Attack Move Verifiers :
Our Experiences of Exploiting and Enhancing Move-based Blockchain

Zhaofeng Chen
Reseacher@CertiK Skyfall Team

CERTIK

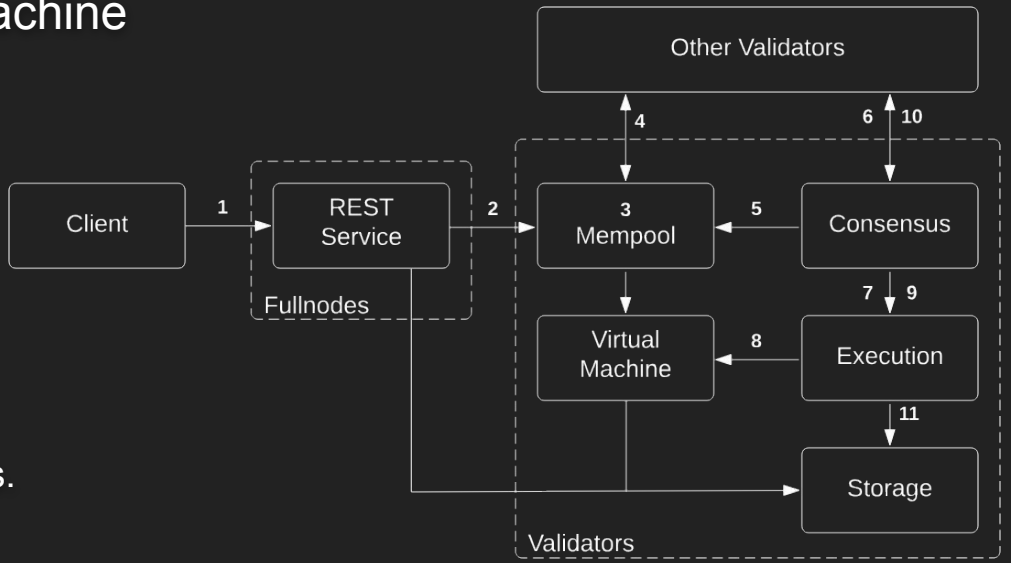
Blockchain Infrastructure

An Abstraction of Global World Machine

- Decentrialized Network
- Permissionless
- Public states
  - Published Contract
  - Ledger with assets

User interact with blockchain

- R/W: Submit transactions
  - Publish/Execute smart contracts.
- R: Query onchain states
  - RPC



CERTIK

The Demand Of A More Secure SC Language

Smart contract (SC) hacks: 100M+ loss

- Define new asset types
- Read, write, and transfer assets
- Check access control policies

Existing SC language does not support well for

- Safe abstractions for custom assets, ownership, access control
- Temporarily borrowing an asset in a callee function
- Declaring an asset type in contract 1 that is used by contract 2

rekt

1. **Ronin Network - REKT** *Unaudited*
   $624,000,000 | 03/23/2022
2. **Poly Network - REKT** *Unaudited*
   $611,000,000 | 08/10/2021
3. **BNB Bridge - REKT** *Unaudited*
   $586,000,000 | 10/06/2022
4. **SBF - MASK OFF** *N/A*
   $477,000,000 | 11/12/22
5. **Wormhole - REKT** *Neodyme*
   $326,000,000 | 02/02/2022
6. **Mixin Network - REKT** *N/A*
   $200,000,000 | 09/23/2023
7. **Euler Finance - REKT** *Sherlock*
   $197,000,000 | 03/13/2023
8. **BitMart - REKT** *N/A*
   $196,000,000 | 12/04/2021
9. **Nomad Bridge - REKT** *N/A*
   $190,000,000 | 08/01/2022
10. **Beanstalk - REKT** *Unaudited*
    $181,000,000 | 04/17/2022

CERTIK

Move In The New Generation Of Blockchains

A new smart contract language for Layer1 blockchains with rich unique security features

- New programming paradigm: **Ownership**, **Static Types**, etc.
- Safer SC languages, advanced testing/analysis/verification tools

*"If you **give** me a coin, I will **give** you a car title"*

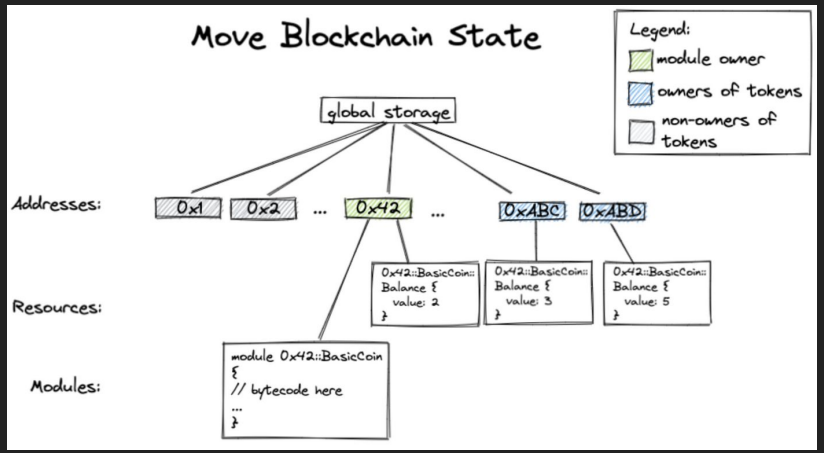
```
fun buy(c: Coin): CarTitle
```

*"If you **show** me your title and **pay** a fee, I will **give** you a car registration"*

```
fun register(c: &CarTitle, fee: Coin): CarRegistration {...}
```

Move In A Nutshell - Resource Abstraction

```
address 0x2 {
    module Coin {
        struct Coin {
            value: u64,
        }

        public fun mint(value: u64): Coin {
            Coin { value }
        }

        public fun burn(coin: Coin): u64 {
            let Coin { value } = coin; // unpack
            value
        }
    }
}
```



- Resource Identifier: $Address:$Module
- Customize Type: struct (pack/unpack)
- Function Visibility

https://github.com/move-language/move/tree/main/language/documentation/tutorial

CERTIK

Move In A Nutshell - Ownership

```
address 0x2 {
    module Coin {
        struct Coin {
            value: u64,
        }

        public fun mint(value: u64): Coin {
            Coin { value }
        }

        public fun burn(coin: Coin): u64 {
            let Coin { value } = coin; // unpack
            value
        }
    }
}
```

Protect Against

```
fun no_copy(c: Coin) {
    let x = copy c; // error

    let y = &c;
    let copied = *y; // error
}
```

Duplication

```
fun no_double_spend(c: Coin) {
    pay(move c);
    pay(move c); // error
}
```

Double Spending

```
fun no_drop() {
    let _coin = Coin::mint(100); // error
}
```

Destruction

Ensures that digital assets behave like physical ones
Type system prevents misuse of asset values

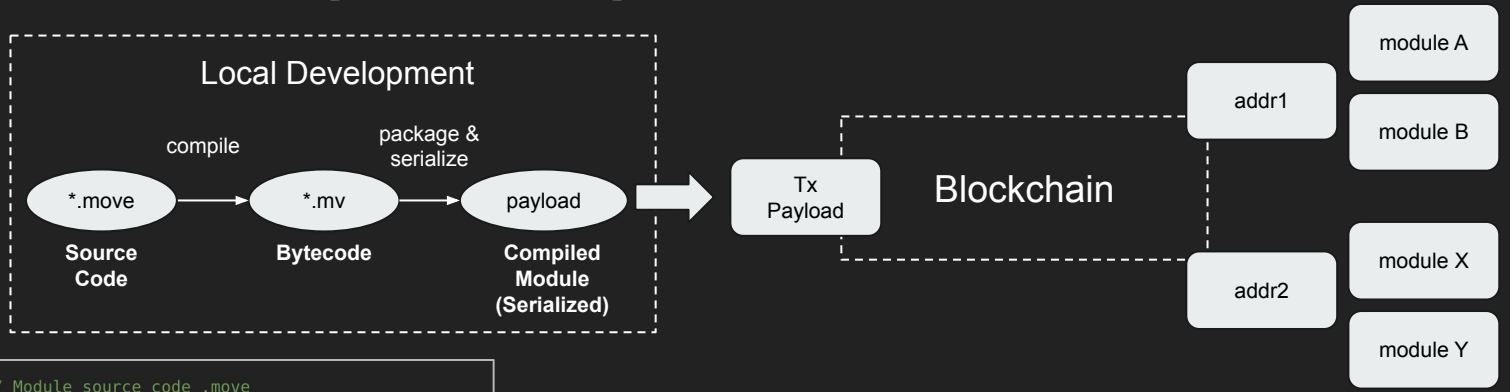
Move: A Secure Programming Paradigm For Sc Development

- Static Typing
  - Ownership, borrow, mutation semantic
  - No type conversions
- Resource-Oriented Programming Model
  - No Duplication
    - resource cannot be copied by default
    - only moved between storage locations
  - No Drop

