A Single Transaction to Rule Them All: Attacking Blockchain Validators

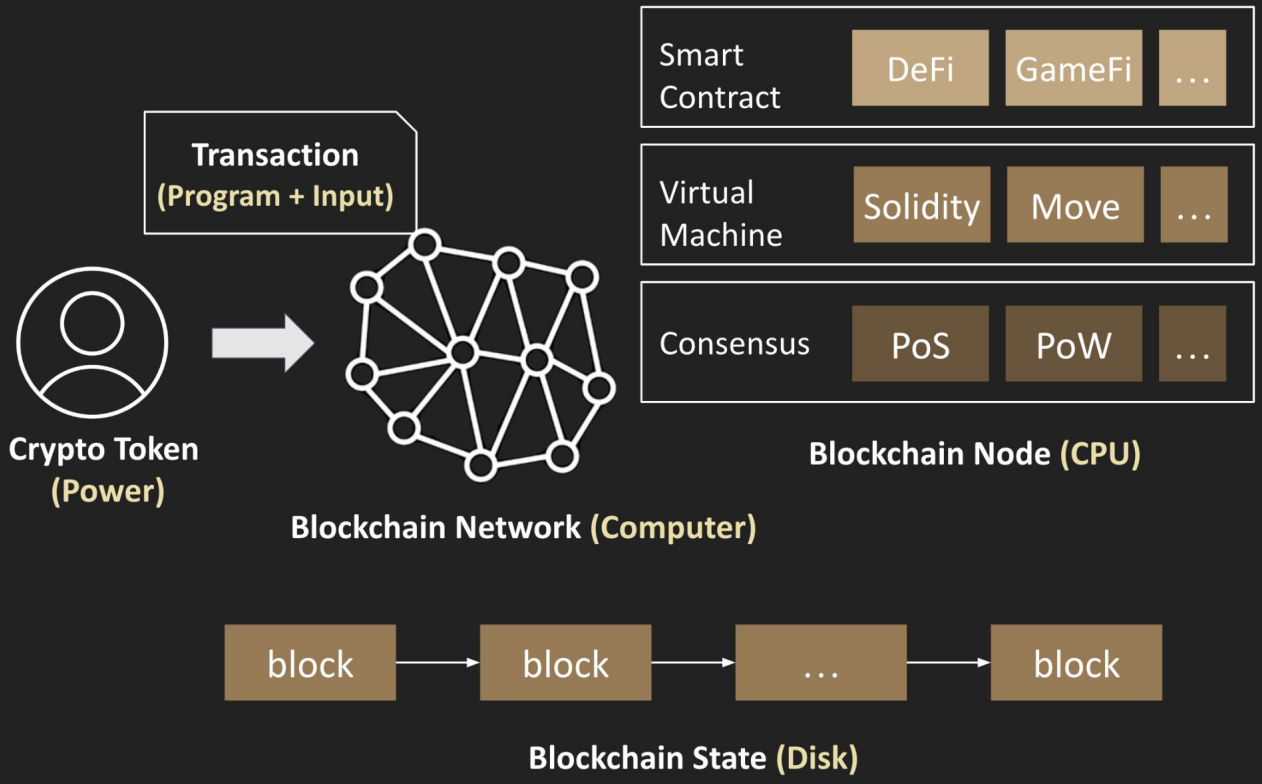
Zhaofeng Chen, Yuxiao Wang
CertiK Skyfall

Background

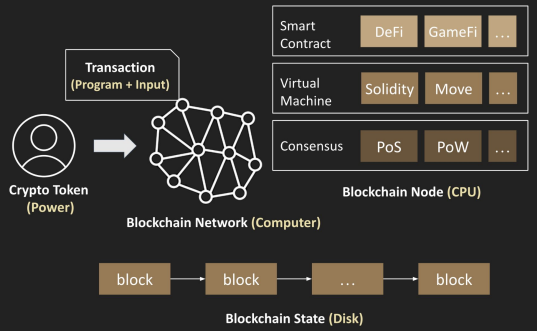
- Research Interest: Web3 Infrastructure Security
  - Nodes, virtual machines (VMs) across diverse blockchain ecosystems.
- POC 2023
  - Attack Move Verifiers: Our Experiences of Exploiting and Enhancing Move-based Blockchain
    - Insights into implementation flaws in on-chain security component
- Today’s topic: The Overlooked Risk of Resource Exhaustion
  - Exploring on-chain resource usage across multiple dimensions.
  - How to disrupt blockchain networks by exhausting resources.

CERTIK

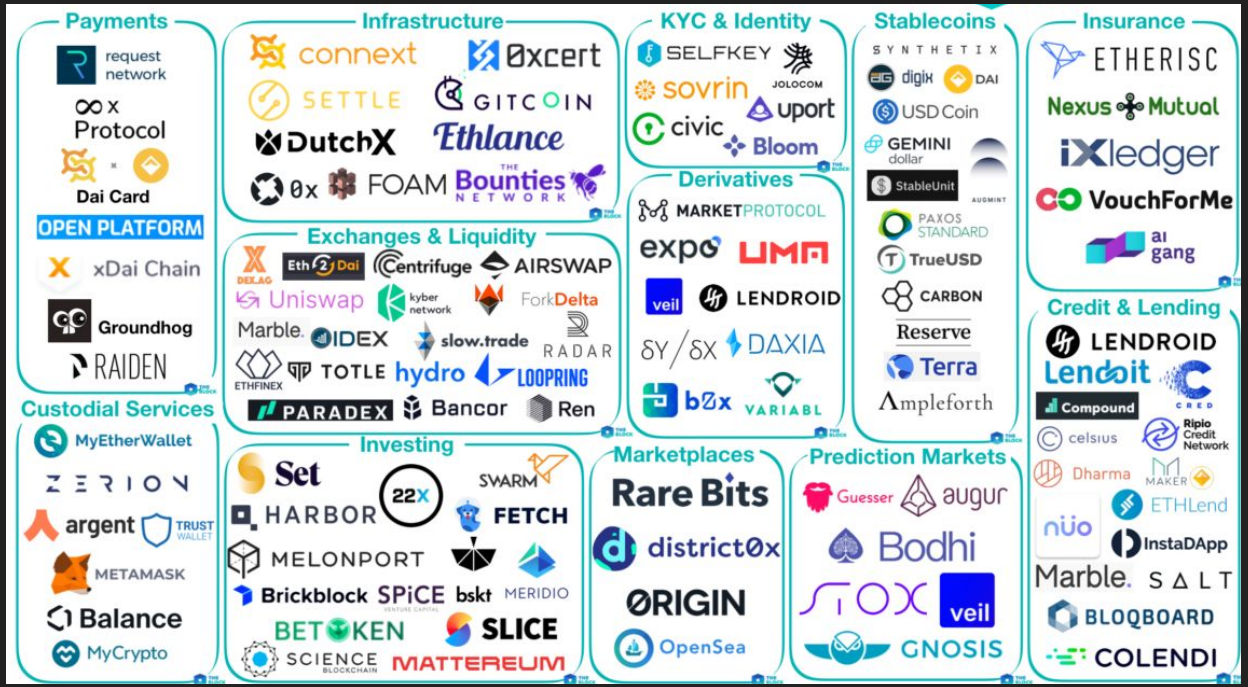
Blockchain Infrastructure as the World Machine



Empower The Decentralized Application Ecosystem



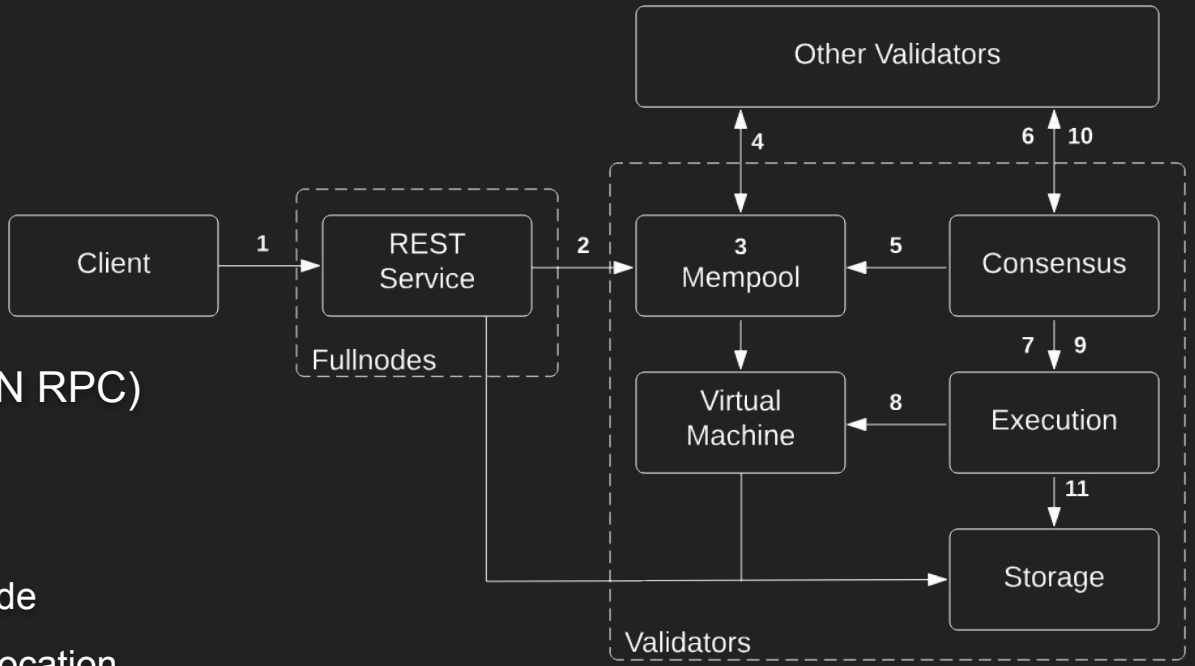
- Network Resilience (Liveness)
- Tamper-proof (State integrity)
- Deterministic Execution
- Permissionless Access



