



# Locate Vulnerabilities of Ethereum Smart Contracts with Semi- Automated Analysis

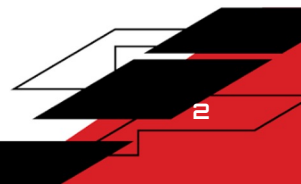
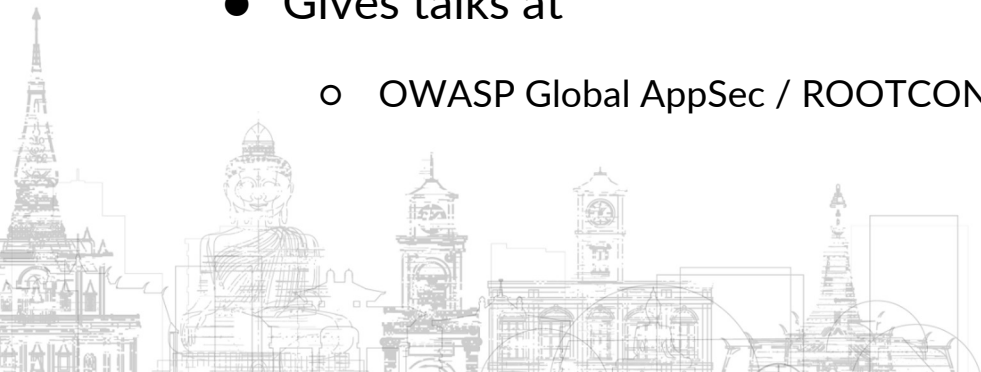
Boik Su

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



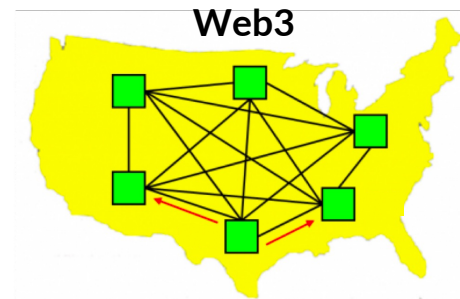
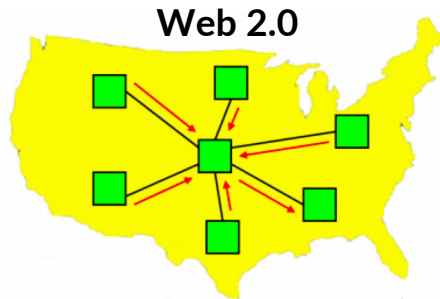
# Distributed Ledger Technology (DLT)

- “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



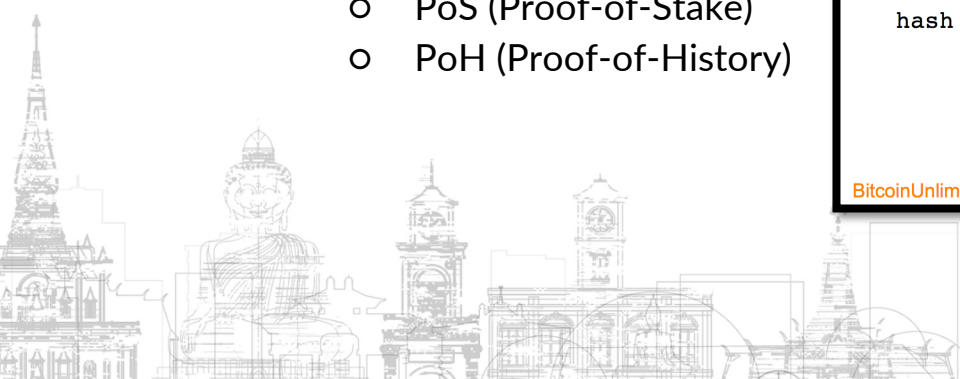
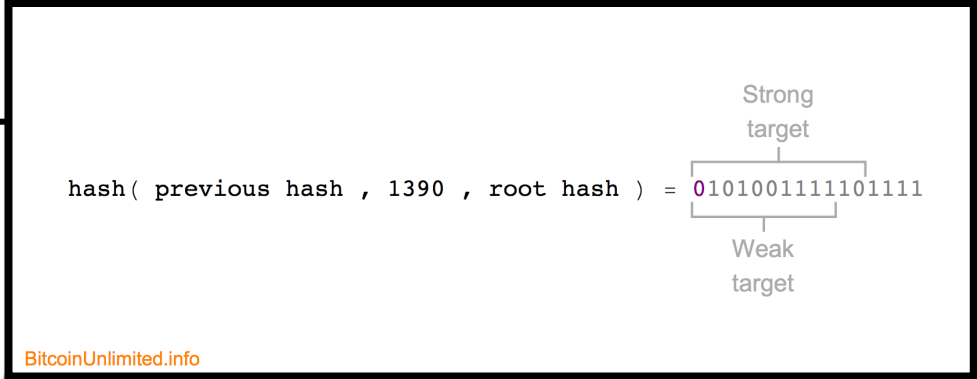
# Consensus Algorithm

- 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)