Move Background

Security Enforcement in Move

Move Developer's Perspective - Publish Module & Execute




```
// Module source code .move
module 0xCAFE::BasicCoin {
   struct Coin has key {
       value: u64,
   }

   public fun mint(account: signer, value: u64) {
       move_to(&account, Coin { value })
   }
}
```

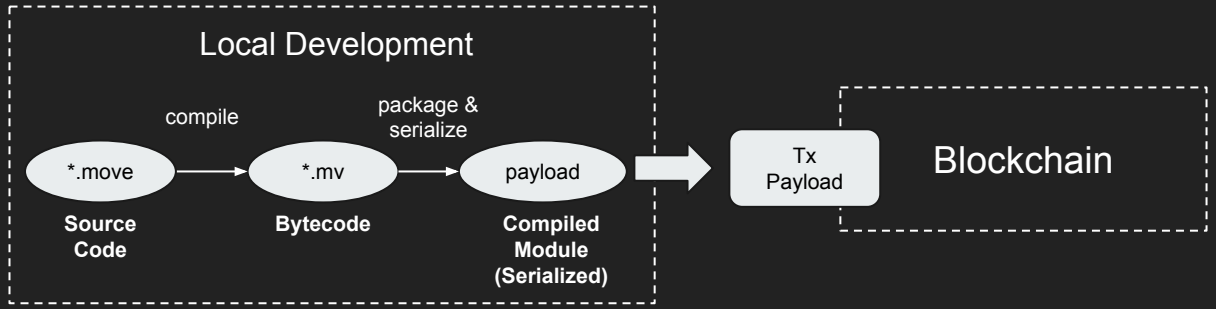
```
0xCAFE::App {
  use 0xCaFE::BasicCoin;

  pub entry fun add_balance(s: signer) {
     assert!(signer::address_of(&s) == @0x42, 0);
     BasicCoin::mint(s, 1024);
  }
}
```

Different Transaction Types

- Publish Module
- Execute **Entrypoint Function**

CERTIK

Move Developer's Perspective - Type Safety Enforcement




```
public fun burn_twice(coin : Coin) {
    burn(coin);
    burn(coin);
}
```

Enforced By Move Compiler

```
$ aptos move compile --package-dir ~/test/Demo/
error[E06002]: use of unassigned variable
   ┌─ /home/ubuntu/test/Demo/sources/Coin.move:50:11
   │
49 │         burn(coin);
   │              ----
   │              │
   │              The value of 'coin' was previously moved here.
   │              Suggestion: use 'copy coin' to avoid the move.
50 │         burn(coin);
   │              ^^^^ Invalid usage of previously moved variable
   │
'coin'.
```

CERTIK

Attacker's Perspective - Module Formats

```
// Module source code .move
module 0xCAFE::BasicCoin {
   struct Coin has key {
       value: u64,
   }

   public fun mint(account: signer, value: u64) {
       move_to(&account, Coin { value })
   }
}
```

Source Code

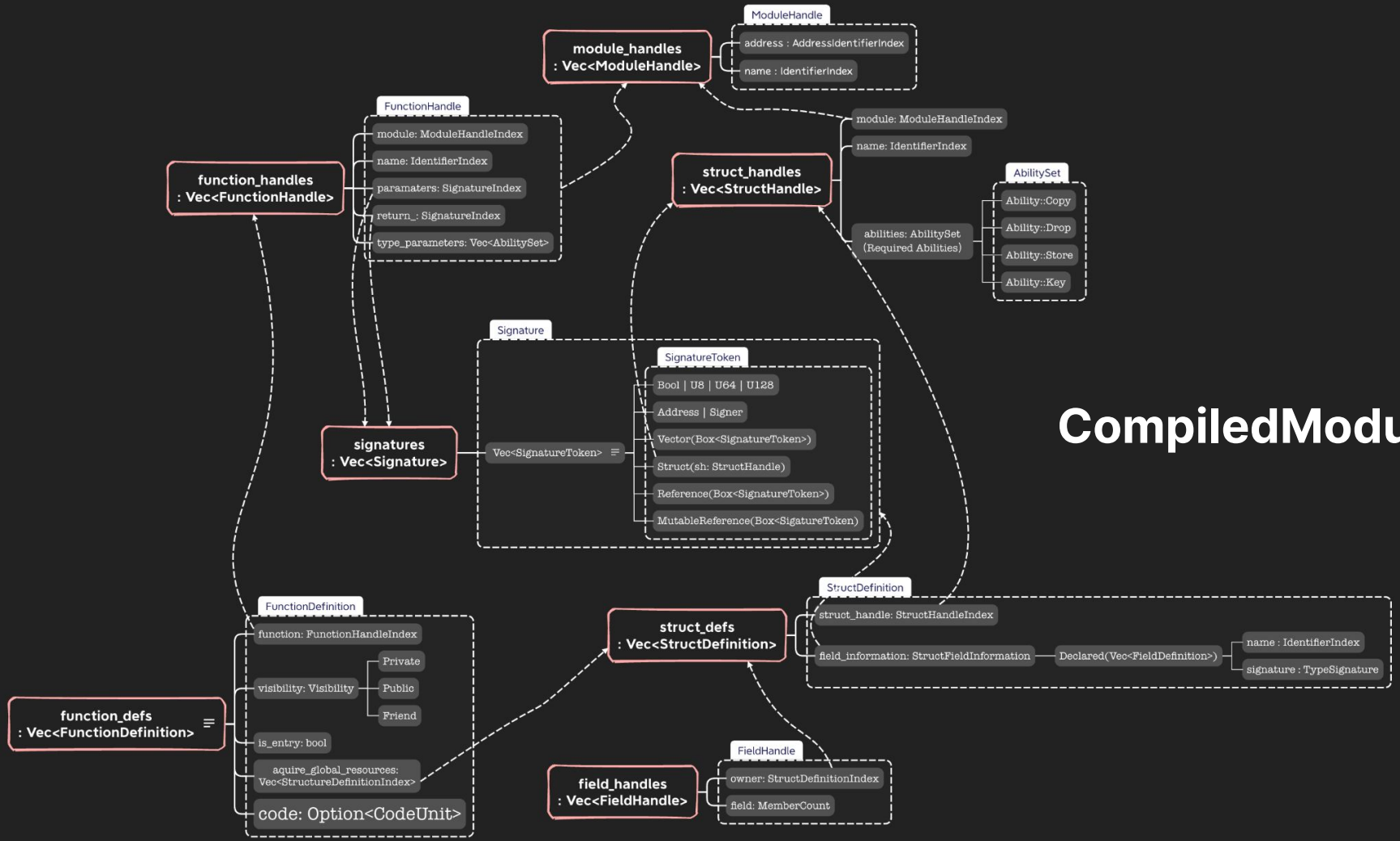
```
// Move disassembled bytecode
module cafe.BasicCoin {
struct Coin has key {
    value: u64
}


public mint(account: signer, value: u64) {
B0:
    0: ImmBorrowLoc[0](account: signer)
    1: MoveLoc[1](value: u64)
    2: Pack[0](Coin)
    3: MoveTo[0](Coin)
    4: Ret
}
}
```

Bytecode

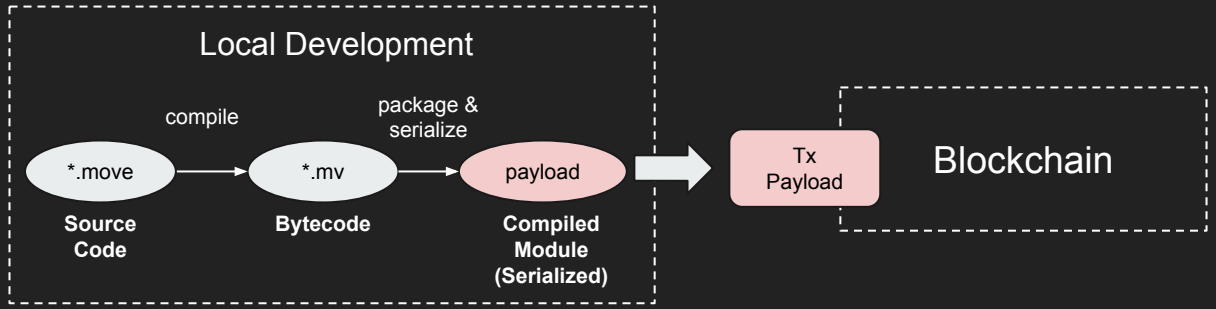
```
pub struct CompiledModule {
    /// Version number found during deserialization
    pub version: u32,
    ...
    /// Handles to external dependency modules and self.
    pub module_handles: Vec<ModuleHandle>,
    /// Handles to external and internal types.
    pub struct_handles: Vec<StructHandle>,
    /// Handles to external and internal functions.
    pub function_handles: Vec<FunctionHandle>,
    ...
    /// Locals signature pool. The signature for all locals of the
functions defined in the module.
    pub signatures: SignaturePool,
    /// All identifiers used in this module.
    pub identifiers: IdentifierPool,
    /// All address identifiers used in this module.
    pub address_identifiers: AddressIdentifierPool,
    /// Constant pool. The constant values used in the module.
    pub constant_pool: ConstantPool,
    ...
    /// Types defined in this module.
    pub struct_defs: Vec<StructDefinition>,
    /// Function defined in this module.
    pub function_defs: Vec<FunctionDefinition>,
}
```

