nonce public key is added while deriving the authentication key.

OS-MSF-ADV-00 [med] [resolved] | Missing TransactionPayload Type Validation

Description

The `TransactionPayload` struct contains the payload and its type. Aptos supports payloads of type:

- WriteSet
- Scripts
- ModuleBundle
- EntryFunction

```rust
/// Different kinds of transactions.
pub enum TransactionPayload {
    /// A system maintenance transaction.
    WriteSet(WriteSetPayload),
    /// A transaction that executes code.
    Script(Script),
    /// A transaction that publishes code.
    Module(Module),
    /// A transaction that executes an existing entry function published
    ↪  on-chain.
    EntryFunction(EntryFunction),
}
```

The payload in the `TransactionPayload` struct can be any transaction type, not just `EntryFunction`. This assumption should be validated. For example, many internal functions such as register payload validation assume the layout of the `TransactionPayload` is a `EntryFunction`.

Remediation

Remove the function payload `EntryFunction` deserialization so that the transaction can be of any type.

In addition, in `momentum_safe::register` validate the `type_uleb128` against `EntryFunction`.

Patch

Fixed in 6f30e62.

OS-MSF-ADV-01 [med] [resolved] | Missing Chain ID Validation

Description

In order to uniquely identify a network, a chain id is assigned.

aptos-core/types/src/chain_id.rs RUST

```rust
pub enum NamedChain {
    /// Users might accidentally initialize the ChainId field to 0, hence
     ↪  reserving ChainId 0 for accidental
    /// initialization.
    /// MAINNET is the Aptos mainnet production chain and is reserved for
     ↪  1
    MAINNET = 1,
    // Even though these CHAIN IDs do not correspond to MAINNET, changing
     ↪  them should be avoided since they
    // can break test environments for various organisations.
    TESTNET = 2,
    DEVNET = 3,
    TESTING = 4,
    PREMAINNET = 5,
}
```

Transactions can be replayed from one chain to another if this field isn't properly validated. In the context of a multisig, the transaction can be added but will fail at execution. Nonetheless, this represents a potential UX risk and is worth remediating.

Remediation

Validate `chain_id` in the `validate_txn_payload` function.

DIFF

```diff
--- a/move/sources/momentum_safe.move
+++ b/move/sources/momentum_safe.move
@@ -332,6 +332,9 @@ module msafe::momentum_safe {
         let cur_sn = account::get_sequence_number(msafe_address);
         assert!(cur_sn <= tx_sn, EMSAFE_TX_SEQUENCE_NUMBER_INVALID);

+        // replace 1337 with the desired chain id and 7331 with a custom
     ↪  abort code.
+        assert!(transaction::get_chain_id(&txn) == 1337, 7331);
+
```

```
        let expire = transaction::get_expiration_timestamp_secs(&txn);
        assert!(expire > utils::now_seconds(), EMSAFE_TX_EXPIRED);
```

Patch

Fixed in 6f30e62.

OS-MSF-ADV-02 [med] | Sequential Search Leads To Gas Griefing

Description

When confirming a Momentum Safe registration, the address is removed from the `pending` vector of the owner's `OwnerMomentumSafes` using a linear search. As anyone can register Momentum Safes for the owner, this causes the `pending` vector to grow.

Note that an attacker can register a Momentum Safe in $O(1)$ time, but all future operations will cost $O(n)$ for the victim. This asymptotic difference makes it a viable gas-griefing attack vector.

RUST

```
fun confirm_pending_msafe(
    owner: address,
    msafe: address,
) acquires OwnerMomentumSafes, RegisterEvent {
    assert!(exists<OwnerMomentumSafes>(owner), EADDRESS_NOT_REGISTRERED);
    let pendings = &mut
     ↪  borrow_global_mut<OwnerMomentumSafes>(owner).pendings;

    let (exist, index) = vector::index_of(pendings, &msafe);
    assert!(exist, EMSAFE_NOT_REGISTERED);
    vector::swap_remove(pendings, index);
    add_momentum_safe(owner, msafe);
}
```

RUST

```
public fun index_of<Element>(v: &vector<Element>, e: &Element): (bool,
     ↪  u64) {
    let i = 0;
    let len = length(v);
    while (i < len) {
        if (borrow(v, i) == e) return (true, i);
        i = i + 1;
    };
    (false, 0)
}
```

This operation of finding the Momentum Safe address in the vector could consume an unbounded amount of gas.