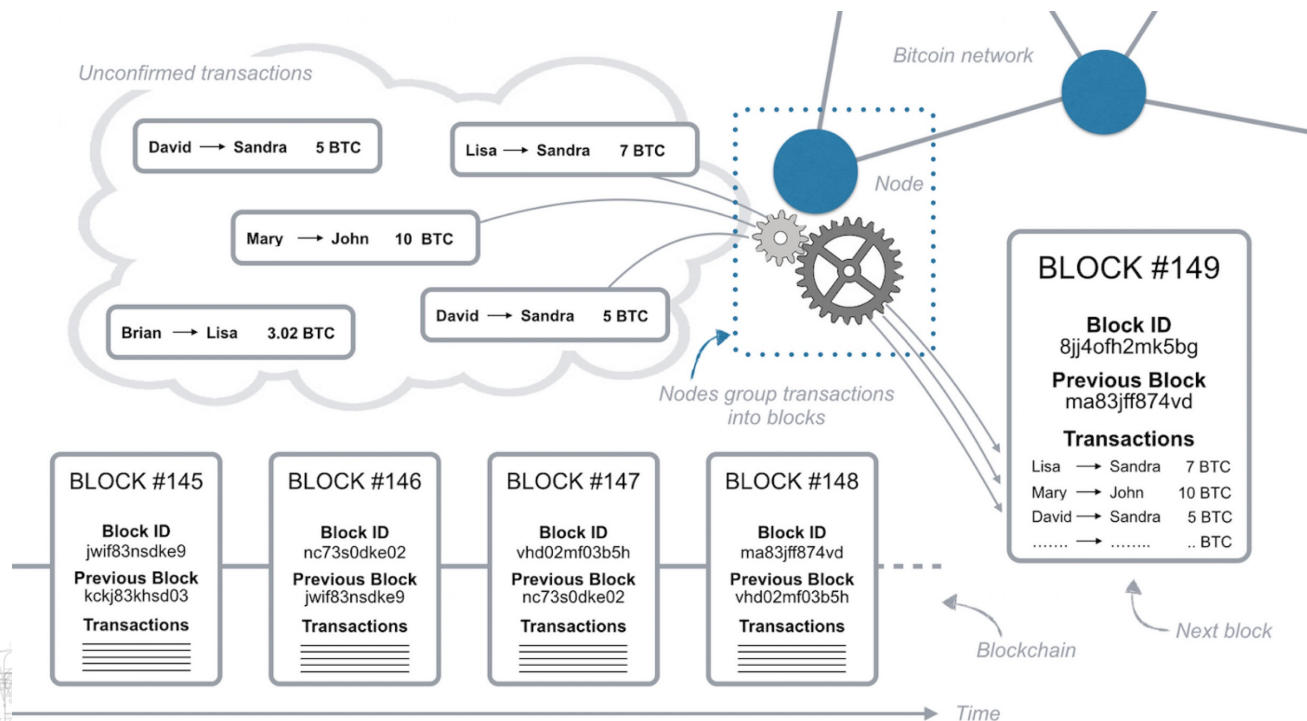
# Consensus Algorithm

- 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)

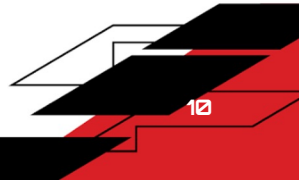
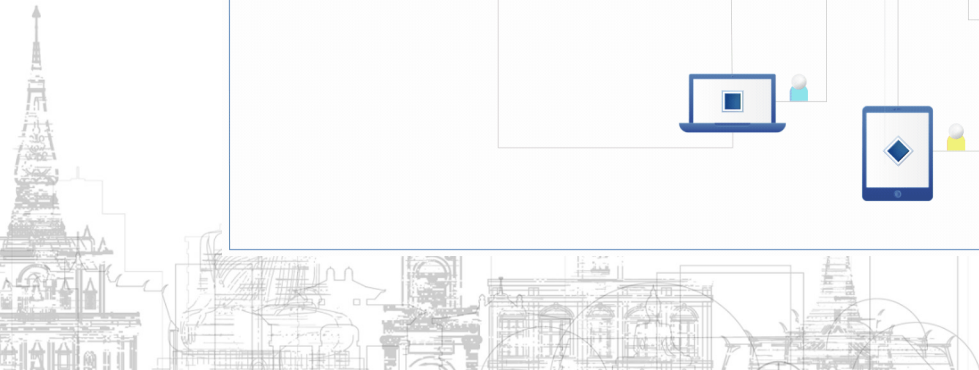
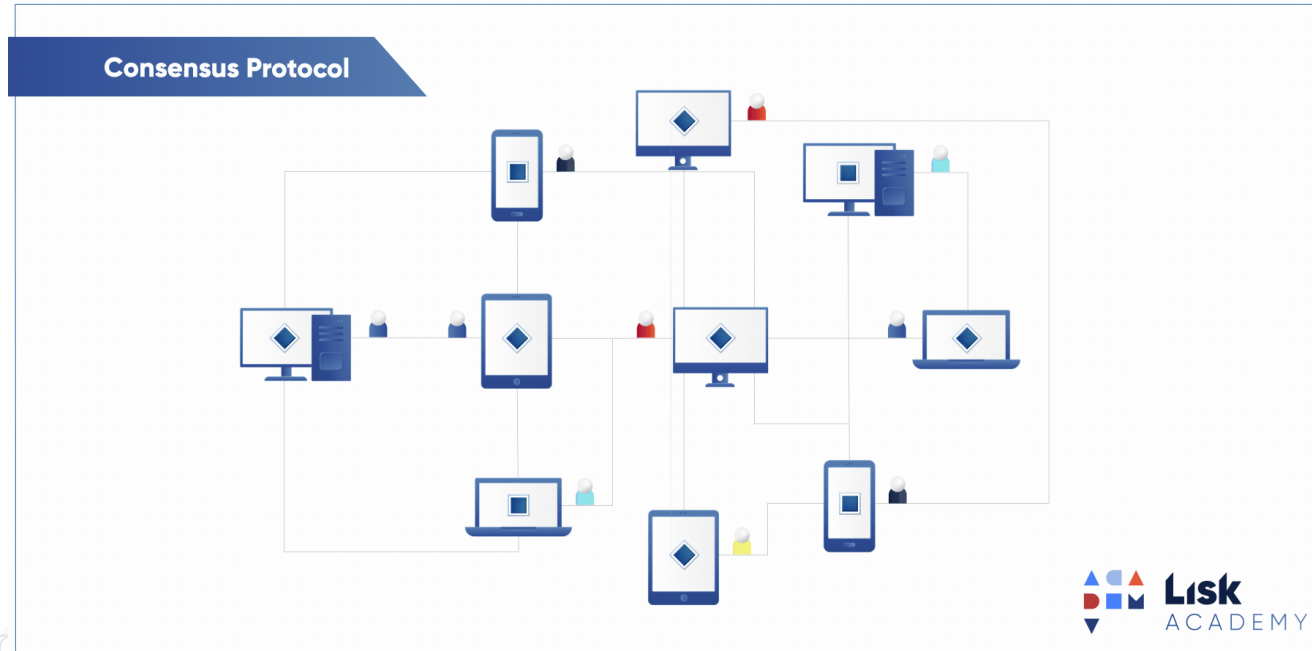




