Locate Vulnerabilities of Ethereum Smart Contracts with Semi-Automated Analysis

Boik Su
Senior security researcher, CyCraft
Boik Su

- Senior Security Researcher @ CyCraft
- CHROOT’s member, a local hacker group in Taiwan
- Specialization
  - Web Security / AD Security / Blockchain Security
- Gives talks at
  - OWASP Global AppSec / ROOTCON / HITCON
Outline

- Intro to Blockchain & Web3
- EVM-based Smart Contract Basics
- Reverse Engineering & CFG
- Cases & Futures
Outline

• Intro to Blockchain & Web3
• EVM-based Smart Contract Basics
• Reverse Engineering & CFG
• Cases & Futures
Distributed Ledger Technology (DLT)

● “Bitcoin”, the first cryptocurrency, was invented in 2008 by an unknown person or group of people using the name Satoshi Nakamoto

● The term “Blockchain” was later invented due to the release of the white paper and its fundamental cores, Peer-to-Peer Network and Consensus Algorithm

● “DLT” is later named as a category that covers technologies like Blockchain, having high levels of transparency, integrity and availability in a decentralized framework
Peer-to-Peer Network

- Web 2.0, known as Social Network, focuses on sharing data and contents under famous entities such as Google, Meta, Apple, ...
- Web3, known as Blockchain-empowered Network, focuses on the controls of owned data and identities
Consensus Algorithm

- The most important component of a blockchain that ensures the safety of the network
- Participants need to fulfill certain requirements to make a transaction
- The mainstream ones
  - PoW (Proof-of-Work)
  - PoS (Proof-of-Stake)
  - PoH (Proof-of-History)
Consensus Algorithm

- The most important component of a blockchain that ensures the safety of the network.
- Participants need to fulfill certain requirements to make a transaction.
- The mainstream ones:
  - PoW (Proof-of-Work)
  - PoS (Proof-of-Stake)
  - PoH (Proof-of-History)

\[
\text{hash( previous hash, 1390, root hash )} = 0101011111011111
\]

Source: BitcoinUnlimited.info
How does “blockchain” work?
How does the “network” look like?
Generations

● 1st Gen: Bitcoin blockchain (Payment System)
● 2nd Gen: Ethereum blockchain (On-chain traditional finance)
● Next Gen?
  ○ IoT (Internet of Things)
  ○ AI (Artificial Intelligence)
Generations

- 1st Gen: Bitcoin blockchain (Payment System)
- 2nd Gen: Ethereum blockchain (On-chain traditional finance)
- Next Gen?
  - IoT (Internet of Things)
  - AI (Artificial Intelligence)
  - Superconductor (?)
Outline

- Intro to Blockchain & Web3
- EVM-based Smart Contract Basics
- Reverse Engineering & CFG
- Cases & Futures
Ethereum blockchain & Smart Contract

- Ethereum blockchain introduces a new function called “Smart Contract”, which is simply a program run on the blockchain.
- Smart contracts can define rules, like a regular contract, and automatically enforce them via the code.
- Dapps (Decentralized Apps) have their backend code (smart contracts) running on a blockchain like Ethereum to ensure decentralization and availability.
- “DeFi (Decentralized Finance)” then starts thriving.
Short ver.

Web 2.0

Diagram by Preethi Kasireddy
Ethereum VM (Virtual Machine)

- Smart contracts run on EVM (Ethereum VM)
- The EVM executes as a stack machine, and each compiled smart contract bytecode executes as a series of EVM opcodes like XOR, AND, ADD, SUB, etc
- Each EVM opcode is 1-byte, and therefore, we can have 256 different opcodes at maximum (142 currently)
- Each programmable computation is intrinsically bounded by fees, which is a specific amount of gas
Outline

- Intro to Blockchain & Web3
- EVM-based Smart Contract Basics
- Reverse Engineering & CFG
- Cases & Futures
Hacks in Web3

Web 2.0

Browser

Internet

Web server

Front-end

JavaScript, HTML, CSS

Back-end

Node.js, Python, Java, Go, etc.