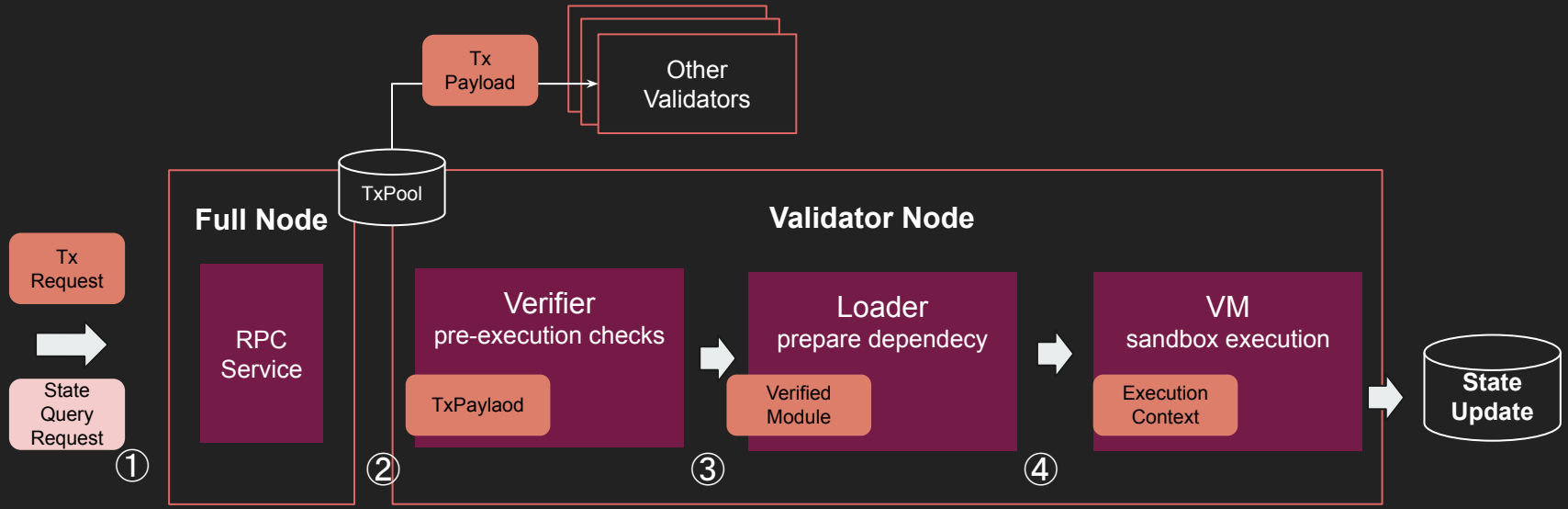
DApp ecosystem powered by smart contracts

Open Interaction With The World Machine

- Entrypoint:
  - FullNode RPC
  - Validator Node
- Payload:
  - HTTP Request (JSON RPC)
    - State Query
    - Transactions
      - Contract Code
      - Contract Invocation



Threat Modeling: Attacker Controlled Data



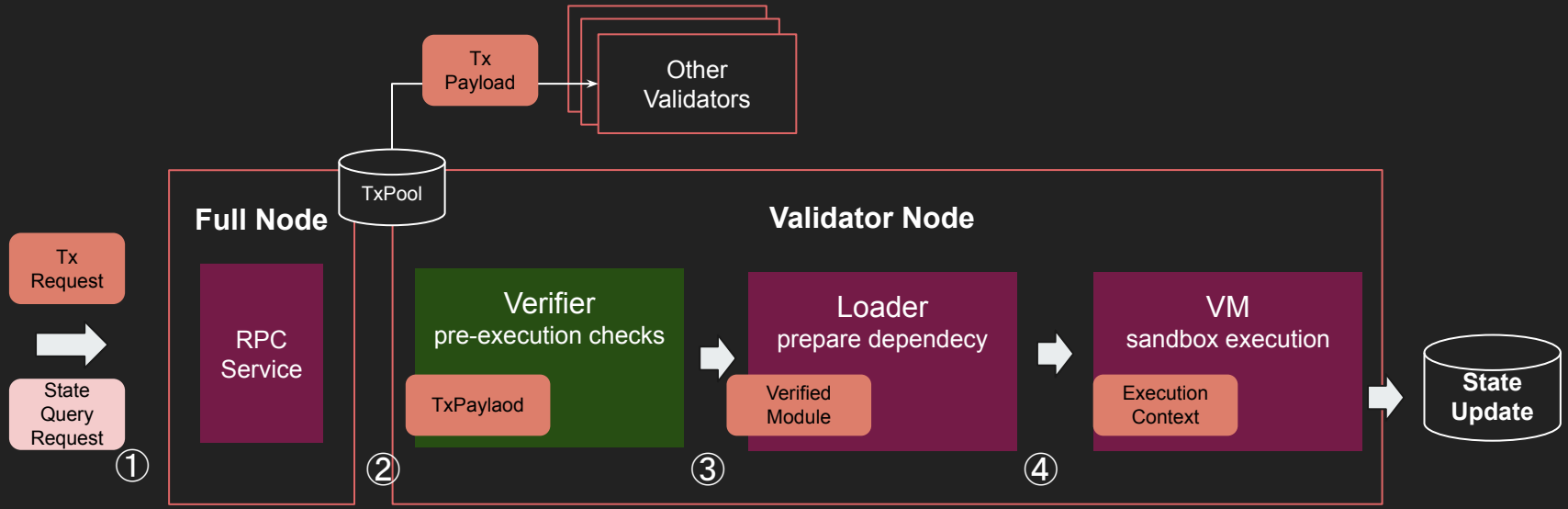
RPC Requests
- submitTransaction
- getObjectByAddress

TxPayload
- PublishModule
- FunctionInvocation
  - Target Function
  - Args

VerifiedModule
- offset in bound
- no duplicated definition
- no type safety violation
- …

ExecContext
- loaded function
- ModuleCache
  - dependent types
  - func, struct, …

Threats Covered in POC 2023 Talk



Implementation Flaw in Move Verifiers

- Logic flaws in security primitives can bypass safety
  - e.g., type safety bypass.
- Rust's Primitives and Robustness Issues
  - e.g. Integer Overflow, panics.

POC 2023 Talk: Hamsterwheel Attack

https://www.certik.com/resources/blog/3TuPZ1LnPTwkzXZQcEJ3aR-the-hamsterwheel-an-in-depth-exploration-of-a-novel-attack-vector-on-the-sui

```
// 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);
  }
  return changed;
```

Critical — Level

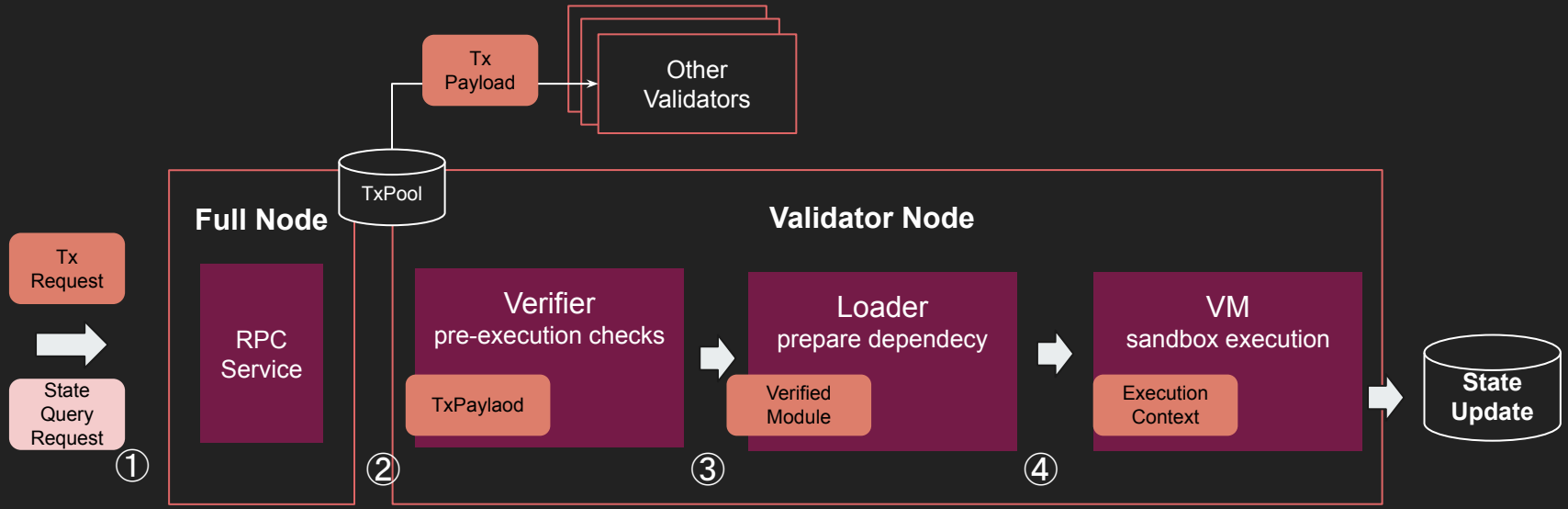
USD $100,000 - $500,000 — Payout

PoC Required

Network not being able to confirm new transactions (total network shutdown) requiring a hard fork to resolve — Impact