# How does "blockchain" work?



# 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

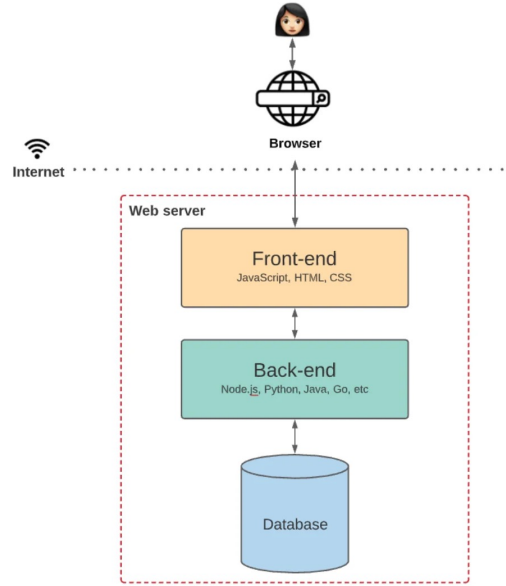


# Ethereum blockchain & Smart Contract

- 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



## Web3

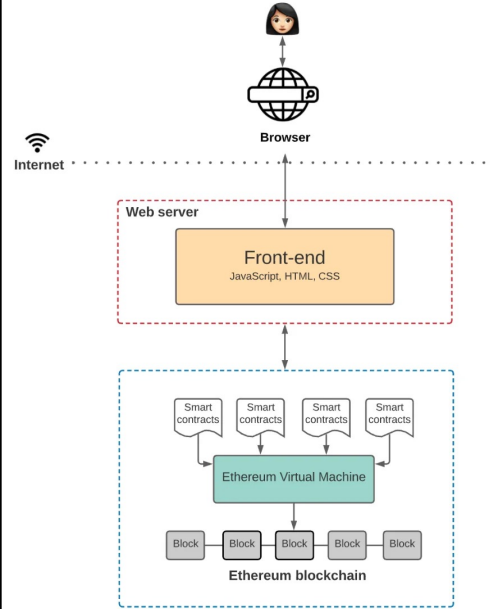


Diagram by Preethi Kasireddy

# 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**

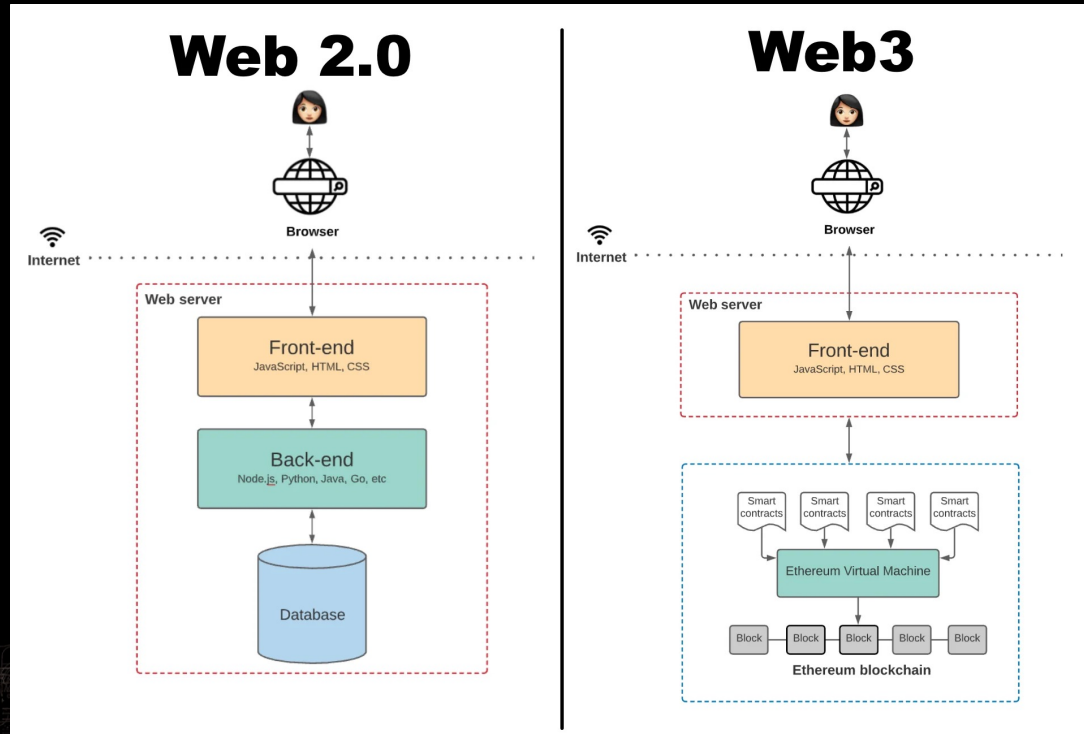


# Outline

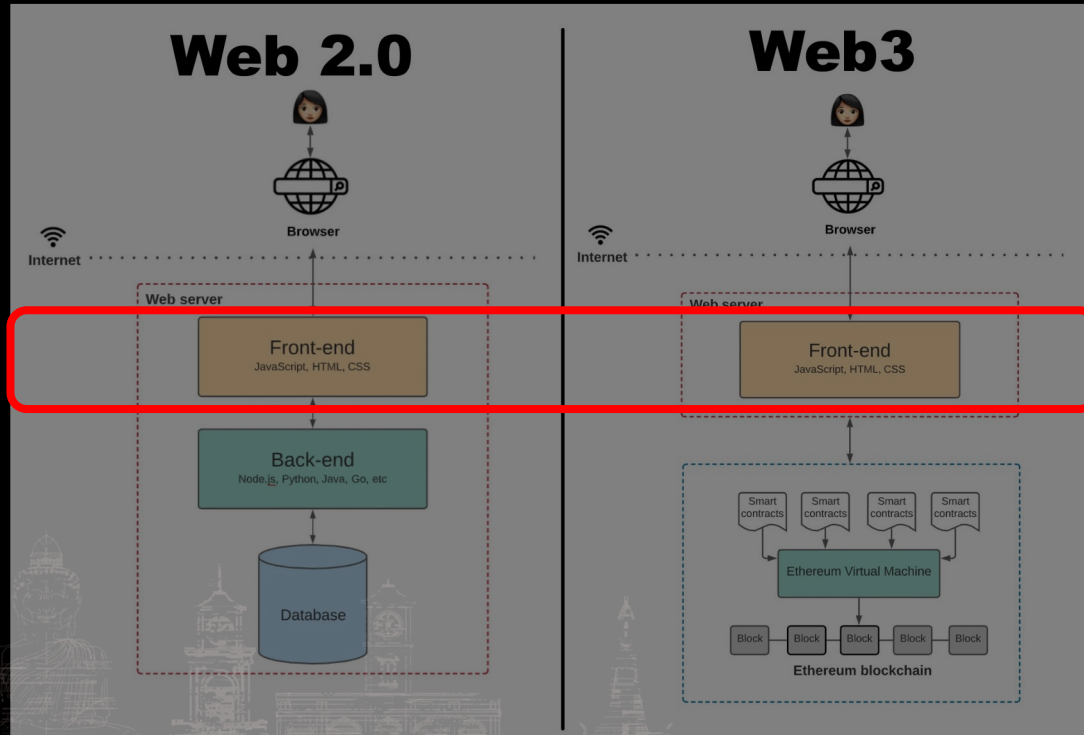
- Intro to Blockchain & Web3
- EVM-based Smart Contract Basics
- Reverse Engineering & CFG
- Cases & Futures



# Hacks in Web3



# Hacks in Web3 (Front-End)



It's Web 2.0 things...

# Hacks in Web3 (Front-End)

4/18/22

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

 **Curve Finance**  
@CurveFinance · Follow

Don't use the frontend yet. Investigating!


 **samczsun is occasionally shitposting** 🗨️ @samczsun  
🚩 🚩 🚩

@CurveFinance frontend is compromised, do not use it until further notice!