Payload

(before serialization)


ModuleHandle
module_handles
: Vec<ModuleHandle>
address : AddressIdentifierIndex
name : IdentifierIndex
FunctionHandle
module: ModuleHandleIndex
name: IdentifierIndex
paramaters: SignatureIndex
return_: SignatureIndex
type_parameters: Vec<AbilitySet>
function_handles
: Vec<FunctionHandle>
module: ModuleHandleIndex
name: IdentifierIndex
struct_handles
: Vec<StructHandle>
AbilitySet
Ability::Copy
Ability::Drop
Ability::Store
Ability::Key
abilities: AbilitySet
(Required Abilities)
Signature
SignatureToken
Bool | U8 | U64 | U128
Address | Signer
Vector(Box<SignatureToken>)
Struct(sh: StructHandle)
Reference(Box<SignatureToken>)
MutableReference(Box<SigatureToken)
signatures
: Vec<Signature>
Vec<SignatureToken>
CompiledModule
StructDefinition
struct_handle: StructHandleIndex
field_information: StructFieldInformation
Declared(Vec<FieldDefinition>)
name : IdentifierIndex
signature : TypeSignature
FunctionDefinition
function: FunctionHandleIndex
visibility: Visibility
Private
Public
Friend
is_entry: bool
aquire_global_resources:
Vec<StructureDefinitionIndex>
code: Option<CodeUnit>
function_defs
: Vec<FunctionDefinition>
struct_defs
: Vec<StructDefinition>
FieldHandle
owner: StructDefinitionIndex
field: MemberCount
field_handles
: Vec<FieldHandle>

Attacker's Perspective - Bypass Compiler's Enforcement




```
public fun burn_twice(coin : Coin) {
    burn(coin);
    burn(coin);
}
```

```
$ aptos move compile --package-dir ~/test/Demo/
error[E06002]: use of unassigned variable
   ┌─ /home/ubuntu/test/Demo/sources/Coin.move:50:11
   │
49 │         burn(coin);
   │              ----
   │              │
   │              The value of 'coin' was previously moved here.
   │              Suggestion: use 'copy coin' to avoid the move.
50 │         burn(coin);
   │              ^^^^ Invalid usage of previously moved variable 'coin'.
```

```
public burn_twice(Arg0:: Coin) {
B0:
        0: MoveLoc[0](Arg0: Coin)
        1: Call[0](burn(Coin): u64)      //   call burn(coin) 1st time
        2: MoveLoc[0](Arg0: Coin)
        3: Call[0](burn(Coin): u64)      //   call burn(coin) 2nd time;
        4: Pop
        5: Pop
        6: Ret
}
```

Craft Voilations in Bytecode

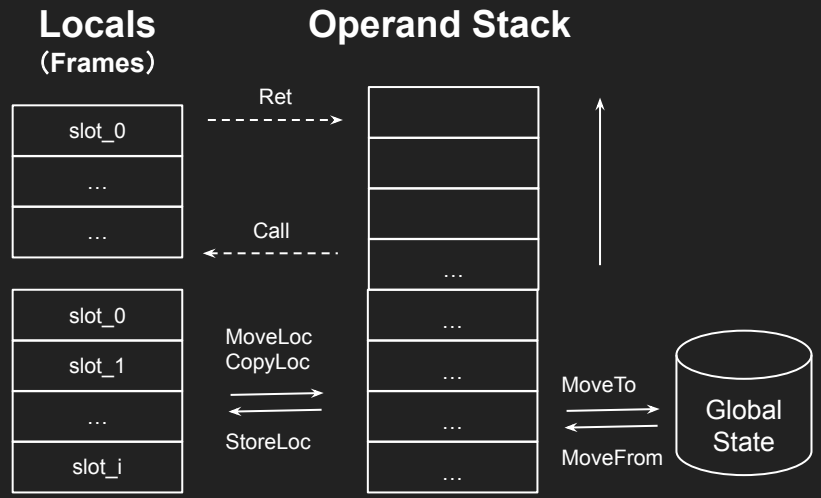
Move Virtual Machine - Stack Machine

Each call stack has its own local variables

- function arguments: from caller
- locals: from other vars or global states

Intereptation each instruction

- Computation on the operand stack
- Move data between locals and operand stack
- Create/destroy call stack frames



CERTIK

Move ByteCode - Encoded With TypeInfo

Global Access with *$struct_definition_index*

- MoveFrom(*$sd_idx*), MoveTo(*$sd_idx*), BorrowGlobal(*$sd_idx*)

Locals Access with *$local_slot_index*

- MoveLoc(*$ls_idx*), CopyLoc(*$ls_idx*), StoreLoc(*$ls_idx*), BorrowLoc(*$ls_idx*)

Structs Access with *$struct_definition_index*

- Pack(*$sd_idx*), Unpack(*$sd_idx*), BorrowField(*$sd_idx*),

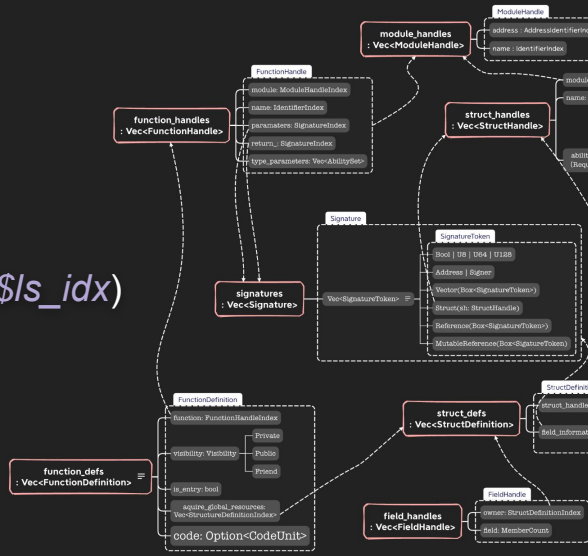
Vector

- VecPack<T, N>, VecUnpack<T, N>

References: ReadRef, WriteRef

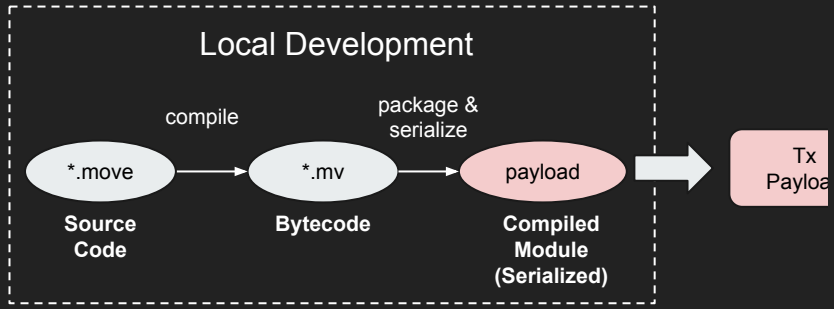
Control-flow: Call<p>, Ret, Br, BrTrue, BrFalse, Abort

Stack: Pop, Not, Add, Sub, Mul, Div, BitOr, BitAnd, Xor, Lt, Gt, Le, Ge, Or, And, Eq, Neq, Shl, Shr