Database

Web3

Browser

Internet

Web server

Front-end

JavaScript, HTML, CSS

Ethereum Virtual Machine

Smart contracts

Ethereum blockchain
Hacks in Web3 (Front-End)

It's Web 2.0 things...
Hacks in Web3 (Front-End)

Palisade identifies Wormable Cross-Site Scripting Vulnerability affecting Rarible’s NFT Marketplace
Hacks in Web3 (Front-End)

Palisade identifies Wormable Cross-Site Scripting Vulnerability affecting Rarible’s NFT Marketplace

XSS (Cross-Site Scripting)
Hacks in Web3 (Front-End)

Palisade identifies Wormable Cross-Site Scripting Vulnerability affecting Rarible’s NFT Marketplace

DNS Cache Spoofing
Hacks in Web3 (Front-End)

Palisade identifies Wormable Cross-Site Scripting Vulnerability affecting Rarible’s NFT Marketplace

UI Spoofing
Hacks in Web3 (Back-End)

Web 2.0

Web server

Front-end
JavaScript, HTML, CSS

Back-end
Node.js, Python, Java, Go, etc.

Database

Web 3

Web server

Front-end
JavaScript, HTML, CSS

Ethereum Virtual Machine

Smart contracts

Ethereum blockchain

Smart Contract
Hacks in Web3 (Back-End)

UNISWAP BUG BOUNTY

Uniswap Labs recently advertised a boosted $3M bounty program for bug reports. To our knowledge, ours was the only bug report that Uniswap acted upon.

1/ The Harmony team has identified a theft occurring this morning on the Horizon bridge amounting to approx. $100MM. We have begun working with national authorities and forensic specialists to identify the culprit and retrieve the stolen funds.

More 📄

7:13 AM · Jun 24, 2022

The wormhole network was exploited for 120k wETH.

ETH will be added over the next hours to ensure wETH is backed 1:1. More details to come shortly.

We are working to get the network back up quickly. Thanks for your patience.

6:25 AM · Feb 3, 2022
Hacks in Web3 (Back-End)

UNISWAP BUG BOUNTY

Uniswap Labs recently advertised a boosted $3M bounty program for bug reports. To our knowledge, ours was the only bug report that Uniswap acted upon.

Harmony (@harmonyprotocol · Follow)

1/ The Harmony team has identified a theft occurring this morning on the Horizon bridge amounting to approx. $100MM. We have begun working with national authorities and forensic specialists to identify the culprit and retrieve the stolen funds.

More ☰
7:13 AM · Jun 24, 2022

Wormhole (@wormholecrypto · Follow)

The wormhole network was exploited for 120k wETH.

ETH will be added over the next hours to ensure wETH is backed 1:1. More details to come shortly.

We are working to get the network back up quickly. Thanks for your patience.

6:25 AM · Feb 3, 2022
Hacks in Web3 (Back-End)

Total value stolen in crypto hacks and number of hacks, 2016 - 2022

© Chainalysis
Reverse Engineering a Contract

- Everything uploaded on the blockchain is **consistent**, **verifiable**, and publicly **available**
- For transparency and reputation, some projects will disclose their source code on **GitHub**, **Etherscan**, etc
- If you want, you can always get a copy of a smart contract bytecode even if it’s not open-sourced
- There are **no secrets** on the blockchain...
White Box Testing

- Which means that we have the source code

```solidity
function allowListMint(uint256 quantity, bytes32[] calldata proof)
    external
    payable
    callerIsUser
{
    uint256 price = uint256(saleConfig.mintlistPrice);
    require(price != 0, "pre sale has not begun yet");
    require(
        allowlist[msg.sender] < 1,
        "You can only mint once during pre-sale."
    );
    bytes32 leaf = keccak256(abi.encodePacked(msg.sender));
    require(_verify(leaf, proof), "Invalid Signature proof supplied.");
    require(totalSupply() + quantity <= collectionSize, "reached max supply");
    require(price <= msg.value, "Invalid funds provided");
    allowlist[msg.sender]++;
    _safeMint(msg.sender, quantity);
    refundIfOver(price);
}
```
White Box Testing