4:40 AM · Aug 10, 2022


👍 649    💬 Reply    🔗 Copy link

Read 70 replies




Signing your Request...


Ethereum Mainnet

 0xC9a...a152

New address detected! Click here to add to your address book.

<https://rtfknike.xyz>

 0xC9a...a152 : CLAIM REWARDS ⓘ

 0.11637182 ETH  
\$146.98

DETAILS   DATA   HEX

EDIT

**Estimated gas fee** ⓘ    \$1.34    0.001061 ETH

Site suggested  
Very likely in <15 seconds

**Max fee:** 0.00106077 ETH

Total    \$148.32

# Hacks in Web3 (Front-End)

4/18/22

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

## XSS (Cross-Site Scripting)

**Curve Finance** @CurveFinance · Follow

Don't use the frontend yet. Investigating!

**samczsun is occasionally shitposting** @samczsun

@CurveFinance frontend is compromised, do not use it until further notice!

4:40 AM · Aug 10, 2022

649 Reply Copy link

Read 70 replies

Signing your Request...

Ethereum Mainnet

0xC9a...a1S2

New address detected! Click here to add to your address book.

<https://rtfktnike.xyz>

0xC9a...a1S2 : CLAIM REWARDS

0.11637182 ETH  
\$146.98

DETAILS DATA HEX

EDIT

Estimated gas fee \$1.34 0.001061 ETH

Site suggested Very likely in <15 seconds

Max fees: 0.00106077 ETH

Total \$148.32

# Hacks in Web3 (Front-End)

4/18/22

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

 **Curve Finance**  
@CurveFinance · Follow

Don't use the frontend yet. Investigating!


 **samczsun is occasionally shitposting** 🗨️ @samczsun  
🌟🌟🌟

@CurveFinance frontend is compromised, do not use it until further notice!

4:40 AM · Aug 10, 2022

👍 649    💬 Reply    🔗 Copy link

Read 70 replies

 **DNS Cache Spoofing**


Ethereum Mainnet

Account 1 → 0xC9a...a1S2

New address detected! Click here to add to your address book.

https://rtfktnike.xyz

0xC9a...a1S2 : CLAIM REWARDS

 0.11637182 ETH  
\$146.98

DETAILS    HEX

Estimated gas fee    \$1.34    0.001061 ETH

Site suggested  
Very likely in <15 seconds

Max fee: 0.00106077 ETH

Total    \$148.32

# Hacks in Web3 (Front-End)

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

4/18/22

Curve Finance @CurveFinance · Follow

Don't use the frontend yet. Investigating!

samczsun is occasionally shitposting @samczsun

@CurveFinance frontend is compromised, do not use it until further notice!

4:40 AM · Aug 10, 2022

649 Reply Copy link

Read 70 replies

UI Spoofing

Signing your Request...

Ethereum Mainnet

Account 1 → 0xC9a...a152

New address detected! Click here to add to your address book.

https://rtfknike.xyz

0xC9a...a152 : CLAIM REWARDS

0.11637182 ETH

\$146.98

DETAILS DATA HEX

EDIT

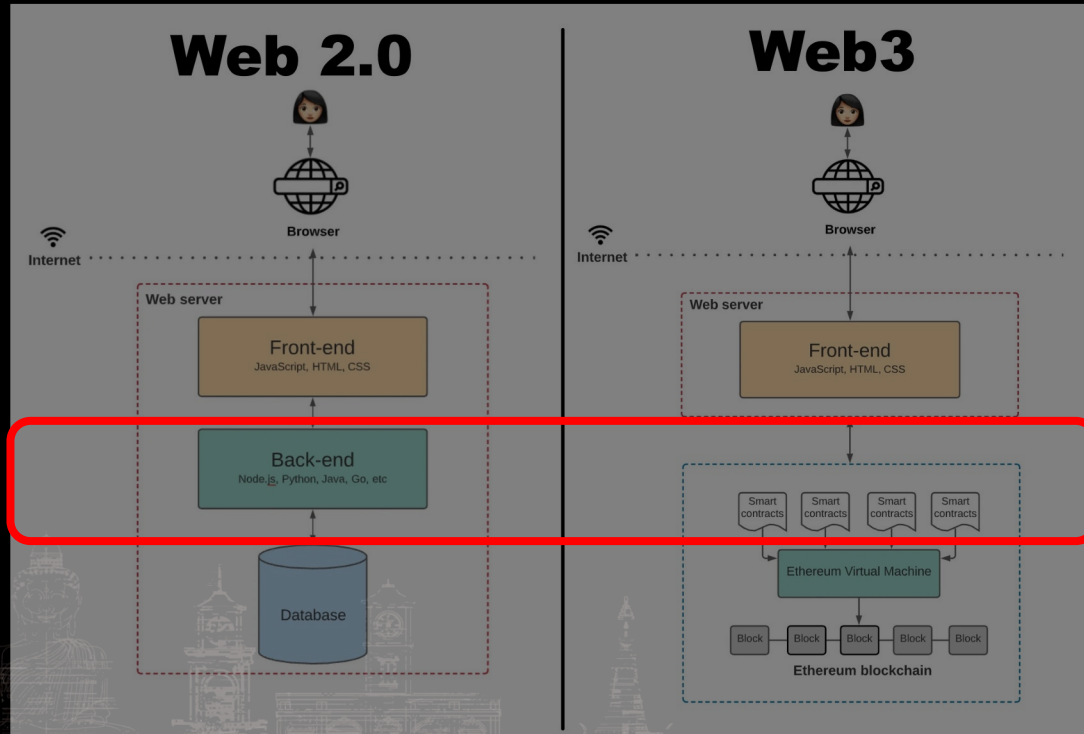
Estimated gas fee \$1.34 0.001061 ETH

Site suggested Very likely in <15 seconds

Max fees 0.00106077 ETH

Total \$148.32

# Hacks in Web3 (Back-End)



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.