Critical

This Talk: Resource Exhaustion Threats



- Network (Full Node) : RPC Framework
- CPU & Memory (Full Node + Validator Node)
  - Triggered by single request (< 1MB)
  - Target different phases to consume excessive resources

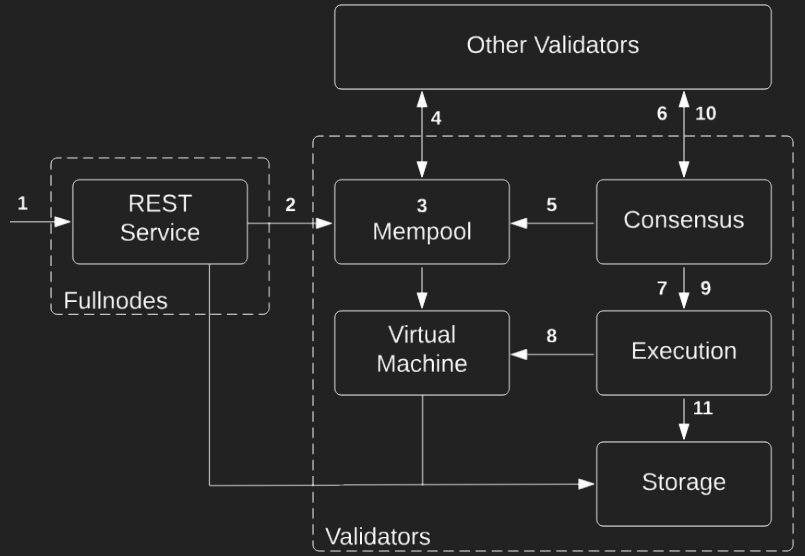
Why Resource Exhaustion Attacks Matter

Attack Vectors

- Transactions: arbitrary programs/invocations
- Malformed RPC Request

Consequences

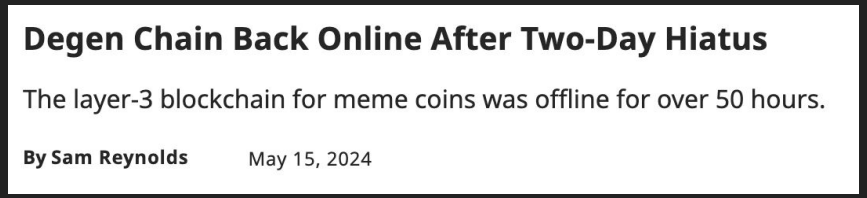
- Full Node Exhaustion
  - Prevents nodes from syncing and propagating transactions
  - No new transactions can be processed
- Validator Network Exhaustion:
  - Disables block validation, threatening consensus and blockchain security.
  - Network total shutdown



Demo

The Resource Exhaustion Attack

Consequences of Blockchain Network Outage



Degen Chain Back Online After Two-Day Hiatus

The layer-3 blockchain for meme coins was offline for over 50 hours.

By Sam Reynolds May 15, 2024

https://www.coindesk.com/business/2024/05/15/degen-chain-back-online-after-two-day-hiatus/

Telegram-linked TON blockchain is back online after seven hour outage

By **Callan Quinn**

28 August 2024 at 17:45

• An abnormal transaction load knocked o

https://www.dlnews.com/articles/defi/telegram-linked-ton-blockchain-is-back-online-after-outage/

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.

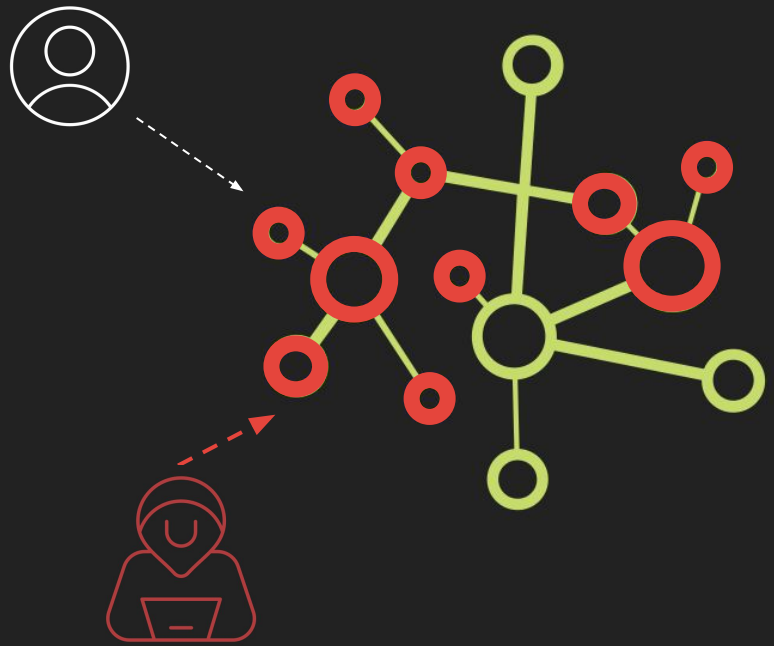
https://cointelegraph.com/news/reliably-unreliable-solana-price-dives-after-latest-network-outage

Consequences After Network Outage

- Cannot confirm new blocks
- DApp suspension
- Native token price drop

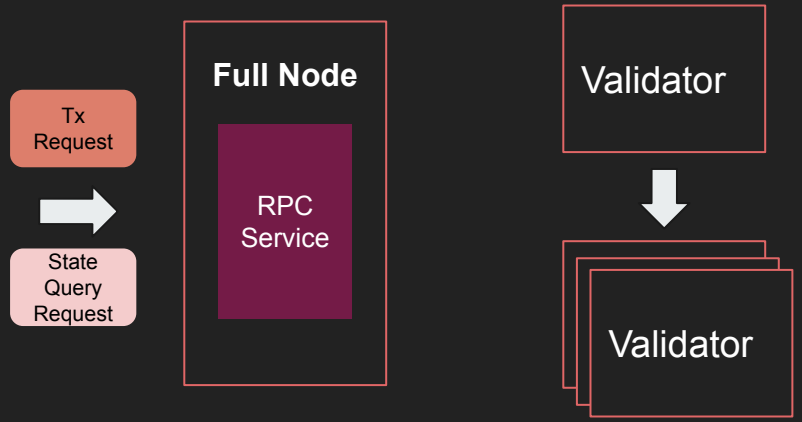
Resource Exhausion: Network

- Network
  - Flooding nodes with requests to cause bandwidth/memory issues.
- CPU
  - Overloading validator nodes with expensive computations.
- Memory
  - Consuming memory resources with malformed complex transaction.



Network Resource Exhaustion - Threat Overview

- Target: Rust Network Framework
  - Fullnodes, Validator
- Threats
  - Traditional DDoS (not our focus)
    - Lower level: bandwidth, IP traffic
  - Application layer
    - HTTP JSON RPC
    - Rust Web Frameworks




```rust
// build our application with a route
let app = Router::new()
    // `GET /` goes to `root`
    .route("/", get(root))
    // `POST /users` goes to `create_user`
    .route("/users", post(create_user));

// basic handler that responds with a static string
async fn root() -> &'static str {
    "Hello, World!"
}
```

Rust Web Framework: Past Incidents

Rust Web Framework: efficiently handling for

- protocol parsing, raw payload deserialization, async handling, and response serialization.

```
POST /ENDPOINT HTTP/1.1
Host: ...
Content-Type: application/json-rpc
Content-Length: ...

{
 "jsonrpc": "2.0",
 "id": 1,
 "method": "getObject",
 "params": {"object_id": 0xaabbccdd},
}
```

axum-core missing request size limit DoS HIGH

CVE-2022-3212

conduit-hyper missing request size limit DoS HIGH

CVE-2022-39294

https://research.jfrog.com/vulnerabilities/

Content-Length: 999999999

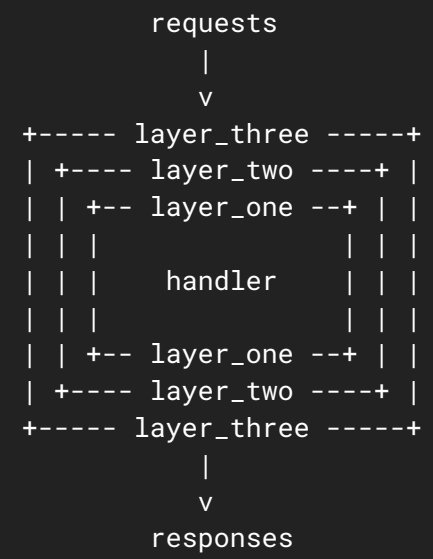
CERTIK

Rust Web Framework: Configurable Middleware

Middleware Design

- Connection timeout
- Maximum connections
- Request body size

```rust
let app = Router::new()
    .layer(TimeoutLayer::new(Duration::from_secs(10)))
    .layer(RequestBodyLimitLayer::new(4096))
    .route("/", post(|request: Request| async{}))
```



Middleware is generally **optional** and **disabled by default**,
leading to potential misconfigurations.