CERTIK

Attacker's Perspective - Malform CompiledModule



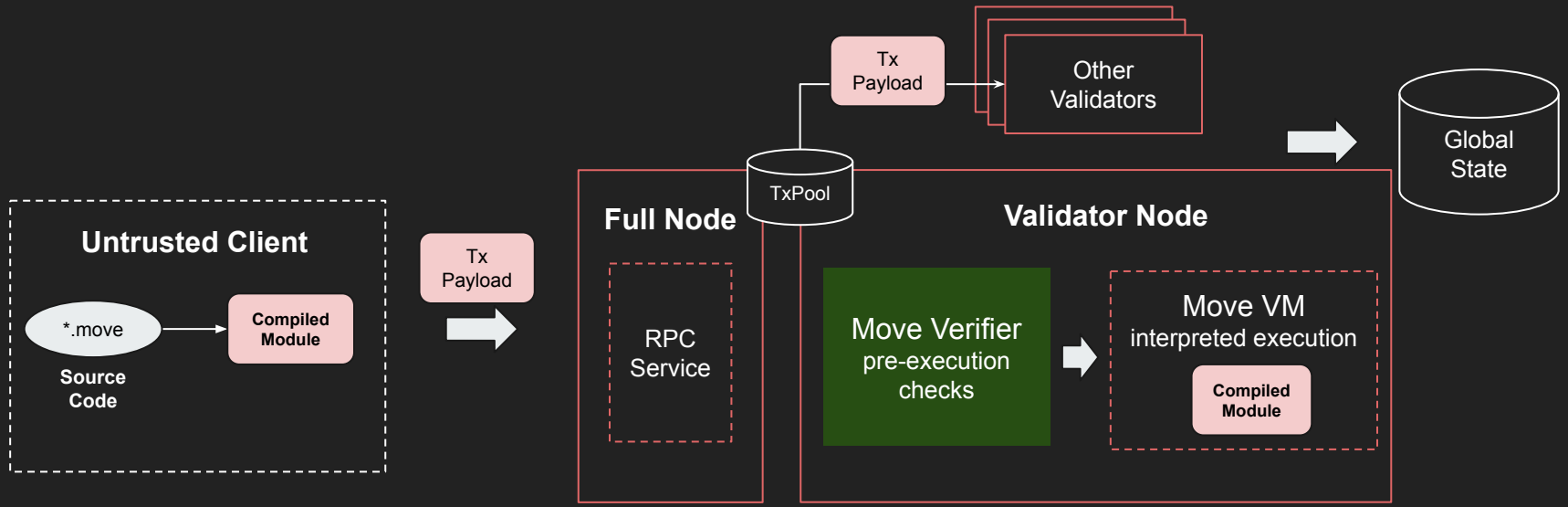

```
public burn_twice(Arg0:: Coin) {
B0:
        0: MoveLoc[0](Arg0: Coin)
        1: Call[0](burn(Coin): u64)      //   call burn(coin) 1st time
        2: MoveLoc[0](Arg0: Coin)
        3: Call[0](burn(Coin): u64)      //   call burn(coin) 2nd time;
        4: Pop
        5: Pop
        6: Ret
}
```

```
pub struct CompiledModule {
    /// Version number found during deserialization
    pub version: u32,
    ...
    /// Handles to external dependency modules and self.
    pub module_handles: Vec<ModuleHandle>,
    /// Handles to external and internal types.
    pub struct_handles: Vec<StructHandle>,
    /// Handles to external and internal functions.
    pub function_handles: Vec<FunctionHandle>,
    ...
    /// Locals signature pool. The signature for all locals of the
functions defined in the module.
    pub signatures: SignaturePool,
    /// All identifiers used in this module.
    pub identifiers: IdentifierPool,
    /// All address identifiers used in this module.
    pub address_identifiers: AddressIdentifierPool,
    /// Constant pool. The constant values used in the module.
    pub constant_pool: ConstantPool,
    ...
    /// Types defined in this module.
    pub struct_defs: Vec<StructDefinition>,
    /// Function defined in this module.
    pub function_defs: Vec<FunctionDefinition>,
}
```

Fully controllable TxPayload

CERTIK

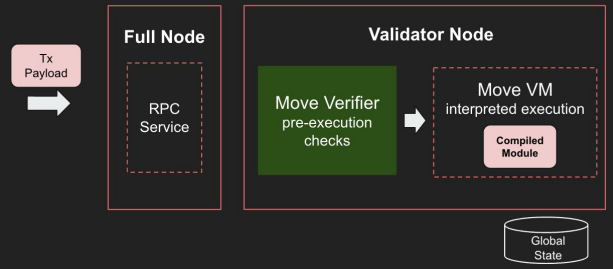
Blockchain's Perspective: On-chain Security Enforcement



Designed to defend against malformed TxPayload

CERTIK

MV Verifier: Security Checks




```
pub struct CompiledModule {
    /// Version number found during deserialization
    pub version: u32,
    ...
    /// Handles to external dependency modules and self.
    pub module_handles: Vec<ModuleHandle>,
    /// Handles to external and internal types.
    pub struct_handles: Vec<StructHandle>,
    /// Handles to external and internal functions.
    pub function_handles: Vec<FunctionHandle>,
    ...
    /// Locals signature pool. The signature for all locals of the
functions defined in the module.
    pub signatures: SignaturePool,
    /// All identifiers used in this module.
    pub identifiers: IdentifierPool,
    /// All address identifiers used in this module.
    pub address_identifiers: AddressIdentifierPool,
    /// Constant pool. The constant values used in the module.
    pub constant_pool: ConstantPool,
    ...
    /// Types defined in this module.
    pub struct_defs: Vec<StructDefinition>,
    /// Function defined in this module.
    pub function_defs: Vec<FunctionDefinition>,
}
```

Structural Checks

- BoundsChecker
- LimitsVerifers
- DuplicationChecker
- SignatureChecker
- InstructionCosnsitency
- RecursiveStructDefChecker
- …

Semantic Checks

- StackUsageVerifier
- TypeSafetVerifier
- LocalSafetyVerifier
- ReferenceVerifier
- AcquiresVerifier

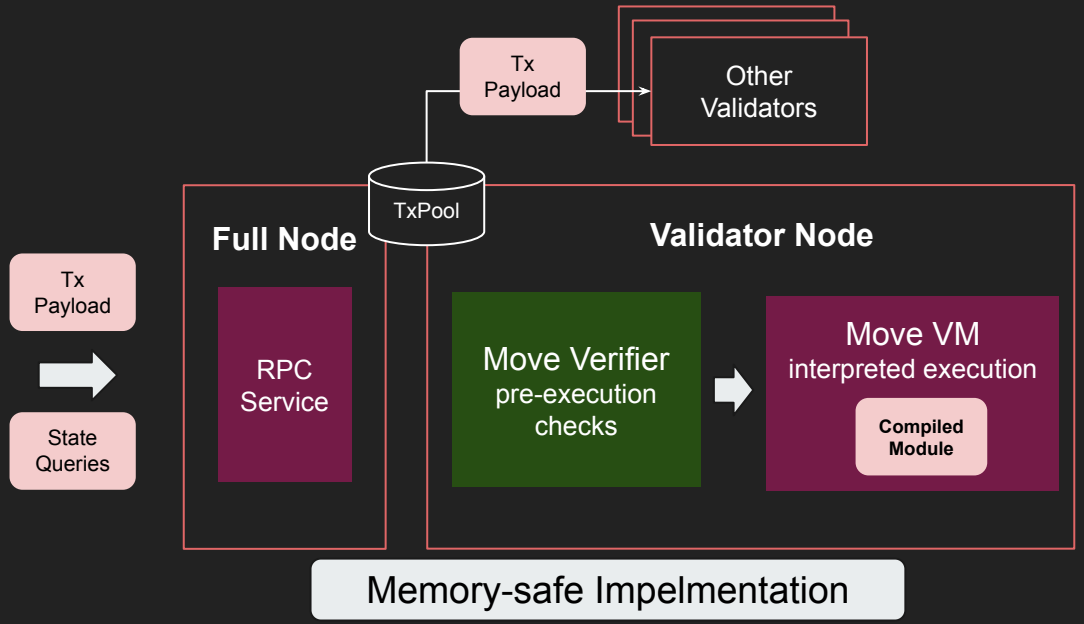
Mandatory verification stage before execution

- 20+ checkers
  - modules, struct_def, function_def, constant
  - signatures

Move Background
Security Enforcement in Move

Threat Modeling of Move-based Blockchains

Attack Surface Analysis: Targets



Full Node RPC Service

- Checks Tx size, signature, nonce, etc.
- Query on-chain states.

DoS Issues (e.g. Resource exhaustion, panic)

Full node outage cuts off the connection between users and the network. The blockchain is still operational.

Validator Node Network

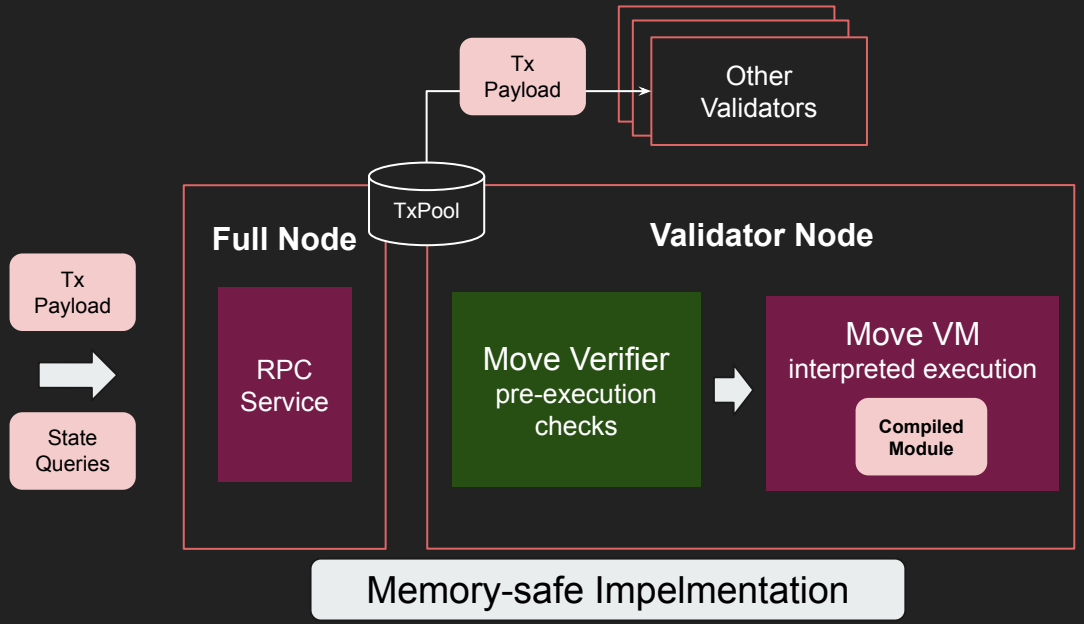
- Move VM + Verifier

DoS Issues (e.g. Resource exhaustion, panic)