**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)

## 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

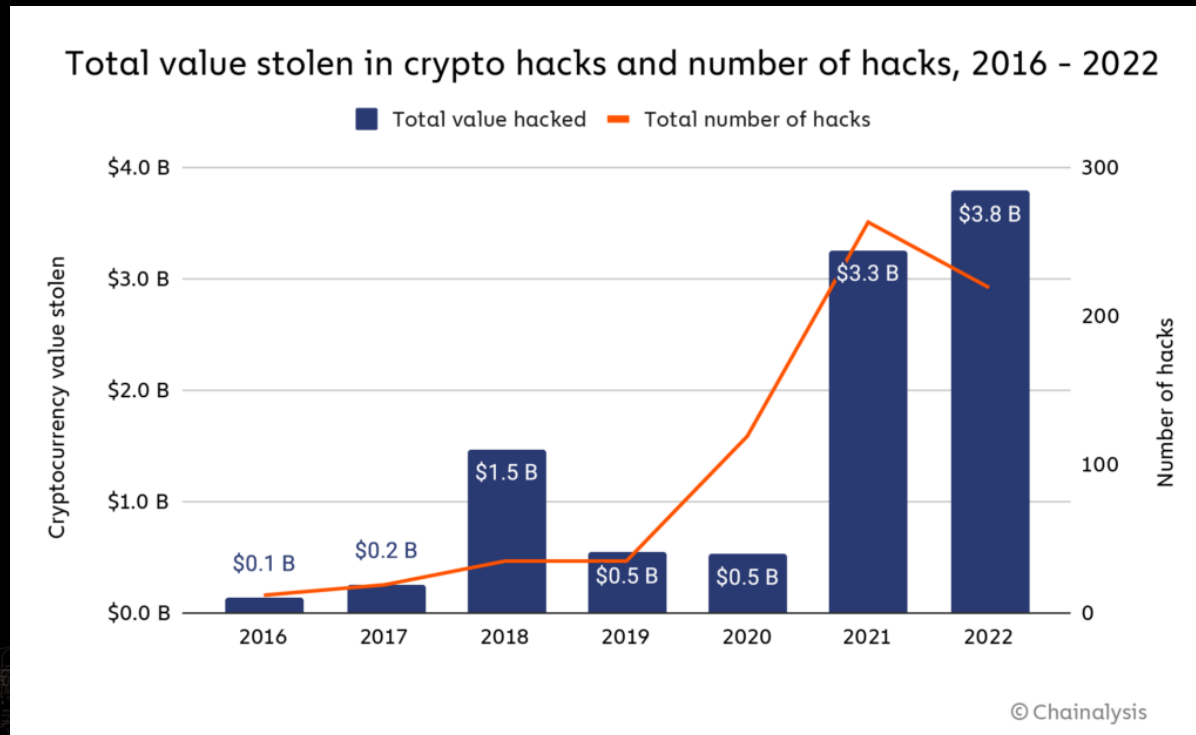
The wormhole network was exploited for **120k wETH**.

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)



# 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

```
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

```
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?

```
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

```
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 Test

- Which means that we have the source code
- We can use this to our advantage to do a static analysis to easily discover
- You can mint as many items as you want, paying only the price of one item

```
uint256 quantity, bytes32[] calldata proof)
    saleConf;
    e sale";
    sender];
    mir";
    acked(msg.sender);
    valid Signature proof supplied.");
    v <= collectionSize. "reached max supply");
    nvalid funds provided");
    quantity);
```

# 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

```
> [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

```
▼ [call][231631] 0x0000007370AF0000aD00Be0EFD
  [call][4786][0x] [Isekai Meta: ISEKAI Toker
  [staticcall][2534][0x] [Wrapped Ether].ba
  ▼ [call][111479][0x] [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

Assembly view	Bytecode view
<pre>PUSH1 e0 PUSH1 02 EXP PUSH1 00 CALLDATALOAD :</pre>	<pre>0x60e060020a600035...</pre>

# CFG (Control Flow Graph)

```

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

Source Code

**B1**

```

w = 0;
x = x + y;
y = 0;
if( x > z)
    
```

**B2**

```

y = x;
x++;
    
```

**B3**

```

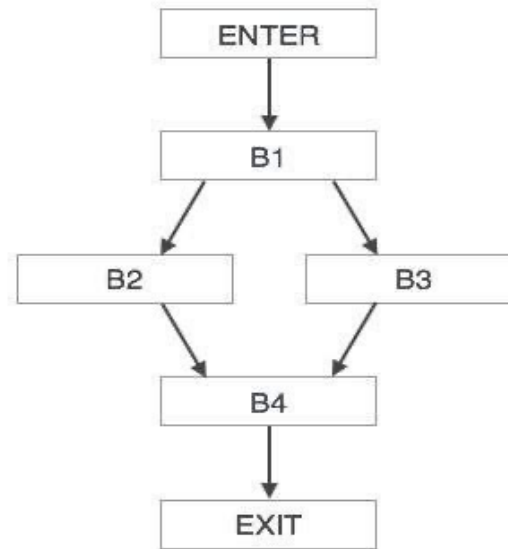
y = z;
z++;
    
```

**B4**

```

w = x + z;
    
```

Basic Blocks



Flow Graph

# CFG (Control Flow Graph)

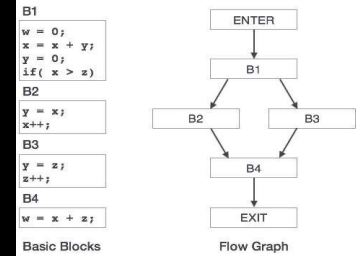
- Why do we need to construct a CFG
  1. To have correct executing logics

```

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

```

Source Code



```

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

```

# CFG (Control Flow Graph)

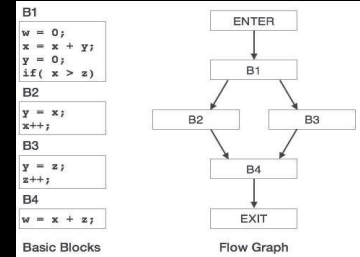
- Why do we need to construct a CFG
  1. To have correct executing logics

```

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

```

Source Code



↓

```

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

```

# CFG (Control Flow Graph)

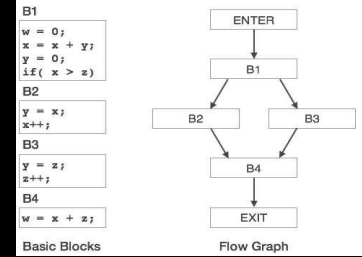
- Why do we need to construct a CFG
  1. To have correct executing logics

```

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

```

Source Code



```

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

```

$$= ECX * EAX$$

$$= ECX * 2$$





# CFG (Control Flow Graph)

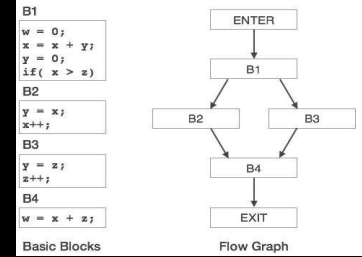
- Why do we need to construct a CFG
  1. To have correct executing logics

```

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

```

Source Code



```

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

```

~~= ECX \* EAX~~  
~~= ECX \* 2~~



# CFG (Control Flow Graph)

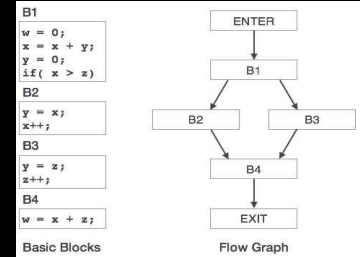
- Why do we need to construct a CFG
  1. To have correct executing logics