Remediation

Instead of using a vector to store the pending addresses, consider a table.

Patch

Fixed in 997875e.

OS-MSF-ADV-03 [low] | Invalid Threshold range

Description

The threshold represents the minimum number of signatures required for authenticating a transaction.

The Aptos multisig implementation validates that the threshold is not zero.

aptos-core/crates/aptos-crypto/src/multi_ed25519.rs RUST

```rust
impl MultiEd25519PublicKey {
/// Construct a new MultiEd25519PublicKey.
/// --- Rules ---
/// a) threshold cannot be zero.
/// b) public_keys.len() should be equal to or larger than threshold.
/// c) support up to MAX_NUM_OF_KEYS public keys.
pub fn new(
    public_keys: Vec<Ed25519PublicKey>,
    threshold: u8,
) -> std::result::Result<Self, CryptoMaterialError> {
    let num_of_public_keys = public_keys.len();
    if threshold == 0 || num_of_public_keys < threshold as usize {
        Err(CryptoMaterialError::ValidationError)
    } else if num_of_public_keys > MAX_NUM_OF_KEYS {
        Err(CryptoMaterialError::WrongLengthError)
    } else {
        Ok(MultiEd25519PublicKey {
            public_keys,
            threshold,
        })
    }
}
```

Remediation

Add an assertion check to ensure that the threshold is greater than 0.

DIFF

```
--- a/move/sources/creator.move
+++ b/move/sources/creator.move
@@ -540,6 +540,10 @@ module msafe::creator {
         let owners_count = vector::length(&owners);
         assert!(owners_count > 1, EOWNERS_LESS_THAN_TWO);
```

```
         assert!(owners_count <= MAX_MSAFE_OWNERS_LIMIT,
    ↪  EMSAFE_OWNERS_EXCEED_LIMIT);
+
+        // replace the 1337 with invalid threshold error code
+        assert!((threshold as u64) > 0, 1337);
+
         assert!((threshold as u64) <= owners_count,
    ↪  ETHRESHOLD_BEYOND_PUBLIC_KEYS);

         let public_keys = get_public_keys(&owners);
```

Patch

Fixed in 921ad74

OS-MSF-ADV-04 [low] | Inaccurate MAX_MSAFE_OWNERS_LIMIT

Description

An additional nonce key is added in `derive_multisig_auth_key`. This means that the actual `MAX_MSAFE_OWNERS_LIMIT` should be one less than the Aptos enforced maximum of 32, or a total of 31.

move/sources/utils.move RUST

```
public fun derive_multisig_auth_key(
    pubkeys: vector<vector<u8>>,
    threshold: u8,
    nonce: u64,
    module_address: address
): vector<u8> {
    // Write the owner public keys to buffer.
    let pk_bytes = vector::empty<u8>();
    let i = 0;
    while (i < vector::length(&pubkeys)) {
        vector::append(&mut pk_bytes, *vector::borrow(&pubkeys, i));
        i = i + 1;
    };

    // Write the additional nonce as the last public key.
    vector::append(&mut pk_bytes, nonce_to_public_key(nonce,
    ↪  module_address));

    // Write signature threshold and multi-ed25519 schema identifier.
    vector::push_back(&mut pk_bytes, threshold);
    vector::push_back(&mut pk_bytes, 1);

    // Derive the authentication key from the buffer.
    hash::sha3_256(pk_bytes)
}
```

Remediation

Change the MAX_MSAFE_OWNERS_LIMIT from 32 to 31.

DIFF

```
--- a/move/sources/creator.move
+++ b/move/sources/creator.move
```

```
@@ -146,7 +146,7 @@ module msafe::creator {
     /// the real-time network situation.
     const MIN_REGISTER_TX_GAS: u64 = 2000;
     const MIN_REGISTER_TX_GAS_PRICE: u64 = 1;
-    const MAX_MSAFE_OWNERS_LIMIT: u64 = 32;
+    const MAX_MSAFE_OWNERS_LIMIT: u64 = 31;

     /// Wallet creation information stored under the deployer's account.
     struct PendingMultiSigCreations has key, drop {
```

Patch

Fixed in 353c75e.

05 | General Findings

Here we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they represent anti patterns and could lead to security issues in the future.