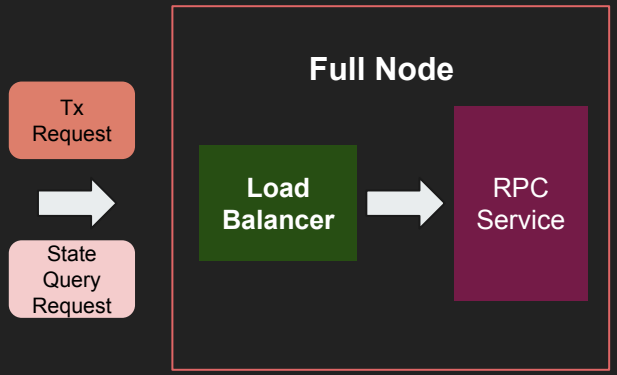
Middlewares Are Not Configured In The Codebase

Blockchain	RPC Framework	Web Framework	Timeout MiddleWare	#Connections MiddleWare	ReqBody Size MiddleWare
Aptos	poem-openapi	poem	Not Configured	Not Configured	Not Configured
Sui	jsonrpsee	axum	Not Configured	Not Configured	Not Configured
Solana	jsonrpc	jsonrpc-http-server	Not Configured	Not Configured	Not Configured
Near	jsonrpc	actix	Not Configured	Not Configured	Not Configured

Given fullnode container with 32GB of memory, **if no middleware is configured**, an attacker could exhaust the entire memory by using 3000 connections.

Attack Limitations

- No propagation
  - The attack is isolated to the targeted node, with no risk of spreading to other nodes.
- No Amplification
  - Attackers cannot amplify traffic to increase impact, making these attacks costly & less scalable.
- **Load Balancers/Proxies** Configurations
  - Deployment environments often enable load balancers or proxies to enforce restrictions.



Defense At Deployment Phase

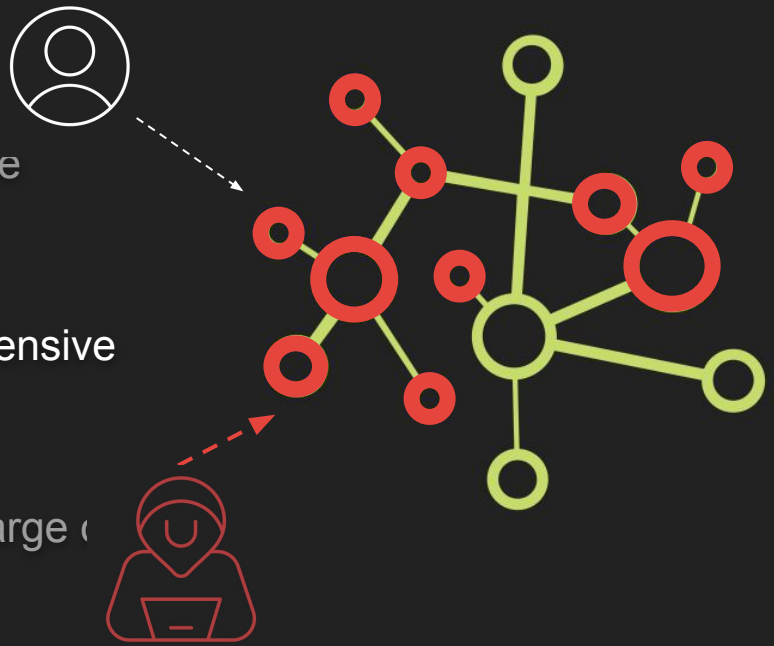
Blockchain Deployment Configures & Restrictions

Blockchain	RPC Framework	Web Framework	Deployment Timeout	Deployment Request BodySize
Aptos	poem-openapi	poem	**~ 5min**	**> 2MB**
Sui	jsonrpsee	axum	**< 1min**	**~ 2MB**
Solana	jsonrpc	jsonrpc-http-server	**< 1min**	**~ 50KB**
Near	jsonrpc	actix	**< 1min**	**> 2 MB**

The deployment of mainnet RPC services configures timeout & request body size restrictions to mitigate the network exhaustion attack.

Resource Exhausion: CPU

- Network
  - Flooding nodes with requests to cause bandwidth/memory issues.
- CPU
  - Overloading validator nodes with expensive computations.
- Memory
  - Consuming memory resources with large complex transactions.



CPU Resource Exhaustion

Contracts with infinite loops or excessively complex operations can cause CPU resource exhaustion or delayed block confirmation.

Mitigations by Design

- Dynamic (Runtime Gas Charging) :
  - Charge gas for every instruction executed, ensuring that resource usage (CPU) is proportional to the cost paid by the user.
- Static (Pre-execution Analysis) :
  - Use static analysis to detect potential deadloops or infinite recursions in contract code before execution.
    - The analysis also requires charging/metering.

CERTIK

Gas Charging: Challenges

- When to charge gas?
  - before or after operation
- What to charge for?
  - Instruction Execution
    - OPCode, native function
  - Instruction operands
  - External arguments
  - Stack, heap consumption
    - Memory resources.

```
Bytecode::LdU8(int_const) => {
  gas_meter.charge_simple_instr(S::LdU8)?;
  interpreter.operand_stack.push(Value::u8(*int_const))?;
}

fn charge_simple_instr(&mut self, instr: SimpleInstruction)
-> PartialVMResult<()> {
    let (pops, pushes, pop_size, push_size) =
        get_simple_instruction_stack_change(instr);
    self.charge(1, pushes, pops, push_size.into(), pop_size.into())
}
```

Gas Charging: Challenges

- When to charge gas?
- What to charge for?
- How much gas to charge?
  - unit_cost
    - type complexity
    - memory layout / size
  - base_cost
    - Is zero base_cost acceptable ?

```
//native sha2_256
gas cost:
  base_cost + unit_cost * input_length_in_bytes
```

Gas charging is an effective way to limit the resource usage (CPU, Memory).

Can a malformed tx consume huge CPU cycles with limited gas cost?

Gas Charging - Common Threats

- Charge sequence errors
  - Charging gas after execution or before?
- Incorrect gas charging
  - Charging less gas than the actual CPU usage.
  - e.g. An example of Solana Chain in next page
- Missing gas charges
  - Failing to charge gas for certain operations, such as pre-execution security validation.
    - e.g. HamsterWheel Attack in the POC2023 talk.
      - Infinite loop in the *Abstract Interpretation* Verifier.

```
Bytecode::CopyLoc(idx) => {
    // TODO(Gas): We should charge gas before
copying the value.
    let local = locals.copy_loc(*idx as usize)?;
    gas_meter.charge_copy_loc(&local)?;
    interpreter.operand_stack.push(local)?;
}
```

Solana Incorrect Gas Charging Issue

$base^{exponent}$ % modulus

```rust
/// Big integer modular exponentiation
pub fn big_mod_exp(
    base: &[u8], exponent: &[u8], modulus: &[u8]
) -> Vec<u8> {
   ...
   syscalls::sol_big_mod_exp(...) // charging & compute
   ...
}
```

$n\_bytes^2 / 33$ ➡ $n\_bytes^2 / 2 + 190$

gas charging

The input bytes length can impact the CPU consumption. Insufficient charge can lead to longer computation time.

```
-               big_modular_exponentiation_cost: 33,
+               big_modular_exponentiation_base_cost: 190,
+               big_modular_exponentiation_cost_divisor: 2,
```

programs/bpf_loader/src/syscalls/mod.rs

```
@@ -1710,13 +1710,15 @@ declare_builtin_function!(
        let input_len: u64 = std::cmp::max(input_len, params.modulus_len);

        let budget = invoke_context.get_compute_budget();
+       // the compute units are calculated by the quadratic equation `0.5 input_len^2 + 190`
        consume_compute_meter(
            invoke_context,
            budget.syscall_base_cost.saturating_add(
                input_len
                    .saturating_mul(input_len)
-                   .checked_div(budget.big_modular_exponentiation_cost)
-                   .unwrap_or(u64::MAX),
+                   .checked_div(budget.big_modular_exponentiation_cost_divisor)
+                   .unwrap_or(u64::MAX)
+                   .saturating_add(budget.big_modular_exponentiation_base_cost),
            ),
        )?;
```