```

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

```

Source Code



```

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

```

# CFG (Control Flow Graph)

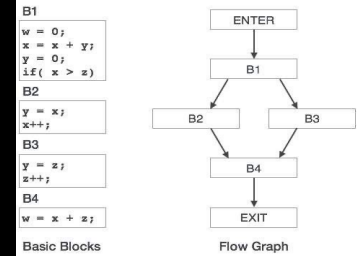

- Why do we need to construct a CFG
  1. To have correct executing logics

```

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

```

Source Code

```

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

```

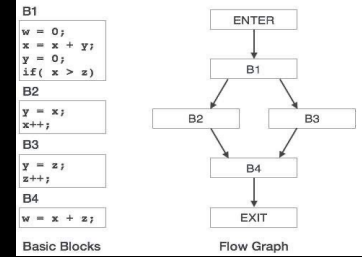
# CFG (Control Flow Graph)

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

```

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

Source Code



```

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

*= ECX \* EAX*  
*= ECX \* (EAX == 0) ? 2 : 3*



# CFG (Control Flow Graph)

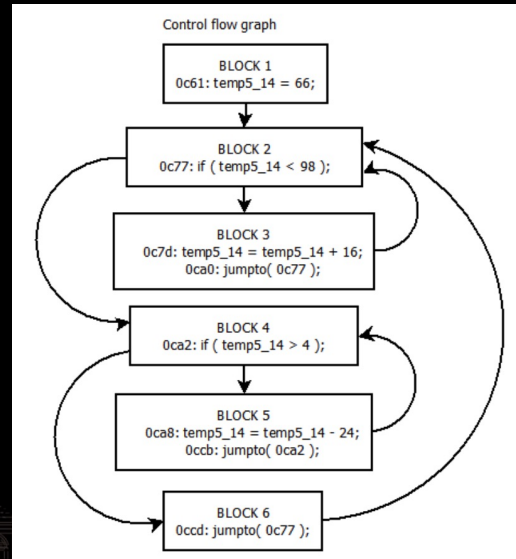
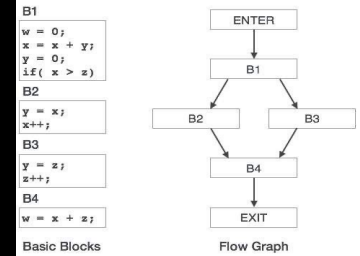
- Why do we need to construct a CFG
  1. To have correct executing logics
  2. To eliminate loops

```

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

```

Source Code



# CFG (Control Flow Graph)

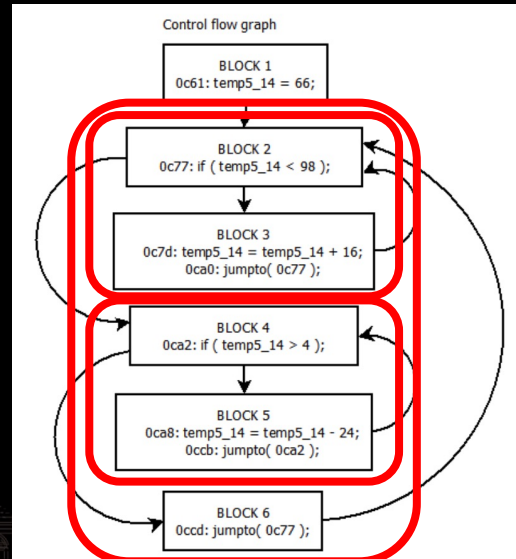
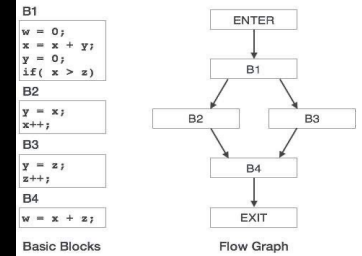
- Why do we need to construct a CFG
  1. To have correct executing logics
  2. To eliminate loops

```

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

```

Source Code



# CFG (Control Flow Graph)

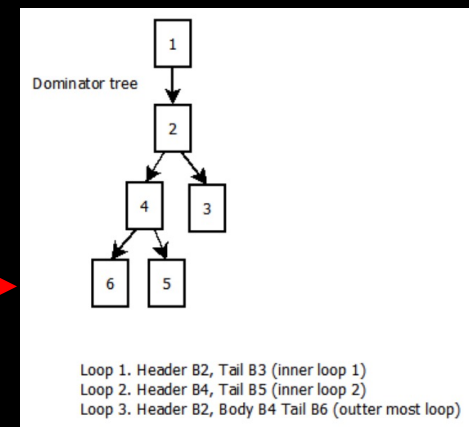
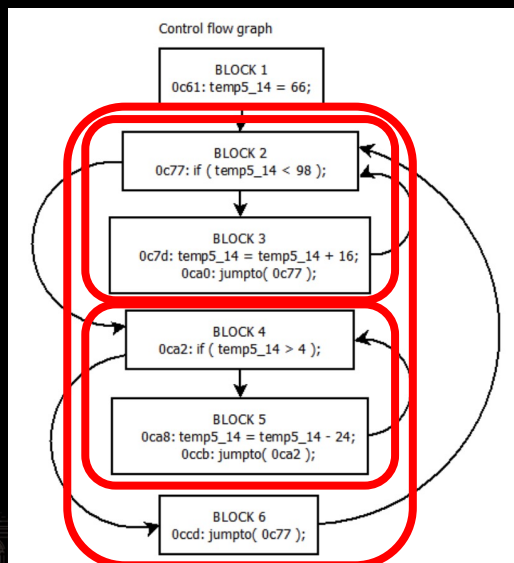
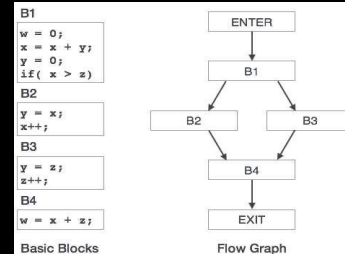
- Why do we need to construct a CFG
  1. To have correct executing logics
  2. To eliminate loops

```

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

```

Source Code



# CFG (Control Flow Graph)

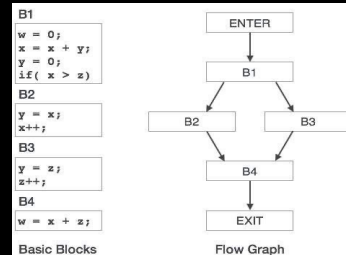
- Why do we need to construct a CFG
  1. To have correct executing logics
  2. To eliminate loops
  3. To transform into SSA form and lift to a higher-level abstraction