- Which means that we have the source code
- We can take advantage of static-analysis tools to easily discover flaws

```solidity
function allowListMint(uint256 quantity, bytes32[] calldata proof)
    external
    payable
    callerIsUser
{
    uint256 price = uint256(saleConfig.mintlistPrice);
    require(price != 0, "pre sale has not begun yet");
    require(
        allowlist[msg.sender] < 1,
        "You can only mint once during pre-sale."
    );
    bytes32 leaf = keccak256(abi.encodePacked(msg.sender));
    require(_verify(leaf, proof), "Invalid Signature proof supplied.");
    require(totalSupply() + quantity <= collectionSize, "reached max supply");
    require(price <= msg.value, "Invalid funds provided");
    allowlist[msg.sender]++;
    _safeMint(msg.sender, quantity);
    refundIfOver(price);
}
White Box Testing

- Which means that we have the source code
- We can take advantage of static-analysis tools to easily discover flaws
- Can you spot the vuln?
White Box Testing

- Which means that we have the source code
- We can take advantage of static-analysis tools to easily discover flaws
- You can mint as many items as you want by paying only the price of one item

```solidity
function allowListMint(uint256 quantity, bytes32[] calldata proof)
    external
    payable
    callerIsUser

    { uint256 price = uint256(saleConfig.mintlistPrice);
    require(price != 0, "pre sale has not begun yet");
    require(
        allowlist[msg.sender] < 1,
        "You can only mint once during pre-sale."
    );
    bytes32 leaf = keccak256(abi.encodePacked(msg.sender));
    require(verify(leaf, proof), "Invalid Signature proof supplied.");
    require(totalSupply() + quantity <= collectionSize, "reached max supply");
    require(price <= msg.value, "Invalid funds provided");
    allowlist[msg.sender]++;
    _safeMint(msg.sender, quantity);
    refundIfOver(price);
    }
```
White Box Testing

- Which means that we have the source code.
- We can take advantage of static-analysis tools to easily discover flaws.
- You can mint as many items as you want by paying only the price of one item.
Black Box Testing

● You can try “Replay Attack”, which simply means you replay the Tx to see if you’re able to reproduce the outcome

● Some will also analyze transactions to understand internal operations

Decoded Actions

º [transfer] amount=0.159 [ETH], from=0x00000
[transfer] id=1910 [ISEKAI], from=0x0000007
[transfer] amount=0.00425 [WETH], from=0xa7
[transfer] amount=0.16575 [WETH], from=0xa7

Call Trace
º [call] [231631] 0x000007370AF0000aD00Be0EFD
[call] [4786] [()] [Isekai Meta: ISEKAI Token]
[staticcall] [2534] [()] [Wrapped Ether].ba
º [call] [111479] [()] [Seaport: Seaport v1.1]
[log] OrderFulfilled(topic_0=0x9d9af8e3}
Black Box Testing

- You can try “Replay Attack”, which simply means you replay the Tx to see if you’re able to reproduce the outcome
- Some will also analyze Txs to understand internal operations
- Or, you can reverse smart contracts, and it will give you a much clearer view of what smart contracts do actually
Disassembly

- No matter what compiled binaries we have, it’s a must to firstly disassemble machine code into disassembly
CFG (Control Flow Graph)

Source Code

```plaintext
w = 0;
x = x + y;
y = 0;
if( x > z )
{
y = x;
x++;
}
else
{
y = z;
z++;
}
w = x + z;
```

Basic Blocks

Flow Graph
Why do we need to construct a CFG
1. To have correct executing logics

L1: MOV EAX, $2
L2: MUL EAX, ECX
L3: MOV DWORD [0x402000], EAX
Why do we need to construct a CFG

1. To have correct executing logics

L1: MOV EAX, $2
L2: MUL EAX, ECX
L3: MOV DWORD [0x402000], EAX
CFG (Control Flow Graph)

- Why do we need to construct a CFG
  1. To have correct executing logics

L1: MOV EAX, $2
L2: MUL EAX, ECX
L3: MOV DWORD [0x402000], EAX

\[ ECX \times EAX = ECX \times 2 \]
CFG (Control Flow Graph)

- Why do we need to construct a CFG
  1. To have correct executing logics

L1: MOV EAX, $2
L2: MUL EAX, ECX
L3: MOV DWORD [0x402000], EAX
CFG (Control Flow Graph)

- Why do we need to construct a CFG
  1. To have correct executing logics

<table>
<thead>
<tr>
<th>L0:</th>
<th>MOV EAX, $3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CMP EBX, $0</td>
</tr>
<tr>
<td></td>
<td>JNE L2</td>
</tr>
<tr>
<td>L1:</td>
<td>MOV EAX, $2</td>
</tr>
<tr>
<td>L2:</td>
<td>MUL EAX, ECX</td>
</tr>
<tr>
<td>L3:</td>
<td>MOV DWORD [0x402000], EAX</td>
</tr>
</tbody>
</table>
Why do we need to construct a CFG
1. To have correct executing logics

L0: MOV EAX, $3
    CMP EBX, $0
    JNE L2
L1: MOV EAX, $2
L2: MUL EAX, ECX
L3: MOV DWORD [0x402000], EAX
Why do we need to construct a CFG

1. To have correct executing logics