Correctness Issues (e.g. Wrong results)

CERTIK

Attack Surface Analysis: Challenges



Challenges

- No more memory-safety Bugs
  - Pure Rust, fobidden_unsafe
- Mandatory verification stage
  - Pass checks before executed by VM
- Charge gas fee during executing
  - Mitigate resource exhaustion

CERTIK

Attack Surface Analysis: Opportunities

No more memory-safety bugs ⇒ Bug patterns unrelated with memory safety

Integer overflow

```
let a = u8::MAX;
let b:u8 = 2;

assert_eq!(1,a+b);   // Panic in Debug
assert_eq!(1,a.add(b));

assert_eq!(None,a.checked_add(b));
assert_eq!(255,a.saturating_add(b));
```

Runtime panics

```
    last_index: CodeOffset,
) -> Result<(), Self::AnalysisError> {
    execute_inner(self, state, bytecode, index)?;
    if index == last_index {
        assert!(self.stack.is_empty());      // <---
        *state = state.construct_canonical_state()
    }
    Ok(())
```

Consequences: Denial of Sevice

CERTIK

Attack Surface Analysis: Opportunities

Mandatory verification stage

⇒ Critical to on-chain security enforcement

⇒ The implementation is complicated (Abstract Interpretation, CFG building, etc.)

Targeted Aspects

- ❖ Correctness
  - Type enforcement failure
- ❖ Robustness
  - Panic
  - Resource exhausion
  - Livelock

Consequences

- ❖ Forging/Stealing Fund (Integrity)
- ❖ Denial of Sevice (Availability)
  - Chain shutdown due to node crashes
  - Chain not responsive to new Txns

The severity of DoS in Web3?

CERTIK

The Realistic Threats of Web3: Network Outage

Aptos Hit With 5-Hour Outage on Blockchain's First Birthday

The speedy layer-1 network is back up and running but the event raised concerns about Aptos' performance.

By Nivesh Rustgi

Oct 19, 2023
2 min read

EZRA REGUERRA FEB 27, 2023

Solana outage triggers ballistic reaction from the crypto community

A community member argued that outages put decentralized finance protocols running on Solana at risk of insolvency.

LUKE HUIGSLOOT FEB 22, 2023

Polygonscan went down, causing unwarranted concern of blockchain outage

Data from PolygonScan showed that the blockchain had not produced any new blocks or processed transactions for some time, leading some to believe it was suffering an outage.

Consequences After Network Outage

- DApp Suspension
- Native Token Price Drop
- Exchange Lockup

Ecosystem Confidence Loss

- DApp Developers + Users
- Token Traders

MARTIN YOUNG JUN 02, 2022

Reliably unreliable: Solana price dives after latest network outage

Solana has suffered its fifth outage of 2022, and the year is only five months old. A bug-related consensus failure was the culprit this time.

CERTIK

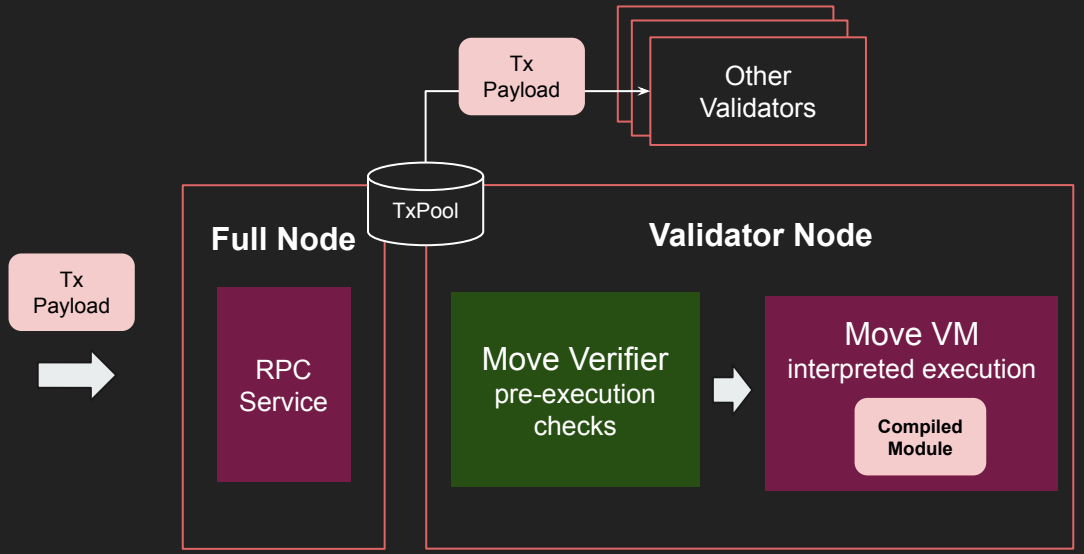
Critical Dos In Web3: Equally Important As Integrity Issues

Blockchain/DLT

- Exceeding the maximum supply of 10 billion SUI + allowing the attacker to claim the excess funds (Critical)
- Loss of Funds (Critical)
- Violating BFT assumptions, acquiring voting power vastly disproportionate to stake, or any other issue that can meaningfully compromise the integrity of the blockchain's proof of stake governance (Critical)
- Network not being able to confirm new transactions (total network shutdown) requiring a hard fork to resolve (Critical)
- Arbitrary, non-Move remote code execution on unmodified validator software (Critical)

https://hackenproof.com/sui/sui-protocol

Critical Dos In Web3: Equally Important As Integrity Issues



Critical DoS in Web3

- Stall tx processing
  - Multiple nodes Validator network.
- Hardfork to resolve
  - Unreconverable by restarting.

The double-edged sword feature

(in the decentralized world)

- Automatic transaction propagation

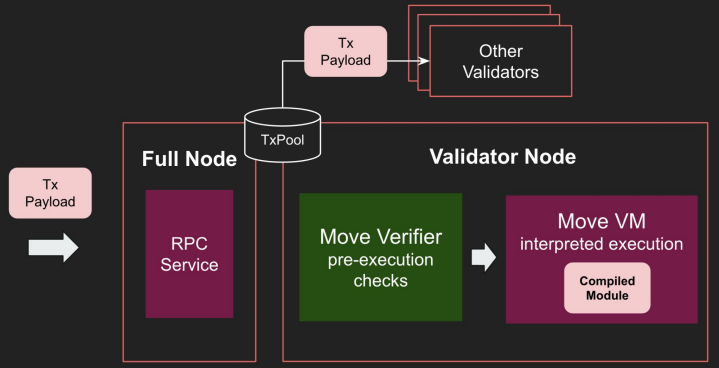
Bug Finding Objectives

Correctness Issues

- Breaking the on-chain TypeSafety enforncement
  - Missing checks in Move Verifier
  - Defect check implementation in Move Verifier

Robustness Issues

- Unrecoverable exceptions
  - Runtime panics
  - Integer overflow, etc.
- Resource exhaustion in Move Verifier or Move VM
  - Deadloops
  - Memory explosion



- Integrity
- Availability

Move Background
Security Enforcement in Move
Threat Modeling of Move-based Blockchains

Hunting For The Bugs

CERTIK

Approach 1: Manually Play with the CompiledModule

Manually Introducing Inconsistency

- Out-of-bound offset
- Mismatched reference index
- Recursive reference tokens
- ….

Verifier checkers catch almost all the malformed behaviors.

```
pub struct CompiledModule {
    /// Version number found during deserialization
    pub version: u32,
    ...
    /// Handles to external dependency modules and self.
    pub module_handles: Vec<ModuleHandle>,
    /// Handles to external and internal types.
    pub struct_handles: Vec<StructHandle>,
    /// Handles to external and internal functions.
    pub function_handles: Vec<FunctionHandle>,
    ...
    /// Locals signature pool. The signature for all locals of the
functions defined in the module.
    pub signatures: SignaturePool,
    /// All identifiers used in this module.
    pub identifiers: IdentifierPool,
    /// All address identifiers used in this module.
    pub address_identifiers: AddressIdentifierPool,
    /// Constant pool. The constant values used in the module.
    pub constant_pool: ConstantPool,
    ...
    /// Types defined in this module.
    pub struct_defs: Vec<StructDefinition>,
    /// Function defined in this module.
    pub function_defs: Vec<FunctionDefinition>,
}
```

CERTIK

Approach 1: Manually Manipulate CompiledModule

Checker sequence matters

- **Bounds Checking**: Ensures each referenced offset is in-bound before access. Mitigates out-of-bounds vulnerabilities.
- **Limit Checking**: Validates number of entries in each table. Prevents overflows.
- **Duplication Checking**: Checks for duplicate entries. Avoids ambiguities.
- **Signature Checking**: Verifies struct/function definitions match declarations. Prevents type confusion.
- …

```
pub fn verify_module_with_config(
    config: &VerifierConfig,
    module: &CompiledModule,
) -> VMResult<()> {
    BoundsChecker::verify_module(module).map_err(|e| {
        e.finish(Location::Undefined)
    })?;
    LimitsVerifier::verify_module(config, module)?;
    DuplicationChecker::verify_module(module)?;
    SignatureChecker::verify_module(module)?;
    InstructionConsistency::verify_module(module)?;
    constants::verify_module(module)?;
    friends::verify_module(module)?;
    ability_field_requirements::verify_module(module)?;
    RecursiveStructDefChecker::verify_module(module)?;
    InstantiationLoopChecker::verify_module(module)?;
    CodeUnitVerifier::verify_module(config, module)?;
    ...
}
```

Crafting edge cases around one checker may be mitigated by prior checkers in the sequence.

