Erays: Reverse Engineering Ethereum’s Opaque Smart Contracts

Yi Zhou, Deepak Kumar, Surya Bakshi, Joshua Mason, Andrew Miller, Michael Bailey

University of Illinois Urbana-Champaign
Introduction:

Ethereum
Introduction:

Ethereum Smart Contracts

- Computer programs on the blockchain
- Written in high level language (Solidity)
- Executed in the Ethereum Virtual Machine (EVM)
contract dummy {
    uint s;

    function foo(uint a) public returns (uint) {
        while (a < s) {
            if (a > 10) {
                a += 1;
            } else {
                a += 2;
            }
        }
        return a;
    }
}
Compiled Contract

6080604052600043610603e5763fffffffff7c010000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
575b600080fd5b348015604e57600080fd5b506058600435606a565b60408
051918252519081900360200190f35b60005b600054821015609357600a82
1115608857600182019150608f565b6002820191505b606d565b50905600a
165627a7a7230582095826fc9f61669f3d0fe36966d60c64042dec36a23ac
89e6b4ebe1752f2c7ca00029
PUSH1 0x80
PUSH1 0x40
MSTORE
PUSH1 0x04
CALLDATASIZE
LT
PUSH1 0x3e
JUMPI
PUSH4 0xffffffff
PUSH29
0x0100000000000000000000000000000000000000000000000000000000
PUSH1 0x00
CALLDATALOAD
...
Problem:
Opaque/proprietary contracts

- EVM bytecode is not easily understandable
- High level source code is not always available
- Contract functionality remains opaque/proprietary
Ecosystem:

How many contracts are there?

- Total Count: 1,024,886
- Unique Count: 34,328
Ecosystem:

How many contracts are opaque/proprietary?

- 10,387 Solidity Source Files Collected (from Etherscan)
- 35 Versions (v0.1.3 to v0.4.19) of Solidity Compilers Used
- 88,426 Unique Binaries Compiled
## Ecosystem: Measuring Opacity

<table>
<thead>
<tr>
<th></th>
<th>Contracts</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Total</strong></td>
<td>1,024,886</td>
</tr>
<tr>
<td><strong>Unique</strong></td>
<td>34,328 (100.0%)</td>
</tr>
<tr>
<td><strong>Unique Transparent</strong></td>
<td>7,734 (22.5%)</td>
</tr>
<tr>
<td><strong>Unique Opaque</strong></td>
<td>26,594 (77.5%)</td>
</tr>
</tbody>
</table>
## Ecosystem: Measuring Opacity

<table>
<thead>
<tr>
<th></th>
<th>Contracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>1,024,886</td>
</tr>
<tr>
<td>Unique</td>
<td>34,328 (100.0%)</td>
</tr>
<tr>
<td>Unique Transparent</td>
<td>7,734 (22.5%)</td>
</tr>
<tr>
<td>Unique Opaque</td>
<td>26,594 (77.5%)</td>
</tr>
</tbody>
</table>
Erays
Erays: System Design

1. Control Flow Graph Recovery
2. Lifting
3. Optimization
4. Aggregation
5. Control Flow Structure Recovery
Control Flow Graph Recovery

- Identify basic block boundaries

```
...  
JUMPDEST
PUSH1 0x0
JUMPDEST
PUSH1 0x0
SLOAD
DUP3
LT
ISZERO
PUSH1 0x93
JUMPI
...  
```
Control Flow Graph Recovery

- Identify basic block boundaries
Control Flow Graph Recovery

- Identify basic block boundaries
- Organize basic blocks into a CFG
  - Emulate the contract using a stack model
  - Explore the contract in a manner similar to Depth First Search
  - Record stack images at each block entrance
Control Flow Graph Recovery
Control Flow Graph Recovery
Control Flow Graph Recovery
Control Flow Graph Recovery
Control Flow Graph Recovery
Control Flow Graph Recovery
Control Flow Graph Recovery
Control Flow Graph Recovery
Lifting: Stack-based to Register-based

- Convert stack-based operations into register-based representation (R. Vallee-Rai 1999)
  - Map stack slots to registers

```
$s0
$s1
$s2
...
```
Lifting: Stack-based to Register-based

- Convert stack-based operations into register-based representation (R. Vallee-Rai 1999)
  - Map stack slots to registers
  - Assign registers to each bytecode (using stack height)

```
ADD
$s2  0x2
$s1  0x3
$s0  0xb2
```
Lifting: Stack-based to Register-based

- Convert stack-based operations into register-based representation (R. Vallee-Rai 1999)
  - Map stack slots to registers
  - Assign registers to each bytecode (using stack height)

```
ADD
$s2 0x2
$s1 0x3
$s0 0xb2
```
Lifting: Stack-based to Register-based

- Convert stack-based operations into register-based representation (R. Vallee-Rai 1999)
  - Map stack slots to registers
  - Assign registers to each bytecode (using stack height)

```
ADD
$s2  0x2 + 0x3
$s1
$s0  0xb2
```
Lifting: Stack-based to Register-based