```
L0: MOV EAX, $3
   CMP EBX, $0
   JNE L2
L1: MOV EAX, $2
L2: MUL EAX, ECX
L3: MOV DWORD [0x402000], EAX
```

\[ \text{Source Code} \]

\[ \text{Basic Blocks} \]

\[ \text{Flow Graph} \]
Why do we need to construct a CFG

1. To have correct executing logics
2. To eliminate loops

CFG (Control Flow Graph)
CFG (Control Flow Graph)

- Why do we need to construct a CFG
  1. To have correct executing logics
  2. To eliminate loops
Why do we need to construct a CFG
1. To have correct executing logics
2. To eliminate loops
Why do we need to construct a CFG
1. To have correct executing logics
2. To eliminate loops
3. To transform into SSA form and lift to a higher-level abstraction

```
x := n        x0 := n
y := m        y0 := m
x := x + y    x1 := x0 + y0
return x      return x1
```
Why do we need to construct a CFG
1. To have correct executing logics
2. To eliminate loops
3. To transform into SSA form and lift to a higher-level abstraction

Okay, then how do we get our hands dirty in making a CFG?

To make it easy, intermediate-language-based analysis could make it easy for us (BNIL, P-Code, Microcode, AIL)
CFG (Control Flow Graph)

```plaintext
0 @ 00000000 push(0x430504000000eb)
1 @ 00000021 push(pop == 0)
2 @ 00000022 push(0x45)
3 @ 00000024 temp.32 = pop
4 @ 00000024 temp.32 = 0x45
5 @ 00000024 jump(temp.32 -> 6 @ 0x46)