CERTIK

Checker Sequence Matters: SignatureChecker

```rust
/// Checks if the given type is well defined in the given context.
/// References are only permitted at the top level.
fn check_signature(&self, idx: SignatureIndex) -> PartialVMResult<()> {
    for token in &self.resolver.signature_at(idx).0 {
        match token {
            SignatureToken::Reference(inner) |
            SignatureToken::MutableReference(inner) => {
                self.check_signature_token(inner)?
            }
            _ => self.check_signature_token(token)?,
        }
    }
    Ok(())
}
```

SignatureChecker: Recursive Call?

```rust
/// Checks if the given type is well defined in the given context.
/// No references are permitted.
fn check_signature_token(&self, ty: &SignatureToken) -> PartialVMResult<()> {
    use SignatureToken::*;
    match ty {
        U8 | U16 | U32 | U64 | U128 | U256 | Bool | Address | Signer | Struct(_)
        | TypeParameter(_) => Ok(()),
        Reference(_) | MutableReference(_) => {
            // TODO: Prop tests expect us to NOT check the inner types.
            // Revisit this once we rework prop tests.
            Err(PartialVMError::new(StatusCode::INVALID_SIGNATURE_TOKEN)
                .with_message("reference not allowed".to_string()))
        }
        Vector(ty) => self.check_signature_token(ty),
        StructInstantiation(_, type_arguments) => self.check_signature_tokens(type_arguments),
    }
}
```

```rust
fn verify_type_node(
    &self,
    config: &VerifierConfig,
    ty: &SignatureToken,
) -> PartialVMResult<()> {
    if let Some(max) = &config.max_type_nodes {
        // Structs and Parameters can expand to an unknown number of nodes, therefore
        // we give them a higher size weight here.
        const STRUCT_SIZE_WEIGHT: usize = 4;
        const PARAM_SIZE_WEIGHT: usize = 4;
        let mut size = 0;
        for t in ty.preorder_traversal() {
            // Notice that the preorder traversal will iterate all type instantiations, so we
            // why we can ignore them below.
            match t {
                SignatureToken::Struct(..) | SignatureToken::StructInstantiation(..) => {
                    size += STRUCT_SIZE_WEIGHT
                }
                SignatureToken::TypeParameter(..) => size += PARAM_SIZE_WEIGHT,
                _ => size += 1,
            }
        }
        if size > *max {
            return Err(PartialVMError::new(StatusCode::TOO_MANY_TYPE_NODES));
        }
    }
    Ok(())
}
```

Mitigated by LimitsChecker

CERTIK

Approach 2: Fuzzing the through CompiledModule

Granularity Tradeoff About CompileModule Mutation

- Entire structure: less efficient, more inconsistency
- Subfields: more focusing, less inconsistency

Limited Exception Signals

- Runtime exception
- Memory corruption
- Cannot catch if there is a checker bypass

```rust
#![no_main]
use libfuzzer_sys::fuzz_target;
use move_binary_format::file_format::CompiledModule;

fuzz_target!(|module: CompiledModule| {
    let _ = move_bytecode_verifier::verify_module(&module);
});
```

```rust
fuzz_target!(|code_unit: CodeUnit| {
    let mut module = from_template();

    let fun_def = FunctionDefinition {
        code: Some(code_unit),
        function: FunctionHandleIndex(0),
        visibility: Visibility::Public,
        is_entry: false,
        acquires_global_resources: vec![],
    };

    module.function_defs.push(fun_def);
    let _ = move_bytecode_verifier::verify_module(&module);
});
```

CERTIK

Approach 3: Review The Code Semantic

Manual review. Dive into each checker.

- Trade off between depth and breadth first exploration
- Avoid getting lost by reviewing with questions
  - What properties is it enforcing? Invariants? Edge cases?
  - Could I implement this code easily?
  - If not, what could possibly be wrong?
  - Any semantic inconsistencies?

Findings

Type 1: Triggering Panic in Validator

CERTIK

#1: Unhandled Panic During Module Normalization

Move VM is implemented in Rust

- Memory Safe but **not Panic Free**
- **Explicit panics**
  - assert!(), panic!(),
  - Err::unwrap()
  - unreachable!()
- **Implicit panics**
  - Hashmap::index
    e.g. map[“key”]

Fix: Introduce Panic Handling

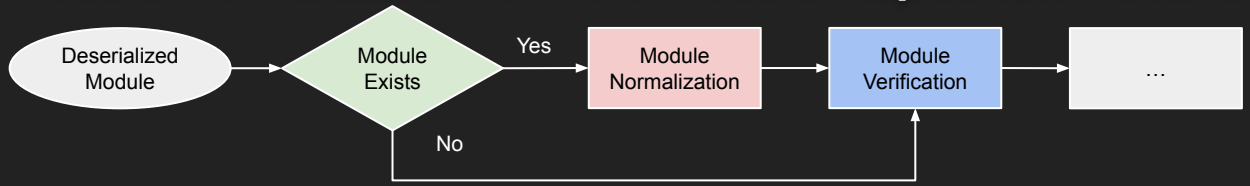
- std::panic::catch_unwind
- use Result to propagate Error

```
// Module Normalization
// normalized::Module::new()
impl Struct {
    /// Create a `Struct` for `StructDefinition` `def` in module `m`.
    /// Panics if `def` is a native struct definition.
    pub fn new(m: &CompiledModule, def: &StructDefinition) -> (Identifier, Self) {
        let handle = m.struct_handle_at(def.struct_handle);
        let fields = match &def.field_information {
            StructFieldInformation::Native =>
                    panic!("Can't extract for native struct"),
        ...
}
```

Module Normalization Panic

Challenges To Discover This Issue

Module Normalization is in the Module **Republish** Routine



Depending on prior states.

Bypass verifier:

- untrusted module is accessed before being verified.

```
for module in &compiled_modules {
    let module_id = module.self_id();
    if data_store.exists_module(&module_id)? {
        let old_module_ref = self.loader.load_module(&module_id, data_store)?;
        let old_module = old_module_ref.module();
        let old_m = normalized::Module::new(old_module);
        let new_m = normalized::Module::new(module);
        let compat = Compatibility::check(&old_m, &new_m);
        if !compat.is_fully_compatible() {
            return Err(...);
        }
    }
    if !bundle_unverified.insert(module_id) {
        return Err(...);
    }
}

self.loader
    .verify_module_bundle_for_publication(&compiled_modules, data_store)?;
```

Findings

Type 2: Bypass Type Safety Enforcement

CERTIK

Type Safety Checker's Enforcement

- The verifier simulates executing each instruction, tracking abstract types
- It checks operand types match expected and operation results are the expected types

```
Bytecode::StLoc(idx) => {
    let operand = safe_unwrap_err!(verifier.stack.pop());
    if &operand != verifier.local_at(*idx) {
        return Err(verifier.error(StatusCode::STLOC_TYPE_MISMATCH_ERROR, offset));
    }
}
```

- Examples
  - Bytecode::Add consumes two integer operands
    - Must be same type (u8, u128, etc.)
    - Pushes result integer back onto stack
  - Bytecode::MoveLoc(idx)
    - Moves local at idx with type T to stack, create a type T on stack
  - Bytecode::StLoc(idx) pops stack into local slot idx
    - Checks stack type matches local slot's sigature

```
let a: bool = true; // local slot 0
let b: u8 = 0;      // local slot 1
MoveLoc(1)       // push b on stack
MoveLoc(0)       // push a on stack
StoreLoc(1)      // pop stack to b
```

#2: Failed to Catch Mismatched Type Pack/UnPack

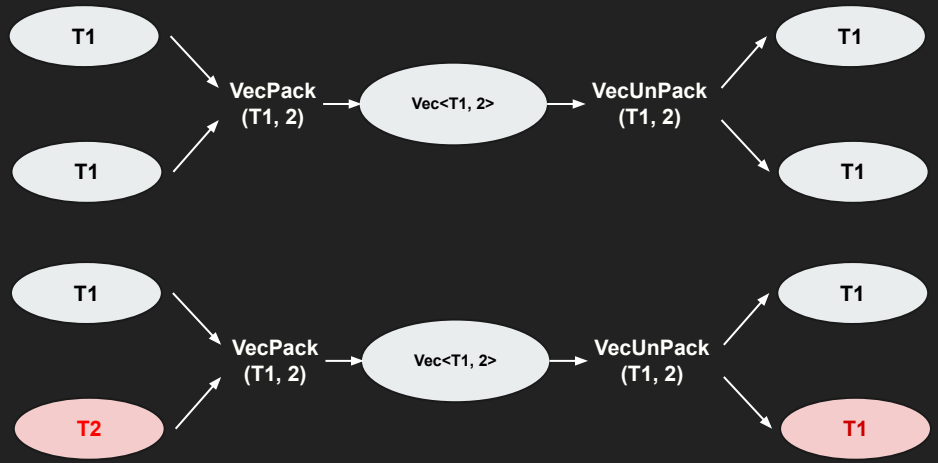
Type checking missing at instruction level