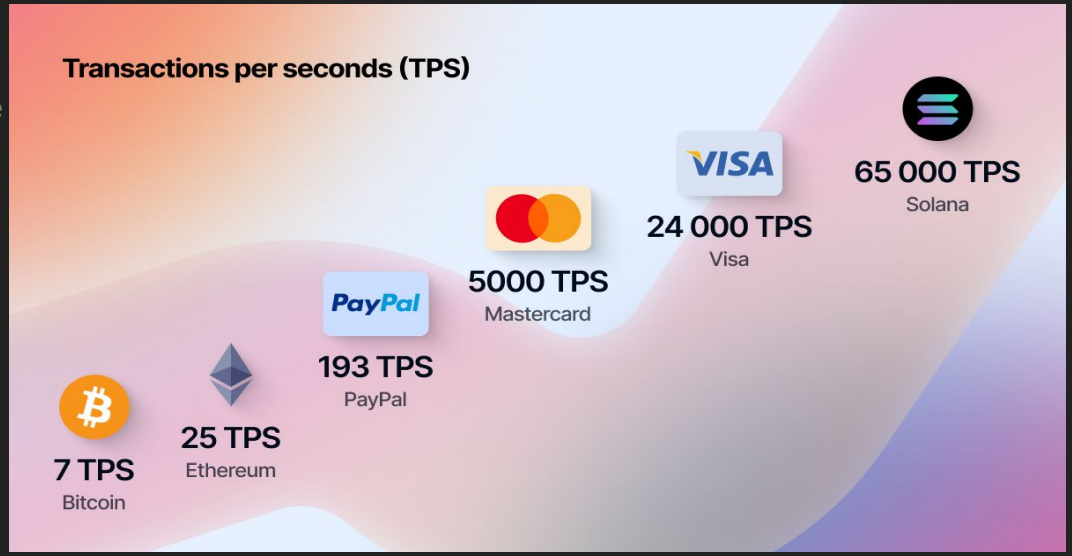
https://web3caff.com/zh/archives/105607

Solana Incorrect Gas Charging Issue

```
/// Big integer modular exponentiation
pub fn big_mod_exp(
    base: &[u8], exponent: &[u8], modulus: &[u8]
) -> Vec<u8> {
   ...
   syscalls::sol_big_mod_exp(...) // charging & compute
   ...
}
```

$n\_bytes^2 / 33$ ➡ $n\_bytes^2 / 2 + 190$

- 4096 bits input takes 890ms
- leads to timeout exception
- automatically retry 150 times
- **takes 130s to resolve the transaction**
  - **~ 0.007 TPS (Tansaction Per Second)**
  - **compared with 65000 TPS**



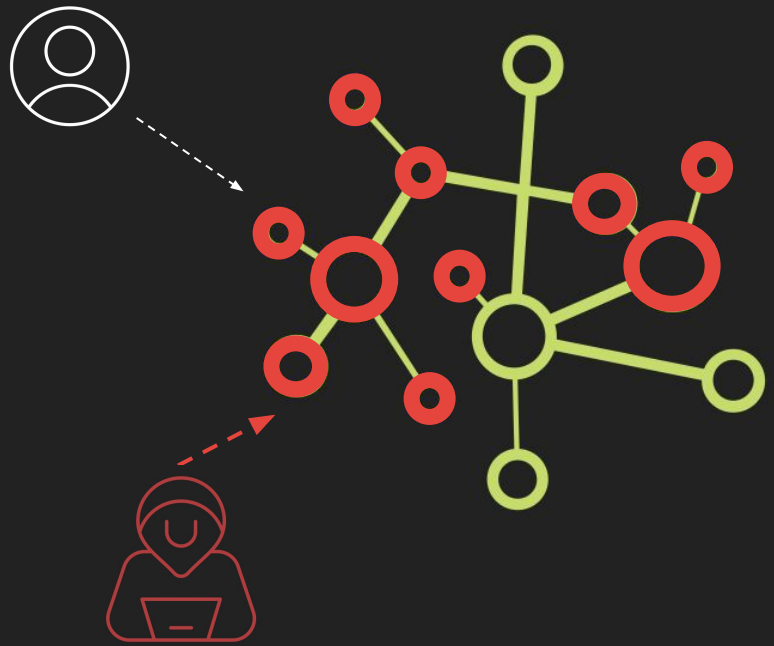
https://www.swaps.app/blog/what-is-solana-and-what-prospects-can-it-bring

CPU Resource Exhaustion Attack Consequence

- Severity: Critical
  - Exploits can be triggered by submitting transactions with low cost.
  - Delay the block confirmation or halt the entire network.
- Low Cost
  - Even with a maximum fee (e.g., $100), the attacker can significantly impact the network.
- Auto-propagation
  - Once exploited, these issues can spread across nodes, disabling validators and halting consensus.

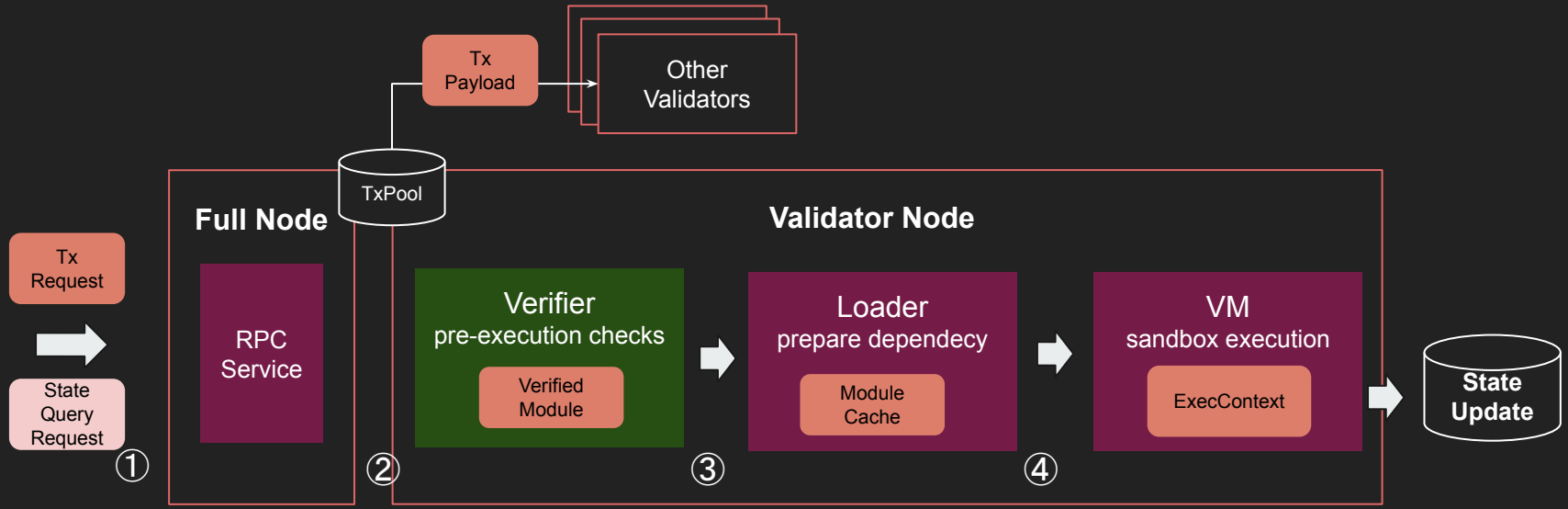
Resource Exhausion: Memory

- Network
  - Flooding nodes with requests to cause bandwidth/memory issues.
- CPU
  - Overloading validator nodes with expensive computations.
- Memory
  - Consuming memory resources with large or complex transactions.



Memory Resource Exhaustion

Malformed contracts or RPC queries can cause excessive memory usage, leading to Out-Of-Memory (OOM) exceptions, and the process being killed by the operating system.



CERTIK

Node requirements of Blockchains (2023)

Blockchain	Node Type	Memory	Contract Size Limit
Aptos (6adfb505)	Fullnode	32GB	60 KB
	Validator	16GB	
Sui (541dd94)	Fullnode	32GB	100 KB
	Validator	32GB	
Cosmos Hub	Fullnode	16GB	800 KB
	Validator	32GB	

Memory amplification (over x100,000) is required to launch memory exhaustion attacks.

Types of Memory Amplification

- Directly controlled memory allocation length

```go
func (p Precompile) ClaimRewards(
    ctx sdk.Context,    ...
    args []interface{},
) ([]byte, error) {
    // parse input arguments
    delegatorAddr, maxRetrieve, err := parseClaimRewardsArgs(args)
    ...
    validators := p.stakingKeeper.GetDelegatorValidators(
          ctx, delegatorAddr.Bytes(), maxRetrieve)
    totalCoins := sdk.Coins{}
    // ...
}
```

```go
// Return all validators that a delegator is bonded to.
If maxRetrieve is supplied, the respective amount will
be returned.
func (k Keeper) GetDelegatorValidators(
    ctx sdk.Context, delegatorAddr sdk.AccAddress,
    maxRetrieve uint32,
) types.Validators {
    validators := make([]types.Validator, maxRetrieve)
    ...
}
```

https://github.com/evmos/evmos/security/advisories/GHSA-68fc-7mhg-6f6c

- Complex executable construction
  - Use of multiple references as length
  - Fabrication of huge objects, internal states

CERTIK

What Makes Memory Amplification Possible

As blockchain evolves, achieving greater flexibility and security requires more **on-chain complexity**.

Flexibility → **WebAssembly Chains**

- *Onchain validation*
- *Onchain complication*
  - *AOT*

Security → **Move Chains**

- *Onchain validation*
- *Onchain verification*
  - *Type Safety*

Evolution of Executable Design

Bitcoin VM

- Portable
- Bitcoin Script
  - Not smart

```
<sig> <pubKey>
OP_DUP
OP_HASH160
<pubKeyHash>
OP_EQUALVERIFY
OP_CHECKSIG
```

EVM

- Turing complete
- Smart contracts
  - onchain type safety not enforced

```
contract SimpleStorage {
    uint storedData;

    function set(uint x) public {
        storedData = x;
    }
}
```

```
function compilationCheck () public {
    A memory a = A ({
        name: bytes32("aa"),
        voteCount:1024
    });
    B memory b = a;
}
```

offchain type safety

```
bytes memory x;
assembly {
  x := 0x40
}
x[0x20] = 0x42;
```

```
PUSH32 0x40
CALLDATALOAD
MSTORE
RETURN
```

inline assembly & bytecode

Both VMs introduced foundational concepts but remain limited in terms of flexibility and security.