```

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

```

Source Code



```

x := n
y := m
x := x + y
return x

```

```

x0 := n
y0 := m
x1 := x0 + y0
return x1

```



# CFG (Control Flow Graph)

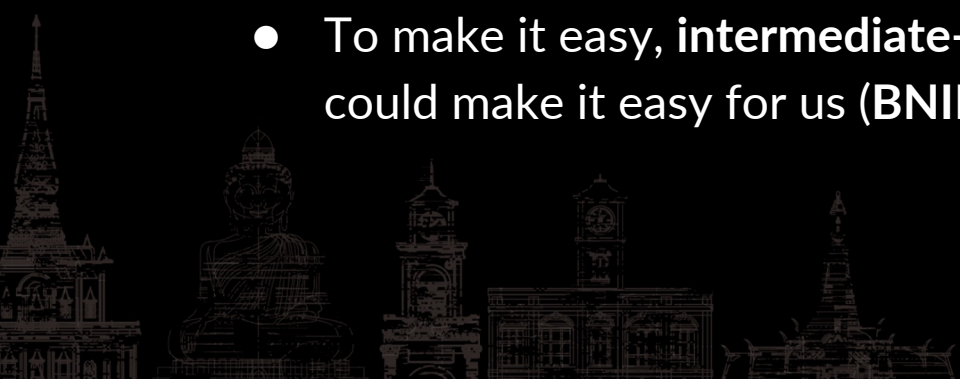
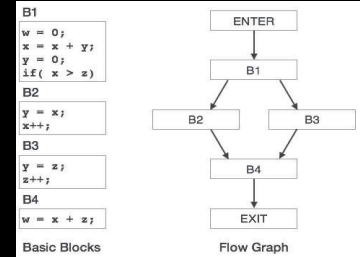
- Why do we need to construct a CFG
  1. To have correct executing logics
  2. To eliminate loops
  3. To transform into SSA form and lift to a higher-level abstraction
- Okay, then how do we get our hands dirty in making a CFG?
- To make it easy, **intermediate-language-based analysis** could make it easy for us (BNIL, P-Code, Microcode, AIL)

```

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

```

Source Code



# CFG (Control Flow Graph)

```

Ethereum Bytecode ▾ Linear ▾ Low Level IL ▾
00000000 int256_t _dispatcher()
⚠ This function has been analyzed with basic analysis only. Enable full analysis of this function.

0 @ 00000000 push(0x430504000000eb)
1 @ 00000021 push(pop == 0)
2 @ 00000022 push(0x45)
3 @ 00000024 temp0.32 = pop
4 @ 00000024 temp0.32 = 0x45
5 @ 00000024 jump(temp0.32 => 6 @ 0x46)

6 @ 00000046 push(0x2c20776f726c640a)
7 @ 00000054 push(0)
8 @ 00000055 temp0.32 = pop
9 @ 00000055 temp0.32 = 0
10 @ 00000055 temp1.32 = pop
11 @ 00000055 temp1.32 = 0x2c20776f726c640a
12 @ 00000055 [temp0.32].32 = temp1.32
13 @ 00000056 push(0xc)
14 @ 00000058 push(0x13)
15 @ 0000005a <return> jump(pop)

00000025 00 00 00-00 00 20 00 01 00 00 00-00 00 00 00 00 00 00 b9-47 00 43 05 b2 0d cd 80 .....G.C.....
00000040 34 0d 93 cd 80 4....

```

# CFG from Binary Ninja

- To guide BN to construct a CFG from an unknown architecture, we firstly need to convert the machine code to the disassembly

EVM Playground GRAYGLACIER Bytecode Current: 3 Total: 21000

```

7f454c46010000000000000000000000004305020003001a0043051a00430504000000eb1560455600
0000000020000100000000000000000000b947004305b20dcd80340d93cd805b6c68656c6c6f
2c20776f726c640a3d52600c6013f3
    
```

[00]	PUSH32	454c46010000000000000000000000004305020003001a0043051a00430504000000eb156045560000000020000100000000000000000000b947004305b20dcd80340d93cd805b6c68656c6c6f2c20776f726c640a3d52600c6013f3
[21]	ISZERO	
[22]	PUSH1	45



```

00000000 int256_t _dispatcher()
⚠ This function has been analyzed with basic analysis only. Enable full analysis of this function.
    
```

00000000	7f454c4601000000...	PUSH32	#454c46010000000000000000000000004305020003001a0043051a00430504000000eb1560455600000020000100000000000000000000b947004305b20dcd80340d93cd805b6c68656c6c6f2c20776f726c640a3d52600c6013f3
00000021	15	ISZERO	
00000022	6045	PUSH1	#45
00000024	56	JUMP	

# CFG from Binary Ninja

- Secondly, we need to tell BN when to branch out, and therefore, BN will construct the CFG for us

# CFG from Binary Ninja

- 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

# CFG from Binary Ninja

- 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

# CFG from Binary Ninja

- 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

# CFG from Binary Ninja

- 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)



# CFG from Binary Ninja

- 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

# CFG from Binary Ninja

Thanks to [EthereumJS Monorepo](#)

```
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 === 'JUMPI' && 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] };
  }
}
```



# CFG from Binary Ninja

[00]	PUSH32	454c46010
[21]	ISZERO	
[22]	PUSH1	45
[24]	JUMP	
[25]	STOP	
[26]	STOP	
[27]	STOP	

STACK

45
0

```

null,
null,
{
  "instructions": [
    {
      "pc": 36,
      "name": "JUMP"
    }
  ],
  "trueBranches": [
    69
  ]
}
]

```

# CFG from Binary Ninja

```
// 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

```

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());
}

```

```

null,
null,
null,
null,
null,
[
  "0x0",
  "0x68656c6c6f2c20776f726c640a"
]

```

```

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 });
}

```

```

null,
null,
null,
null,
null,
"0x0"

```

# CFG from Binary Ninja

- 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

# CFG from Binary Ninja



## Recent

- 1: [/Users/boik/Documents/blockchain/eth/contracts/elf.evm](#)
- 2: [/Users/boik/Documents/blockchain/eth/contracts/0xa019c785322b921a84d086502da0d0bdb999f1a.evm](#)
- 3: [/Users/boik/Documents/blockchain/eth/contracts/0x253ef258563E146f685e60219DA56a6b75178E19.evm](#)
- 4: [/Users/boik/Documents/blockchain/eth/contracts/0xE7145dd6287AE53326347f3A6694fCf2954bcD8A.evm](#)
- 5: [/Users/boik/Documents/blockchain/eth/contracts/0x1278fb63b150e1c9cc478824e589045729321c54.evm](#)
- 6: [/Users/boik/Documents/blockchain/eth/contracts/0x61EB5a27E5f79d182fAFA702c509e017c48821Ed.evm](#)
- 7: [/Applications/Mimestream.app/Contents/MacOS/Mimestream](#)
- 8: [/Users/boik/Documents/blockchain/eth/contracts/0x4d5ad9198f71f23bd002ef8445a1a8cf2932c744.evm](#)
- 9: [/Users/boik/Documents/blockchain/eth/contracts/0x76f2cFc1F5Fa8F6a5b3f4c8f4788f0116861F9B.evm](#)
- 10: [/Users/boik/Documents/blockchain/eth/contracts/0x037520c021706e73aa54d81c14808343962770a1.evm](#)

- Open...** Open an existing file.
- Options...** Open an existing file with custom options.
- New** Create a new binary file.
- Triage** Open file(s) for quick analysis in the Triage Summary view.