6 @ 00000046 push(0x2c20776f726c648a)
7 @ 00000046 push(0)
8 @ 00000055 temp.32 = pop
9 @ 00000055 temp.32 = 0
10 @ 00000055 temp.32 = pop
11 @ 00000055 temp.32 = 0x2c20776f726c648a
12 @ 00000065 [temp8..32].32 = temp1.32
13 @ 00000066 push(0x45)
14 @ 00000066 push(0x13)
15 @ 00000055a <return> jump(pop)
```

This function has been analyzed with basic analysis only. Enable full analysis of this function.
To guide BN to construct a CFG from an unknown architecture, we firstly need to convert the machine code to the disassembly.
Secondly, we need to tell BN when to branch out, and therefore, BN will construct the CFG for us
Secondly, we need to tell BN when to branch out, and therefore, BN will construct the CFG for us

- Having said that, branches information are sometimes hard to be deduced
Secondly, we need to tell BN when to branch out, and therefore, BN will construct the CFG for us.

- Having said that, branches information are sometimes hard to be deduced.
- VSA (Value Set Analysis) is a static analysis approach that finds an over-approximation of the values that a location could take at a given program point.
- This can be used to understand the possible targets of indirect jumps, or the possible targets of memory / register write operations.
Secondly, we need to tell BN when to branch out, and therefore, BN will construct the CFG for us

- Having said that, branches information are sometimes hard to be deduced
- VSA (Value Set Analysis) is a static analysis approach that finds an over-approximation of the values that a location could take at a given program point
- This can be used to understand the possible targets of indirect jumps, or the possible targets of memory / register write operations
- Though it suffers from a lack of accuracy, it’s sound
Secondly, we need to tell BN when to branch out, and therefore, BN will construct the CFG for us

- Due to the introduction of the “Gas”, we can simulate every execution steps of smart contracts
- We set an upper bound of the remaining amount of the gas, there will be no infinite steps to follow, or issues like DoS (Denial of Service)
Secondly, we need to tell BN when to branch out, and therefore, BN will construct the CFG for us

○ Due to the introduction of the “Gas”, we can simulate every execution steps of smart contracts

○ We set an upper bound of the remaining amount of the gas, there will be no infinite steps to follow, or issues like DoS (Denial of Service)

○ We can now get accurate values that a location could take at a given program point of stacks / memories / and built-in functions
function compute_CFG() {
    function getBranch(instruction, stack) {
        const branch = {
            instruction,
            nextPc: -1,
            trueBranch: parseInt(stack[0], 16),
            falseBranch: instruction.pc + 1,

            runState: {
                rawMemory: undefined,
                rawStack: undefined,
            }
        }
    }

    if (instruction.name === 'JUMP' && stack.length >= 2) {
        branch.nextPc = (stack[1] > 0) ? branch.falseBranch : branch.trueBranch;

        return branch;
    }

    if (instruction.name === 'JUMP' && stack.length >= 1) {
        if (this.jumpTable.hasOwnProperty(instruction.pc)) {
            this.jumpTable[instruction.pc].instructions.push(branch.instruction);
            this.jumpTable[instruction.pc].trueBranches.push(branch.trueBranch);
        } else {
            this.jumpTable[instruction.pc] = { instructions: [instruction], trueBranches: [branch.trueBranch] };
        }
    }
}
null,
null,
{
  "instructions": [
    {
      "pc": 36,
      "name": "JUMP"
    }
  ],
  "trueBranches": [69]
}
]
// Now, we get the interpreter
this.interpreter = new Interpreter((await this.startExecution(value, data)).interpreter);

// start!
this.round += 1;
this.interpreter.reset(this.gasLimit);
await this.interpreter.run();

while (grey.length) {
    const branch = grey.shift();

    this.round += 1;
    this.interpreter.reset(this.gasLimit);
    await this.interpreter.run(branch);

    bb.walked = true;

    // get the branch that won't follow this time
    const branch = getBranch.call(this, this.instructions[pc], stack);

    if (branch) {
        branch.runState.rawMemory = Buffer.from(this.interpreter._interpreter._runState.memory._store);
        branch.runState.rawStack = Array.from(this.interpreter._interpreter._runState.stack._store);
        branch.runState.rawStack.pop();
        branch.runState.rawStack.pop();
        grey.unshift(branch);
    }
}
CFG from Binary Ninja

```javascript
if (this.instructions[pc].name === 'MSTORE') {
    this.mStores[pc] = [stack[0], stack[1]];
}

if (this.instructions[pc].name === 'RETURNDATASIZE') {
    this.returnDataSizes[pc] = bigIntToHex(this.interpreter.getReturnDataSize());
}

const funcPc = (bb.funcSig) ? ((branch) ? branch.trueBranch : parseInt(stack[0], 16)) : null;

// if we just found a possible function signature, we label the function
if (funcPc) {
    this.add_function({ pc: funcPc, name: bb.funcSig });
}
```
Finally, we give BN these pieces of information via its APIs:

- **get_instruction_text**
  - A list of InstructionTextToken objects for the instruction at the given virtual address with data

- **get_instruction_info**
  - An InstructionInfo object for the instruction at the given virtual address with data

- **get_instruction_low_level_il**
  - Appends LowLevelILExpr objects to the il variable for the instruction at the given virtual address with data
The Bytes Must Flow!

Binary Ninja 3.3 (Arrakis) is now available.

You may have noticed that we’ve introduced a new set of codenames for upcoming releases based on an alphabetical list of famous Sci-Fi/Fantasy planets. Our first release in this theme is named after the famous desert planet from Dune, Arrakis.

So what spiky goodies are in this release?
- Decompiler Improvements
  - Parameter Rejection
  - Improved Objective-C Support
  - Automatic Outlining
- Debugger
  - Type Interactions
    - Create Array Dialog
    - Import/Export Header Files
    - Enumeration Dialog
    - More Windows Improvements
● Intro to Blockchain & Web3
● EVM-based Smart Contract Basics
● Reverse Engineering & CFG
● Cases & Futures
Cases & Futures

- **SAG**? from DEF CON 2018 Quals
  - It gives us a proxy contract