CERTIK

Evolution of Executable Design

Enhanced Flexibility

- e.g. WebAssembly VM
- Ecosystem: Polkadot, Cosmos, etc.
  - C/Rust → LLVM → WebAssembly
  - basic type in bytecode
  - Onchain validation before execution

Enhanced Security

- e.g Move VM
- Ecosystem: Sui, Aptos, etc.

Enhanced type safety, elimination of assembly, and support for custom types for safer and more flexible contract development.

```
(module
  (import "console" "log" (func $log (param i32
i32)))
  (import "js" "mem" (memory 1))
  (data (i32.const 0) "Hi")
  (func (export "writeHi")
    i32.const 0  ;; pass offset 0 to log
    i32.const 2  ;; pass length 2 to log
    call $log
  )
)
```

```
module poc::my_module {
    struct Coin<phantom T> has store {
        value: u64,
    }

    public fun mint<T>(value: u64): Coin<T> {
        //... access control ...
        Coin<T> { value }
    }
}
```

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

No Double Spending

Attacker's Payload - 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>,
}
```

OnChain Represetation

An Example

```
// 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

Attacker can construct malformed payload through source code or CompiledModule.

```
function_handles: [
 {
   module: 0,
   name: 2,  // identifier_idx
   parameters: 0, // sig_idx
   return_: 1,  // sig_idx
   …
}
```

fun **mint**(**[signer, u64]**)

```
identifiers: [
   "poc", "Coin",
   "mint", "value"
]
```

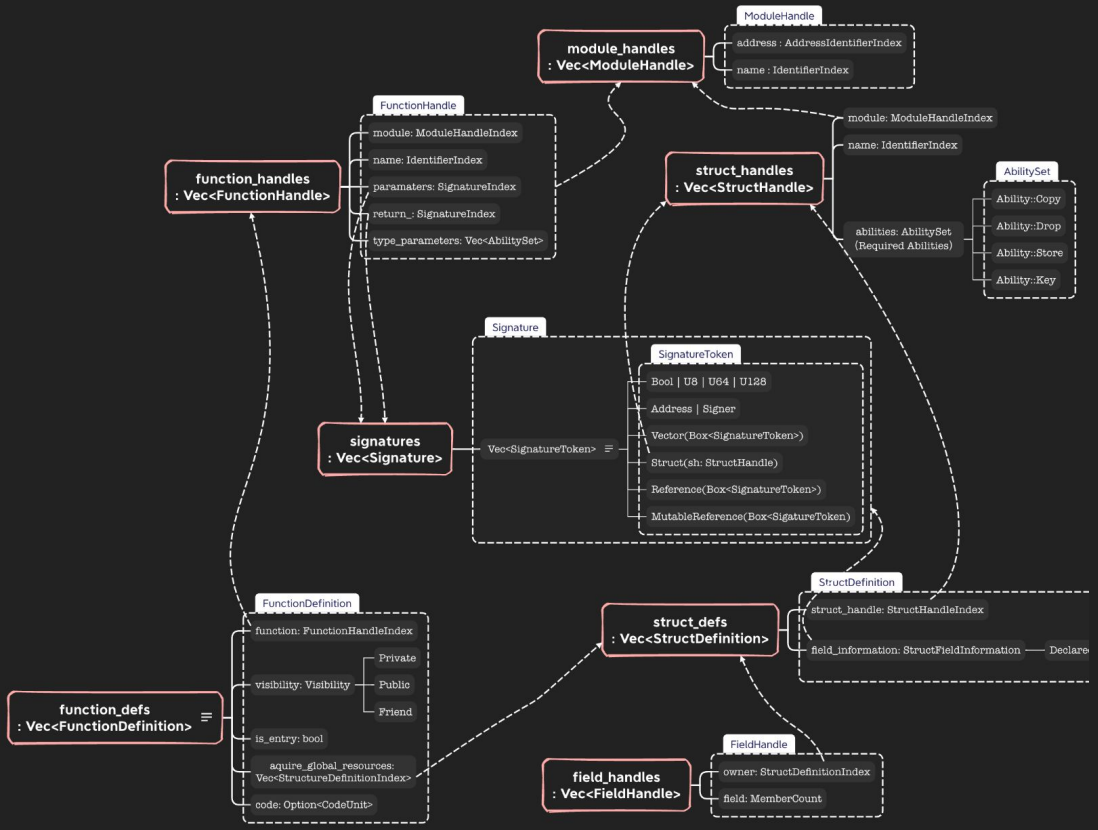
```
signatures: [
   [Signer, U64],
   []
]
```

```
function_defs: [
 {
   function: 0,
   …,
    code: {
       locals: 1,
       code: [
          (ImmBorrowLoc, 0)
          (MoveLoc, 1)
          (Pack,  0)     // struct_def_idx
          (MoveTo, 0)  // struct_def_idx
          Ret
       ]
    }
 }
]
```

```
struct_defs: [
 {
   struct_handle: 0,
   field_information: {
     Declared: [
       {
         name: 3, // identifier_idx
         signature: U64
       }
     ]
   }
 }
],
```

struct **Coin**

CompiledModule Construction: Layout & Challenges



Structural Checks

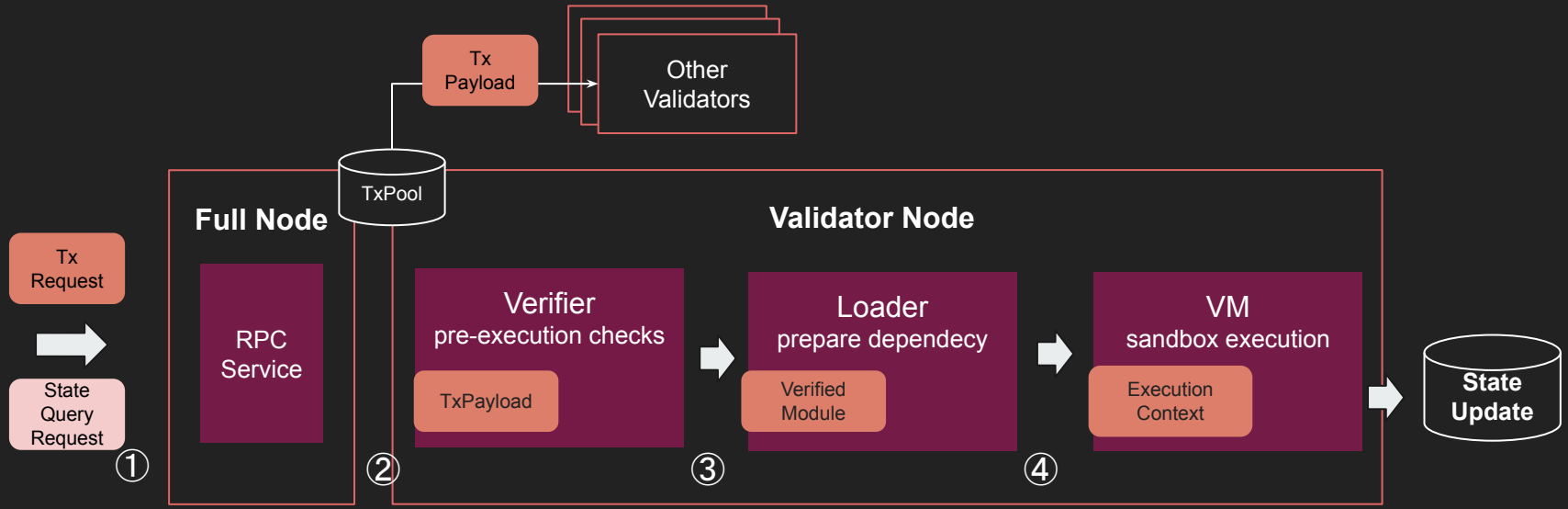
- BoundsChecker
- LimitsVerifers
- DuplicationChecker
- SignatureChecker
- InstructionCosnsitency
- RecursiveStructDefChecker
- …

Semantic Checks

- StackUsageVerifier
- TypeSafetVerifier
- LocalSafetyVerifier
- ReferenceVerifier
- AcquiresVerifier

```
max_loop_depth: Some(5),
max_generic_instantiation_length: Some(32),
max_function_parameters: Some(128),
max_basic_blocks: Some(1024),
max_basic_blocks_in_script: Some(1024),
max_value_stack_size: 1024,
max_type_nodes: Some(256),
max_push_size: Some(10000),
max_dependency_depth: Some(100),
max_struct_definitions: Some(200),
max_fields_in_struct: Some(30),
max_function_definitions: Some(1000),
```