## 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 spicy 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

# Outline

- 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

```
contract SagProxy {  
    ...event PrizeRequest(bytes32 msgHash, uint8 v, bytes32 r, bytes32 s);  
    ...event PrizeReady(address winner, bytes prize);  
  
    ...Sag private sag;  
  
    ...address private owner;
```



# Cases & Futures

- SAG? from DEF CON 2018 Quals
  - It gives us a proxy contract to interact with the private contract behind

```

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

    ... Sag private sag;

    ... address private owner;
  
```

```

0003e3d00001d5b505050500040513d0020611015610510570000a
58b0f492ac22be013a619afc0486aaa433dad38d928098dabe8657
260200183600019166000191681526020018260001916600019168
05b9493505050505600a165627a7a72305820563788b2b3c0a0289
00000000000000a019c785322b921a84d086502da0d0dbdb993fba
  
```



# 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)**

```
.....function 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

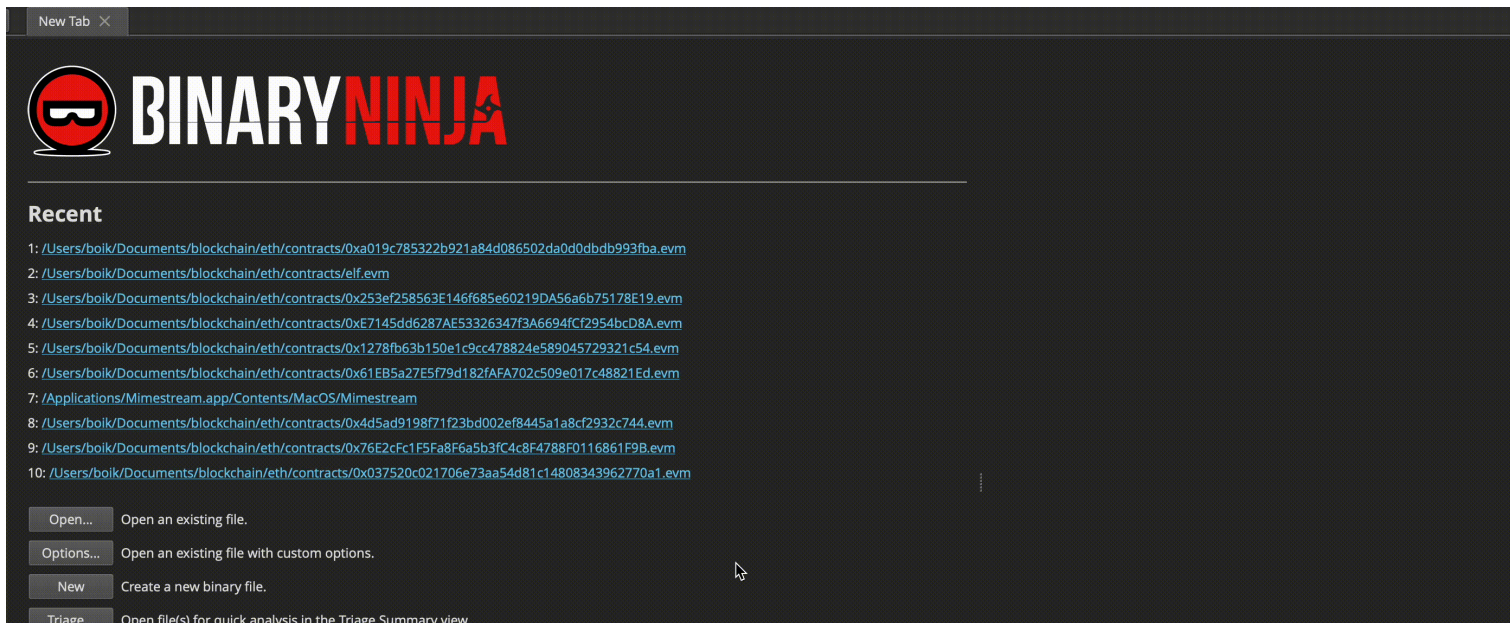
```
function requestPrize(bytes32 msgHash, uint8 v, bytes32 r, bytes32 s) public
    returns (bool is_winner)
{
    if (ecrecover(msgHash, v, r, s) == msg.sender && sag.isWinner(msg.sender)) {
        emit PrizeRequest(msgHash, v, r, s);
        return true;
    }
    return false;
}
```

# 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: **gamble(guess, seed)**
  - Then, we request the prize
  - The Sag contract isn't published and verified, so we reverse it



# Cases & Futures



New Tab ×



## Recent

- 1: [/Users/boik/Documents/blockchain/eth/contracts/0xa019c785322b921a84d086502da0d0dbdb993fba.evm](#)
- 2: [/Users/boik/Documents/blockchain/eth/contracts/elf.evm](#)
- 3: [/Users/boik/Documents/blockchain/eth/contracts/0x253ef258563E146f685e60219DA56a6b75178E19.evm](#)
- 4: [/Users/boik/Documents/blockchain/eth/contracts/0xE7145dd6287AE53326347F3A6694fCf2954bcD8A.evm](#)
- 5: [/Users/boik/Documents/blockchain/eth/contracts/0x1278fb63b150e1c9cc478824e589045729321c54.evm](#)
- 6: [/Users/boik/Documents/blockchain/eth/contracts/0x61EB5a27E5f79d182fAFA702c509e017c48821Ed.evm](#)
- 7: [/Applications/Mimestream.app/Contents/MacOS/Mimestream](#)
- 8: [/Users/boik/Documents/blockchain/eth/contracts/0x4d5ad9198f71f23bd002ef8445a1a8cf2932c744.evm](#)
- 9: [/Users/boik/Documents/blockchain/eth/contracts/0x76E2cFc1F5Fa8F6a5b3fC4c8f4788F0116861F9B.evm](#)
- 10: [/Users/boik/Documents/blockchain/eth/contracts/0x037520c021706e73aa54d81c14808343962770a1.evm](#)

Open... Open an existing file.

Options... Open an existing file with custom options.

New Create a new binary file.

Triage... Open file(s) 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](#)
- [cryptic/ethersplay](#)





Red geometric shapes, including overlapping rectangles and lines, are located in the top-left corner of the slide.

THANK  
YOU!

HAVE  
QUESTIONS?

[boik.su@cycarrier.com](mailto:boik.su@cycarrier.com)