- VecPack OP doesn't check the type of elements

Consequence

- Resource fabrication

  e.g. Coin Type T2 changed to Coin Type T1



CERTIK

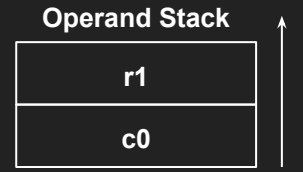
Bypass Type Safety: Forge Assets

- Manipulate at bytecode level
- Attack Primitive
  - Convert to arbitrary type

```
Bytecode::VecPack(idx, num) => {
    let element_type = &verifier.resolver.signature_at(*idx).0[0];
    for _ in 0..*num {
-       verifier.stack.pop().unwrap();
+       let operand_type = verifier.stack.pop().unwrap();
+       if element_type != &operand_type {
+           return Err(verifier.error(StatusCode::TYPE_MISMATCH, offset));
+       }
    }
```

https://github.com/move-language/move/pull/491

```
module example::poc_vecpack {
    struct Coin<phantom T> has key, drop {
        value: u64
    }
    struct Usd {}
    struct Rmb {}

    public entry fun main() {
        let c0 = Coin<Usd>{value: 0};
        let r1 = Coin<Rmb>{value: 1024};

        vec_pack(Coin<Usd>, 1);       // bytecode: pack(r1)
        vec_unpack(Coin<Usd>, 1);     // bytecode: unpack() to stack
        stloc(0);                     // bytecode: store to c0

        let Coin<Usd>{ value } = c0;
        assert!(value == 1024, 0x02); // assertion success
    }
```



Finding

Type 3: Verifier Robustness

CERTIK

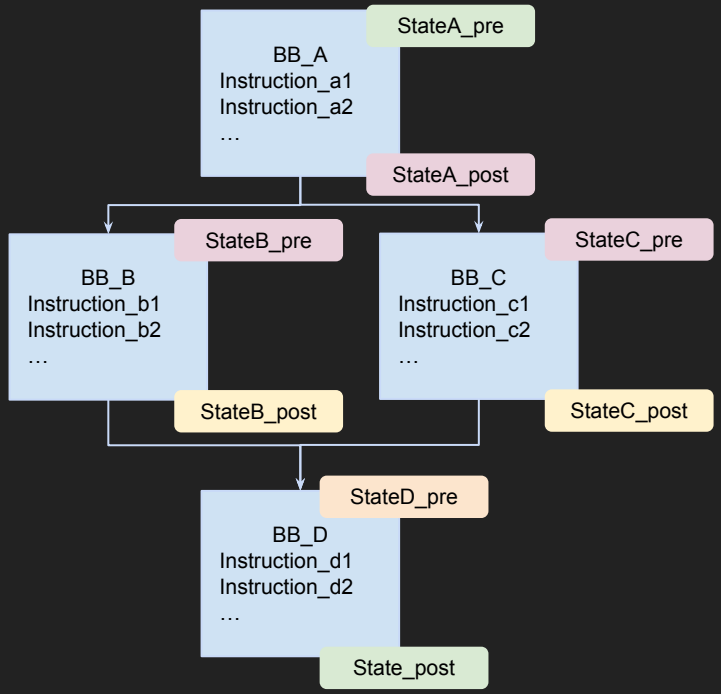
Abstract Interpreter

Automatically compute relevant semantic information about a program by **interpreting** ("executing") it over an **abstract domain** (“states”) instead of concrete values.

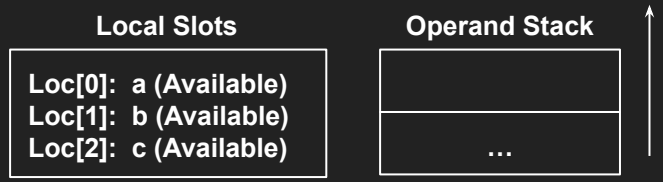
Analysis Plugin
- Defines what to maintain as state in a BB
- Defines how to change states during intepretation
- Defines how to join two BB states

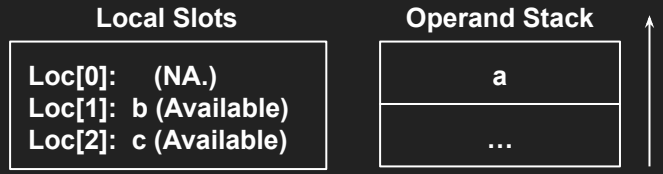
Other Plugins …

Abstract Interpreter Engine
- Maintain pre and post states for each block
- Interpret each instruction within block
- Join current block’s post state with all successor BB’s pre-state
- Keep iterating until no state changes



Abstract Intepreter - Locals Safety Analysis





MovLoc(1) ⇒ Error

```rust
pub(crate) enum LocalState {
    /// The local does not have a value
    Unavailable,
    /// The local was assigned a non-drop value in at least one control flow path,
    // but was `Unavailable` in at least one other path
    MaybeAvailable,
    /// The local has a value
    Available,
}
```

Define State

```rust
Bytecode::MoveLoc(idx) => match state.local_state(*idx) {
    LocalState::MaybeAvailable | LocalState::Unavailable => {
        return Err(state.error(StatusCode::MOVELOC_UNAVAILABLE_ERROR, offset))
    }
    LocalState::Available => state.set_unavailable(*idx),
},
```

Update/Check State During Interpretation

```rust
match (self_state, other_state) {
    // Unavailable on both sides, nothing to add
    (Unavailable, Unavailable) => Unavailable,

    (MaybeAvailable, Unavailable)
    | (Unavailable, MaybeAvailable)
    | (MaybeAvailable, MaybeAvailable)
    | (MaybeAvailable, Available)
    | (Available, MaybeAvailable)
    | (Unavailable, Available)
    | (Available, Unavailable) => MaybeAvailable,

    (Available, Available) => Available,
}
```

Join States

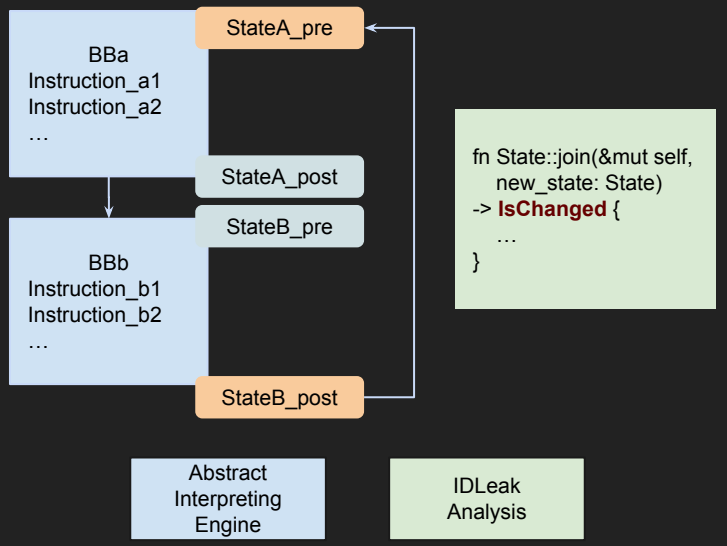
#3: Failed to Handle Infinite Analysis Loops

If there is a backedge in CFG, reanalyze

- until no changes to the BB's joint states

Issue: Inconsistent state joining logic causes infinite abstract interpreting analysis

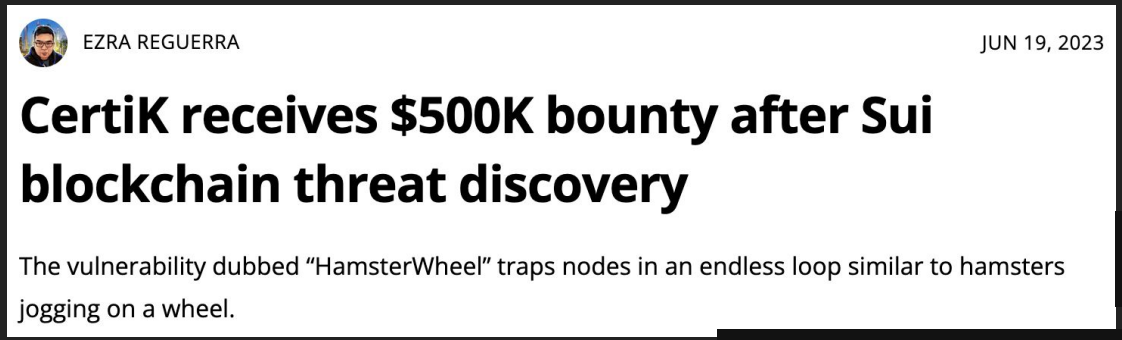
```
// sui-verifier/src/id_leak_verifier.rs
fn join(&mut self, state: &AbstractState,)
-> Result<JoinResult, PartialVMError> {
   let mut changed = false;
   for (local, value) in &state.locals {
      let old_value = *self.locals.get(local);
-     changed |= *value != old_value;
-     self.locals.insert(*local, value.join(&old_value));
+     let new_value = value.join(&old_value);
+     changed |= new_value != old_value;
+     self.locals.insert(*local, new_value);
```



changed flag vs. value update

CERTIK

Sui Bug Bounty - Critical With Maximum Payout



EZRA REGUERRA JUN 19, 2023

CertiK receives $500K bounty after Sui blockchain threat discovery

The vulnerability dubbed “HamsterWheel” traps nodes in an endless loop similar to hamsters jogging on a wheel.