Threat Modeling: Attacker Controlled Data



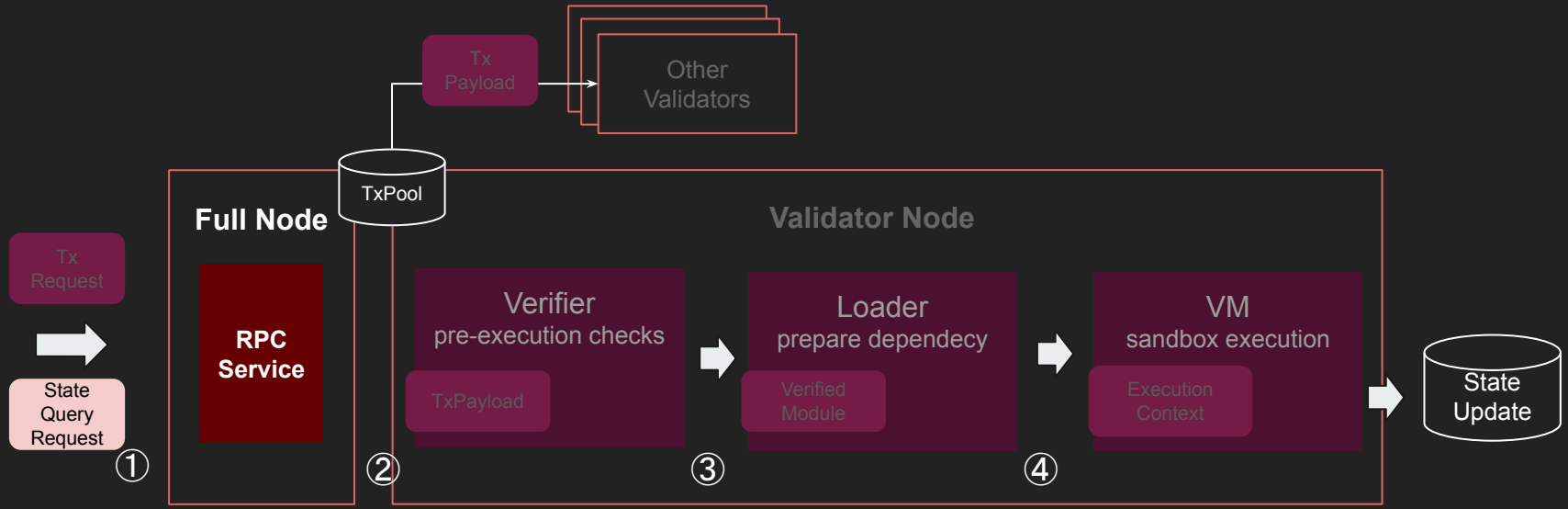
RPC Requests
- submitTransaction
- getObjectByAddress

TxPayload
- PublishModule
- FunctionInvocation
  - Target Function
  - Args

VerifiedModule
- offset in bound
- no duplicated definition
- no type safety violation
- …

ExecContext
- loaded function
- ModuleCache
  - dependent types
  - func, struct, …

Memory Exhaustion Through State Querying



RPC Requests
- submitTransaction
- getObjectByAddress

TxPayload
- PublishModule
- FunctionInvocation
  - Target Function
  - Args

VerifiedModule
- offset in bound
- no duplicated definition
- no type safety violation
- …

ExecContext
- loaded function
- ModuleCache
  - dependent types
  - func, struct, …

RPC API Example: sui_getObject

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "method": "sui_getObject",
  "params": [
    "0xd753f9d8a801c82…aab53bef16dec7",
    {
      "showType": true,
      "showOwner": true,
      "showPreviousTransaction": true,
      "showDisplay": true,
      "showContent": true,
      "showBcs": true,
      "showStorageRebate": true
    }
  ]
}
```

Request

{'jsonrpc': '2.0', 'result': {'data': {'objectId': '0x31f1d8ca0f339cf6f54fef66d89acfebd192d49dc503205dbe38a392441d01c4', 'version': '1', 'digest': 'J7yxwVZ9nNrkHnSjK9rxps1hhRZJsyo5VjEMWFhx9PKk', 'type': 'package', 'owner': 'Immutable', 'previousTransaction': 'AkXHEJivMXht55b4ZPiaRcLepi3xGMMcfe1omTecE47L', 'storageRebate': '1421200', 'display': {'data': None, 'error': None}, 'content': {'dataType': 'package', 'disassembled': {'poc': '// Move bytecode v6\nmodule 31f1d8ca0f339cf6f54fef66d89acfebd192d49dc503205dbe38a392441d01c4.poc {\n\n\n\n\nentry public foo() {\nB0:\n\t0: LdConst[0](vector<u8>: "AAA..)\n\t1: Pop\n\t2: Ret\n\n}\n\nConstants [\n\t0 => vector<u8>: "AAA" // interpreted as UTF8 string\n]\n}'}}, 'bcs': {'dataType': 'package', 'id': '0x31f1d8ca0f339cf6f54fef66d89acfebd192d49dc503205dbe38a392441d01c4', 'version': 1, 'moduleMap': {'poc': 'oRzrCwYAAAAHAQACAwIFBQcBBwgICBAgBjAHDDcKAAEAAAAAAAAADZm9vA3BvYzHx2MoPM5z29U/vZtiaz+vRktSdxQMgXb44o5JEHQHECgIEA0FBQQABBAAAAwcAAQIA'}, 'typeOriginTable': [], 'linkageTable': {}}}}, 'id': 1}

Response

The Disassembled Field in the Response

'disassembled': {'poc': '// Move bytecode v6\nmodule 31f1d8ca0f339cf6f54fef66d89acfebd192d49dc503205dbe38a392441d01c4.poc {\n\n\n\nentry public foo() {\nB0:\n\t0: LdConst[0](vector<u8>: "AAA..)\n\t1: Pop\n\t2: Ret\n\n}\n\nConstants [\n\t0 => vector<u8>: "AAA" // interpreted as UTF8 string\n]\n}'}

```
// Move bytecode v6
module d753f9d8a801833…69212faab53bef16dec7.poc {


entry public foo() {
B0:
        0: LdConst[0](vector<u8>: "AAA..)
        1: Pop
        2: Ret

}

Constants [
        0 => vector<u8>: "AAA" // interpreted as UTF8 string
]
}
```

```
module poc::poc {
    const V: vector<u8> = vector[65, 65, 65];
    public entry fun foo() {
        let _v = V;
    }
}
```

Source code of the Onchain Module

FunctionDef[0].code
- LdConst 0 // **const_idx**
- Pop
- Ret

→

ConstantPool[0]
- **type: Vector(u8)**
- **data: 3, 65, 65, 65**

Fields in CompiledModule (OnChain Module)

The Attacking Flow To Exhaust FullNode Memory

Request
- sui_getObject
- param[0]: object_id
- **"showContent": true,**

FullNode
- Read module from StateDB
- Disassemble module.code
- Serialize Response

FunctionDef[0].code
- **LdConst 0**
- Pop
- Ret

ConstantPool[0]
- **type: Vector(u8)**
- **data: 3, 65, 65, 65**

Response (JSON)

```
{   …
   'content' : {
      disassembled: {
          …
          0: LdConst[0](vector<u8>:
"AAA..)
          ...
      }
     }
}
```

A carefully craft CompileModule may result in super long disassembled str, making the disassembly routine and the result to consume huge memory.

CERTIK

Payload Construction Ideas: Huge Response

ConstantPool[0]
- type: Vector(u8)
- ~~data: 3, 65, 65, 65~~
- data: 1024, 65, 65, …, 65

Increase length

FunctionDef[0].code
- LdConst 0 // 2 bytes
- **LdConst 0 // 2 bytes**
- …
- Pop
- Ret

Make many refs

```
// Move bytecode v6
module d753f9d8a801833…69212faab53bef16dec7.poc {
entry public foo() {
B0:
        0: LdConst[0](vector<u8>:
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA..)
        1: LdConst[0](vector<u8>:
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA..)
        ...
        2: Pop
        3: Ret