ID	Description
OS-MSF-SUG-00	The owner's wallet list should not contain duplicate entries.
OS-MSF-SUG-01	Two error codes in `momentum_safe.move` have the same value
OS-MSF-SUG-02	Note that `MAX_MSAFE_OWNERS_LIMIT` has a semantic meaning with respect to Aptos multisig implementation.

OS-MSF-SUG-00 | Duplicate Address Prevention

Description

When initializing a new wallet a list of owner addresses is provided. This vector should be explicitly checked for duplicate addresses.

Remediation

Define a helper function in the `utils.move` which checks if a given vector has duplicate entries.

utils.move RUST

```
    public fun has_unique_entries(v: &vector<address>): bool {
        let temp = vector::empty<address>();
        let i = 0;
        while (i < vector::length(v)) {
            let element = *vector::borrow(v, i);
            if (!vector::contains(&temp, &element)){
                vector::push_back(&mut temp, element);
            };
            i = i + 1
        };
        vector::length(v) == vector::length(&temp)
    }

    #[test]
    #[expected_failure]
    fun test_unique() {
        let v = vector::empty<address>();
        vector::push_back(&mut v, @0x1);
        vector::push_back(&mut v, @0x2);
        vector::push_back(&mut v, @0x1);

        assert!(has_unique_entries(&v), 1337);
    }
```

Use the function to check for duplicate owners of a Momentum Safe in `init_wallet_creation_internal`.

DIFF

```
--- a/move/sources/creator.move
+++ b/move/sources/creator.move
@@ -538,6 +538,8 @@ module msafe::creator {
```

```
    ) acquires PendingMultiSigCreations, MultiSigCreationEvent {
        // Check the input parameters.
        let owners_count = vector::length(&owners);
+       // replace 1337 with error code for dup owners
+       assert!(utils::has_unique_entries(&owners), 1337)
        assert!(owners_count > 1, EOWNERS_LESS_THAN_TWO);
```

Patch

Fixed in 6f30e62.

OS-MSF-SUG-01 | Error Code Conflict

Description

In `momentum_safe.move`, `EMSAFE_TX_SEQUENCE_NUMBER_INVALID` and `EMSAFE_TX_SENDER_INVALID` have the same value.

Remediation

Change the value of one of the error codes.

DIFF

```
--- a/move/sources/momentum_safe.move
+++ b/move/sources/momentum_safe.move
@@ -88,7 +88,7 @@ module msafe::momentum_safe {
     const EMSAFE_TX_SEQUENCE_NUMBER_INVALID: u64 = 2;

     /// Sender of transaction must be momentum safe address.
-    const EMSAFE_TX_SENDER_INVALID: u64 = 2;
+    const EMSAFE_TX_SENDER_INVALID: u64 = 4;
```

Patch

Fixed in 921ad74

OS-MSF-SUG-02 | Denote MAX_MSAFE_OWNERS_LIMIT Semantic Meaning

Description

The maximum limit for the number of owners of a msafe is derived from the aptos-core multisig crate's `MAX_NUMBER_OF_PUBLIC_KEYS`.

This semantic dependency should be noted.

aptos-core/aptos-move/framework/aptos-stdlib/sources/cryptography/multi_ed25519.move RUST

```
const BITMAP_NUM_OF_BYTES: u64 = 4;

/// Max number of ed25519 public keys allowed in multi-ed25519 keys
const MAX_NUMBER_OF_PUBLIC_KEYS: u64 = 32;
```

Since a nonce public key is added while deriving the authentication key the values will differ by 1. This should be mentioned in the comments while defining the `MAX_MSAFE_OWNERS_LIMIT`.

Remediation

Add a comment stating that the const `MAX_MSAFE_OWNERS_LIMIT` is derived from the Aptos `MAX_NUMBER_OF_PUBLIC_KEYS`.

DIFF

```
--- a/move/sources/creator.move
+++ b/move/sources/creator.move
@@ -146,7 +146,10 @@ module msafe::creator {
     /// the real-time network situation.
     const MIN_REGISTER_TX_GAS: u64 = 2000;
     const MIN_REGISTER_TX_GAS_PRICE: u64 = 1;
+
+    /// Derived constant from aptos core multi-sign implementation
+    /// The allowed max in aptos is 32, since we add a custom nonce
    ↪  pubkey
+    /// this is off by one to the original constant defined in the aptos
    ↪  core.
     const MAX_MSAFE_OWNERS_LIMIT: u64 = 31;
```

Patch

Fixed in 921ad74

06 | Formal Verification

Here we present a discussion about the formal verification of smart contracts. We include example specifications, recommendations, and general ideas to formalize critical invariants.