Level	Payout	
Critical	USD $100,000 - $500,000	PoC Required

Impact	
Network not being able to confirm new transactions (total network shutdown) requiring a hard fork to resolve	Critical
Temporary total network shutdown or unintended chain split (duration greater than 10 minutes)	High
Shutdown of greater than or equal to 30% of network processing nodes without brute force actions, but does not shut down the network	Medium

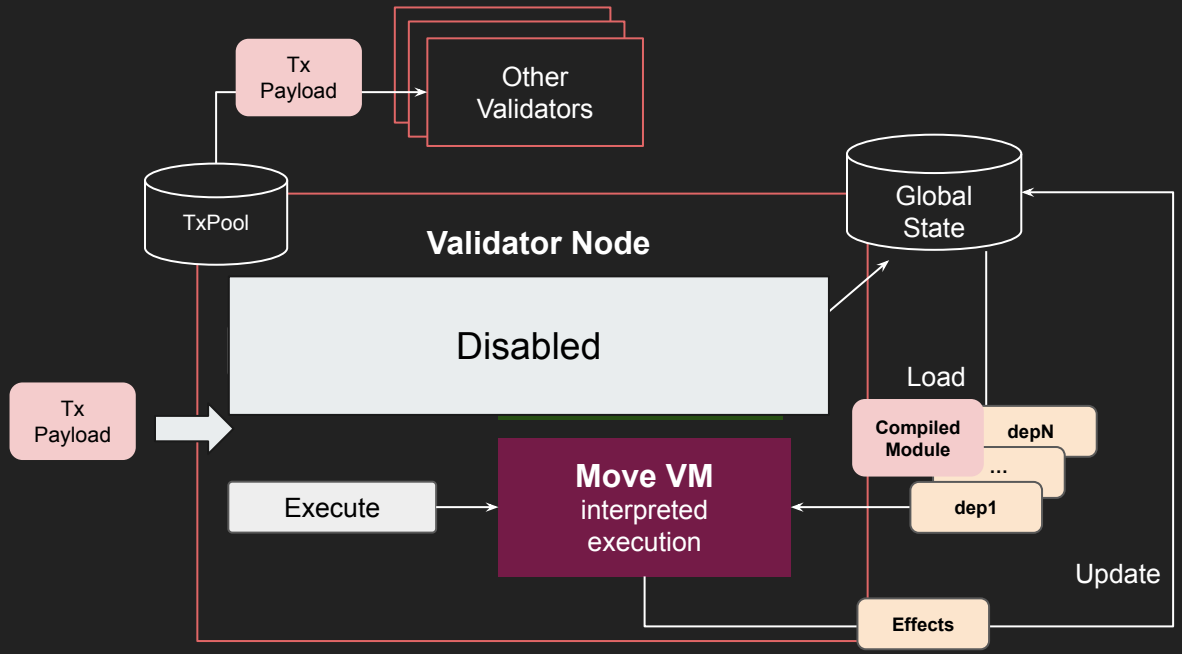
Mitigation I: Metered Analysis

Introduce meters to interpreter engine and all plugins
Limit the cost for verifier analysis (similar as gas fee)

```rust
pub trait AbstractInterpreter: TransferFunctions {
    /// Analyze procedure local@function_view starting from pre-state
    fn analyze_function(
        &mut self,
        initial_state: Self::State,
        function_view: &FunctionView,
        meter: &mut impl Meter,
    ) -> PartialVMResult<()> {
        meter.add(Scope::Function, ANALYZE_FUNCTION_BASE_COST)?;
        let mut inv_map = InvariantMap::new();
```

```rust
impl SuiVerifierMeterBounds {
    fn add(&mut self, ticks: u128) -> PartialVMResult<()> {
        let max_ticks = self.max_ticks.unwrap_or(u128::MAX);

        let new_ticks = self.ticks.saturating_add(ticks);
        if new_ticks >= max_ticks {
            return Err(PartialVMError::new(StatusCode::PROGRAM_TOO_COMPLEX)
                    .with_message(format!(
                        "program too complex. Ticks exceeded `{}` will exceed
                        self.name, self.ticks, ticks, max_ticks
                    )));
        }
        self.ticks = new_ticks;
        Ok(())
    }
}
```

Mitigation II: Sacrificing The Module Publish Functionality



Critical DoS in Web3

- **Stall tx processing**
  - Multiple nodes Validator network.
- Hardfork to resolve
  - Unreconverable by restarting.

Introduce config to launch verifier with publish routine disabled.

- Only allow execution of existing modules.
- **Network is still process tx, partially.**

CERTIK

The rescue workflow - Temporarily Down To Recover

If Move Verifier is under attack

- Reject new attack payload
  - Disable module publish
- Reject malformed payload published
  - Blacklist
    - Tx Signer
    - Tx ID
    - Bad dependency

Critical DoS -> Temporarily DoS (High)

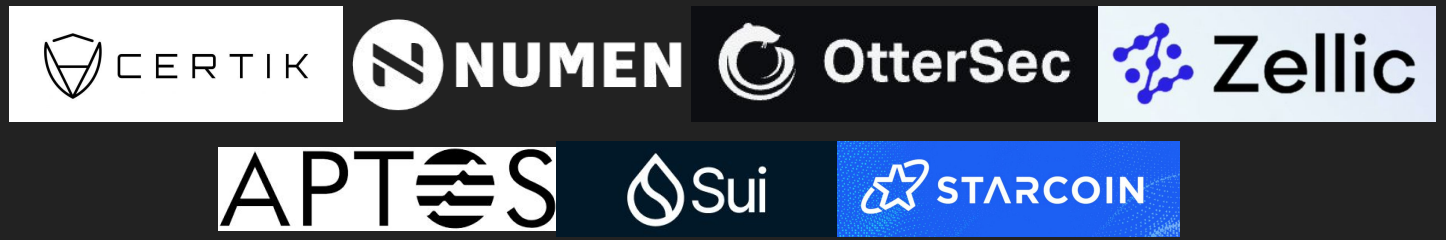
- **Stall tx processing:** Multiple nodes Validator network.
- **Hardfork to resolve:** Unreconverable by restarting.

```
for command in tx_data.kind().iter_commands() {
    deny_if_true!(
        filter_config.package_publish_disabled() && matches!(command, Command::Publish(..)),
        "Package publish is temporarily disabled"
    );
    deny_if_true!(
        filter_config.package_upgrade_disabled() && matches!(command, Command::Upgrade(..)),
        "Package upgrade is temporarily disabled"
    );
}
fn check_signers(filter_config: &TransactionDenyConfig, tx_data: &Transacti
    let deny_map = filter_config.get_address_deny_set();
    if deny_map.is_empty() {
        return Ok(());
    }
    for signer in tx_data.signers() {
        deny_if_true!(
            deny_map.contains(&signer),
            format!(
                "Access to account address {:?} is temporarily disabled",
                signer
            )
        );
    }
```

Other Security Improvement Related to Move Implementations

- Zellic : Vulnerability in CFG construction to Bypass Move Verifiers
- OtterSec: Defense-in-depth hardening to the Move VM Runtime
- Numen: Integer Overflow in Move Verifiers

- and many other findings by community developers and builders …



Summary

- MOVE-Lang introduces new security features for smart contract dev.
- These feature’s success relies on the correct implementations of verifiers.
- CertiK, along with other Web3 security firms and Move community, have continuously improved the security of Move implementations.
- Despite these identified findings, we strongly recommend the adoption of such security features (type safety and formal verification) in blockchain development.

CERTIK

Demo Video

About CertiK

Founded in 2018 by professors of **Columbia** and **Yale**, CertiK is a pioneer in blockchain security, utilizing best-in-class **Formal Verification** and **AI technology** to secure and monitor blockchains, smart contracts, and Web3 apps. CertiK completed **3,300+** audits, secured **$287 billion** of assets.

Our Investors

INSIGHT PARTNERS · SEQUOIA · TIGERGLOBAL · COATUE · Lightspeed · Advent International · Goldman Sachs · coinbase · BINANCE · Yale University