}

Constants [
        0 => vector<u8>:
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA" // interpreted as UTF8
string
]
}
```

Overcome Constrains

- Max data length in constant pool: 1024 * 1024
- Function code Length Limit: 65536
- Module Package Size Limit: 102400

CERTIK

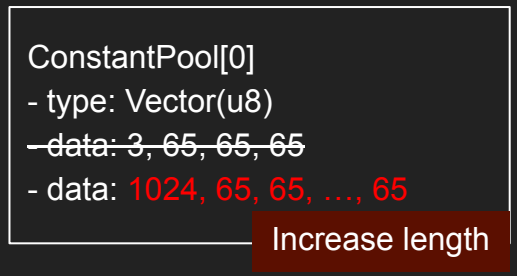
RPC Memory Exhaustion

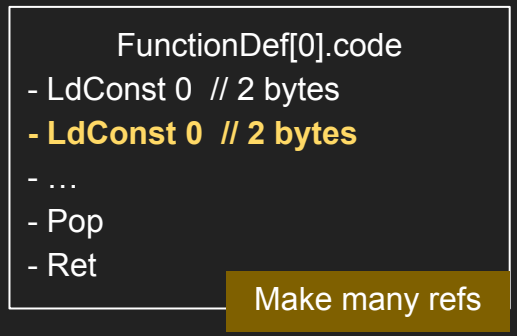
Takeaways

- Amplification can occur when low-cost references lead to significant memory allocations.
- Total Memory Usage >= Memory allocated per reference * Number of references

The Fix

- Remove the `showContent` option
- No disassembly logic in the RPC handling flow





Other Memory Exhaustion Targets

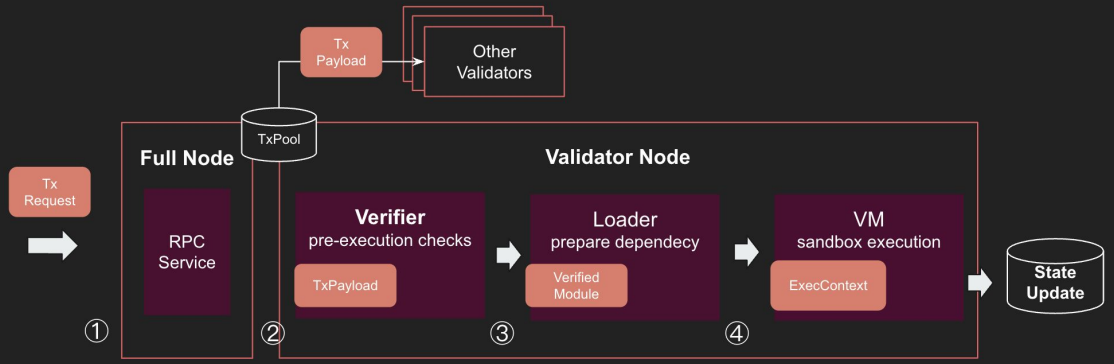
Targeted Phases in Validator (where accumulated resource are allocated)

- Verifier

  maintaining states for static analysis
- Loader

  resolve depedencies before execution
- Runtime

  hold all runtime Values in the Context



RPC Requests
- submitTransaction
- getObjectByAddress

TxPayload
- PublishModule
- FunctionInvocation
  - Target Function
  - Args

VerifiedModule
- offset in bound
- no duplicated definition
- no type safety violation
- …

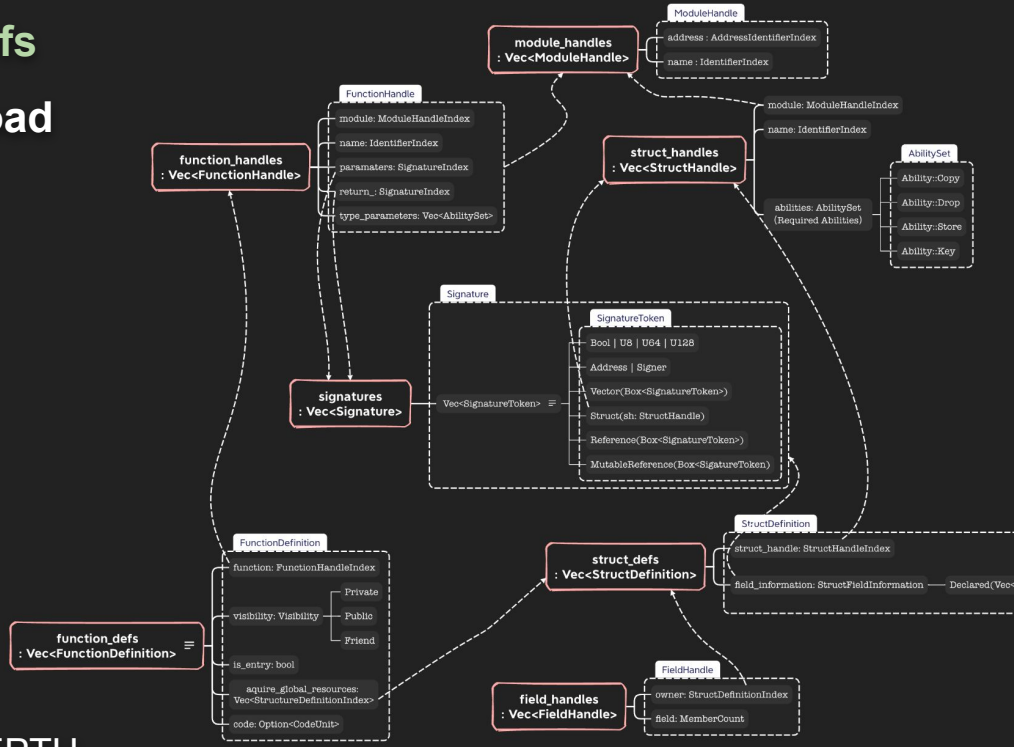
ExecContext
- loaded function
- ModuleCache
  - dependent types
  - func, struct, …

Other Payload Construction Ideas

Memory allocated per ref * Number of refs

Leveraging the compact design of payload

- Tables
  - Const, Identifier, Signature
- Reference to Table Entries
  - FunctionDef
  - StructDef
  - …
- Generic Types
  - Type Instantiation



Resolve multiple constrains

- MAX_IDENT_LEN, MAX_RECURSIVE_DEPTH
- Existing Meters & Gas Chargers

Mitigations of Memory Exhaustion Attack

- Increase node's memory requirement

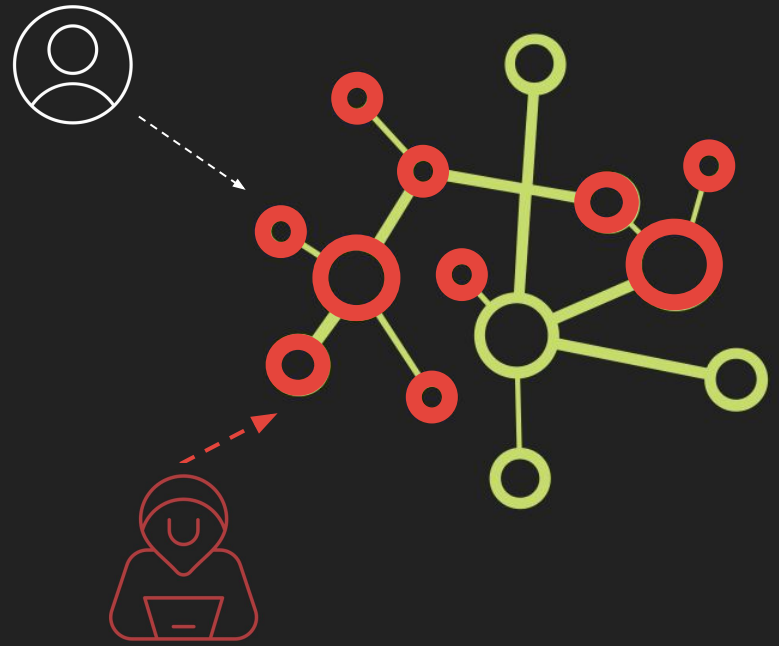
- Add meters at PreExecution time
  - e.g. charge for type information
- Stricter charger at Runtime
  - e.g. base_cost for zero length vector
  - e.g. charge for complexity
- Load time optimization
  - e.g. use shared ref, Arc<Object>

Blockchain	Node Type	Memory
Aptos	Fullnode	32GB
	Validator	16GB ⇒ **64 GB**
Sui	Fullnode	32GB ⇒ **128GB**
	Validator	32GB ⇒ **128GB**

Node memory requirement (2024)

Recap: Dimensions Of Resource Exhausion

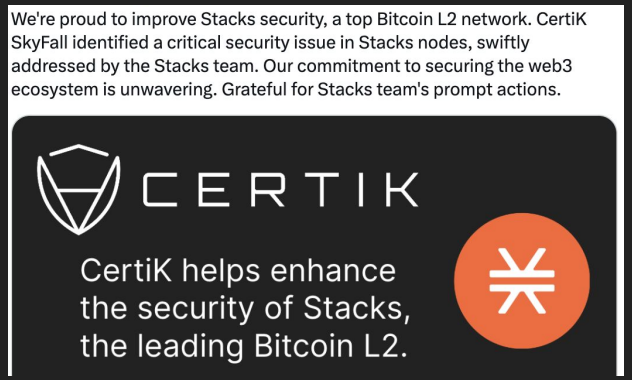
- Network
  - Missing middleware configurations
- CPU
  - Flawed Gas Charger | Meter
    - Missing, inaccurate Charger
- Memory
  - Ignorance of resource usage leads to amplification pattern in core component



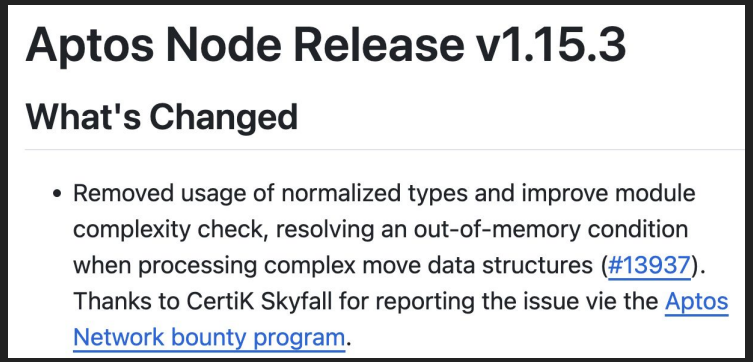
Security Enhancement of Blockchain Ecosystems

https://medium.com/immunefi/sui-temporary-total-network-shutdown-bugfix-review-c271d0319dcc

https://www.certik.com/resources/blog/risk-and-security-enhancement-for-app-chains-an-in-depth-writeup-of-cwa-2023



https://x.com/CertiKSkyfall/status/1786835360628306382



https://github.com/aptos-labs/aptos-core/releases/tag/aptos-node-v1.15.3

Summary

- Blockchains are complex "world machines" with evolving executable designs
  - enhance flexibility and security
  - increase the burden of on-chain measures
- We provide a comprehensive look at resource exhaustion attacks
  - Targeting: network, CPU, and memory
  - Consequence: disrupt entire networks with minimal payloads.
- We share real-world cases illustrating payload construction ideas that can exhaust node resources, analyzing the root causes and discussing migitations.

CERTIK

Thank You