```solidity
contract SagProxy {
    event PrizeRequest(bytes32 msgHash, uint8 v, bytes32 r, bytes32 s);
    event PrizeReady(address winner, bytes prize);

    Sag private sag;

    address private owner;
}
**SAG**? from DEF CON 2018 Quals

- It gives us a proxy contract to interact with the private contract behind

```solidity
contract SagProxy {
    event PrizeRequest(bytes32 msgHash, uint8 v, bytes32 r, bytes32 s);
    event PrizeReady(address winner, bytes prize);

    Sag private sag;

    address private owner;
}
```

08033d000001d5b305053004d513d60206110130103103700000158b0f492ac22be013a619afc0486aaa433dad38d928098dabe86572602018360000191660001916815260201826000191660001916805b949350505055056001a165627a7a72305820563788b2b3c0a0289000000000000a019c785322b921a84d086502da00dbdb993fba
Cases & Futures

- **SAG**? from DEF CON 2018 Quals
  - It gives us a proxy contract to interact with the private contract behind
  - All we know is to pass the function: `gamble(guess, seed)`

```solidity
def gamble(uint256 guess, uint256 seed) public
{
  sag.gamble(guess, seed);
}
```
Cases & Futures

- **SAG**? from DEF CON 2018 Quals
  - It gives us a proxy contract to interact with the private contract behind.
  - All we know is to pass the function: `gamble(guess, seed)`
  - Then, we request the prize.

```solidity
function requestPrize(bytes32 msgHash, uint8 v, bytes32 r, bytes32 s) public
    returns (bool is_winner)
{
    if (ecrecover(msgHash, v, r, s) == msg.sender && sag.isWinner(msg.sender)) {
        emit PrizeRequest(msgHash, v, r, s);
        return true;
    }
    return false;
}
```
Cases & Futures

- **SAG**? from DEF CON 2018 Quals
  - It gives us a proxy contract to interact with the private contract behind
  - All we know is to pass the function: `gambler(guess, seed)`
  - Then, we request the prize
  - The **Sag** contract isn't published and verified, so we reverse it
Cases & Futures

Recent

1. /Users/book/Documents/blockchain/contracts/0ва019c7b6322b921a8d0d06502dea0d0d6db993ba.evm
2. /Users/book/Documents/blockchain/contracts/eff.evm
3. /Users/book/Documents/blockchain/contracts/0x253e528563e146f8e6eeb219d456a6b7677b1e19.evm
4. /Users/book/Documents/blockchain/contracts/0x47154d63747536694bfcc79546c88a.evm
5. /Users/book/Documents/blockchain/contracts/0x379b8e23c5aa1ecbf53f43b9945178321fc5a.evm
6. /Users/book/Documents/blockchain/contracts/0x685f775179d1828c80702c52e6e017e48b2166.evm
7. /Applications/Miostream.app/Contents/MacOS/Miostream
8. /Users/book/Documents/blockchain/contracts/0x36e6f214bf7f0b2d00e28f4a51a4c63392c74a.evm
9. /Users/book/Documents/blockchain/contracts/0x7e625c1c155fa26a5c803d8f7488f01a660119b148e.evm
10. /Users/book/Documents/blockchain/contracts/0x03752b021706e73d64e88e1c438b3439627732a1.evm

Open... Open an existing file.
Options... Open an existing file with custom options.
New... Create a new binary file.
Triage... Open files for quick analysis in the Triage Summary view.
Cases & Futures

● To-dos in the future
  ○ Make it more “smart-contract-like” in decompilation, not c-like
  ○ Have a plugin like IDA F.L.I.R.T. Technology
    ■ Fast Library Identification and Recognition Technology
  ○ Best-effort to decode the 4-byte signatures
References

- REVERSE ENGINEERING A CONTRACT
- Decompiler - how to structure loops
- CS153: Compilers Lecture 23: Static Single Assignment Form
- crytic/ethersplay
THANK YOU!

HAVE QUESTIONS?

boik.su@cyccarrier.com