ID	Description
OS-MSF-VER-00	Resolving issues with bitwise operations
OS-MSF-VER-01	Specifications around formalizing address sizing when represented as u8 vectors

OS-MSF-VER-00 | Bitwise Operations

The Move Prover does not work on code that contains bitwise operations such as | and &.

bytecode_translator.rs RUST

```
    BitOr | BitAnd | Xor => {
        env.error(&loc, "Unsupported operator");
        emitln!(
            writer,
            "// bit operation not supported: {:?}\nassert false;",
            bytecode
        );
    }
```

In order to get the Move Prover to work, rewrite `decode_u64` and `decode_uleb128` to use plain arithmetic operations.

For example,

transaction.move RUST

```
    fun decode_u64(
        r: &mut Reader
    ): u64 {
        let v64 = (decode_u8(r) as u64);
        v64 = v64 | ((decode_u8(r) as u64) << 8);
        v64 = v64 | ((decode_u8(r) as u64) << 16);
        v64 = v64 | ((decode_u8(r) as u64) << 24);
        v64 = v64 | ((decode_u8(r) as u64) << 32);
        v64 = v64 | ((decode_u8(r) as u64) << 40);
        v64 = v64 | ((decode_u8(r) as u64) << 48);
        v64 = v64 | ((decode_u8(r) as u64) << 56);
        v64
    }
```

could be rewritten as

transaction.move RUST

```
    fun decode_u64(
        r: &mut Reader
    ): u64 {
        let v64 = (decode_u8(r) as u64);
        v64 = v64 + ((decode_u8(r) as u64) << 8);
```

```
        v64 = v64 + ((decode_u8(r) as u64) << 16);
        v64 = v64 + ((decode_u8(r) as u64) << 24);
        v64 = v64 + ((decode_u8(r) as u64) << 32);
        v64 = v64 + ((decode_u8(r) as u64) << 40);
        v64 = v64 + ((decode_u8(r) as u64) << 48);
        v64 = v64 + ((decode_u8(r) as u64) << 56);
        v64
    }
```

OS-MSF-VER-01 | Address Sizing

When representing public key addresses as byte vectors, it might be worthwhile to add invariants to ensure that the vectors are indeed properly sized at 32 bytes.

Note that formal verification is an iterative process, and the recommendations in this section are meant to be synergistic. In other words, implementing one suggestion will help with the verification of other invariants.

1. `registry::get_public_key_verified` returns a `vector<u8>` which represents an address. Verify that the returned byte array is indeed 32 bytes as expected.

   registry.move RUST

   ```
   spec get_public_key_verified {
       ensures len(result) == 32;
   }
   ```

2. `OwnerMomentumSafes` contains a `public_key` field which represents the public key of the user when they registered into Momentum Safe. Verify the sizing with a data invariant.

   registry.move RUST

   ```
   spec OwnerMomentumSafes {
       invariant len(public_key) == 32;
   }
   ```

   Note that in order to verify this invariant, you will need to add sizing assertions to functions where `public_key` is modified, such as `registry::register`.

3. `creator::get_public_keys`, converts a vector of `addresss` into a vector of `vector<u8>`. Similarly, verify the returned address sizing with an invariant.

A | Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings can be found in the General Findings section.

Critical	Vulnerabilities that immediately lead to loss of user funds with minimal preconditions Examples: • Misconfigured authority or access control validation • Improperly designed economic incentives leading to loss of funds
High	Vulnerabilities that could lead to loss of user funds but are potentially difficult to exploit. Examples: • Loss of funds requiring specific victim interactions • Exploitation involving high capital requirement with respect to payout
Medium	Vulnerabilities that could lead to denial of service scenarios or degraded usability. Examples: • Malicious input that causes computational limit exhaustion • Forced exceptions in normal user flow
Low	Low probability vulnerabilities which could still be exploitable but require extenuating circumstances or undue risk. Examples: • Oracle manipulation with large capital requirements and multiple transactions
Informational	Best practices to mitigate future security risks. These are classified as general findings. Examples: • Explicit assertion of critical internal invariants • Improved input validation