- Convert stack-based operations into register-based representation (R. Vallee-Rai 1999)
  - Map stack slots to registers
  - Assign registers to each bytecode (using stack height)

```
ADD
$s2 0x5
$s1
$s0 0xb2
```
Lifting: Stack-based to Register-based

- Convert stack-based operations into register-based representation (R. Vallee-Rai 1999)
  - Map stack slots to registers
  - Assign registers to each bytecode (using stack height)

```
ADD $s2 $s1 0x5 $s0 0xb2
```
Lifting: Stack-based to Register-based

- Convert stack-based operations into register-based representation (R. Vallee-Rai 1999)
  - Map stack slots to registers
  - Assign registers to each bytecode (using stack height)

```
ADD $s1, $s2, $s1

$s2
$s1 0x5
$s0 0xb2
```
Lifting: Stack-based to Register-based

- Convert stack-based operations into register-based representation (R. Vallee-Rai 1999)
  - Map stack slots to registers
  - Assign registers to each bytecode (using stack height)
Lifting: Stack-based to Register-based

- Convert stack-based operations into register-based representation (R. Vallee-Rai 1999)
  - Map stack slots to registers
  - Assign registers to each bytecode (using stack height)

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVE</td>
<td>$s4, 0x0</td>
</tr>
<tr>
<td>SLOAD</td>
<td>$s4, [$s4]</td>
</tr>
<tr>
<td>MOVE</td>
<td>$s5, $s2</td>
</tr>
<tr>
<td>LT</td>
<td>$s4, $s5, $s4</td>
</tr>
<tr>
<td>ISZERO</td>
<td>$s4, $s4</td>
</tr>
<tr>
<td>MOVE</td>
<td>$s5, 0x93</td>
</tr>
<tr>
<td>JUMPI</td>
<td>$s5, $s4</td>
</tr>
</tbody>
</table>
Lifting: Stack-based to Register-based

- Convert stack-based operations into register-based representation (R. Vallee-Rai 1999)
- Introduce new instructions
Lifting: Stack-based to Register-based

- Convert stack-based operations into register-based representation (R. Vallee-Rai 1999)
- Introduce new instructions
  - INTCALL, INTRET
  - MOVE
  - ASSERT
  - NEQ, GEQ, LEQ, SL, SR
Optimization: Removing Redundancy

- Global optimizations (1973 G. Kildall)

<table>
<thead>
<tr>
<th>Instruction</th>
<th><code>$s4</code></th>
<th><code>$s5</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVE</td>
<td>0x0</td>
<td></td>
</tr>
<tr>
<td>SLOAD</td>
<td>[<code>${s4}</code>]</td>
<td></td>
</tr>
<tr>
<td>MOVE</td>
<td><code>$s2</code></td>
<td></td>
</tr>
<tr>
<td>LT</td>
<td><code>$s5</code>, <code>$s4</code></td>
<td></td>
</tr>
<tr>
<td>ISZERO</td>
<td><code>$s4</code></td>
<td></td>
</tr>
<tr>
<td>MOVE</td>
<td>0x93</td>
<td></td>
</tr>
<tr>
<td>JUMPI</td>
<td><code>$s4</code></td>
<td></td>
</tr>
</tbody>
</table>
Optimization: Removing Redundancy

- Global optimizations (1973 G. Kildall)
  - Constant propagation

```plaintext
MOVE $s4, 0x0
SLOAD $s4, [0x0]
MOVE $s5, $s2
LT $s4, $s5, $s4
ISZERO $s4, $s4
MOVE $s5, 0x93
JUMPI 0x93, $s4
```
Global optimizations (1973 G. Kildall)

- Constant propagation
- Copy propagation

Optimization: Removing Redundancy

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Destination</th>
<th>Source 1</th>
<th>Source 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVE</td>
<td>$s4, 0x0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SLOAD</td>
<td>$s4, [0x0]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOVE</td>
<td>$s5, $s2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LT</td>
<td>$s4, $s2, $s4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ISZERO</td>
<td>$s4, $s4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOVE</td>
<td>$s5, 0x93</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JUMPI</td>
<td>0x93, $s4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Optimization: Removing Redundancy

- Global optimizations (1973 G. Kildall)
  - Constant propagation
  - Copy propagation
  - Dead code elimination

```
--
SLOAD $s4, [0x0]
--
LT $s4, $s2, $s4
ISZERO $s4, $s4
--
JUMPI 0x93, $s4
```
### Optimization: Removing Redundancy

- **Global optimizations (1973 G. Kildall)**
  - Constant propagation
  - Copy propagation
  - Dead code elimination

- **Local optimizations**

```
  --
  SLOAD     $s4, [0x0]
  --
  LT        $s4, $s2, $s4
  ISZERO    $s4, $s4
  --
  JUMPI     0x93, $s4
```
Optimization: Removing Redundancy

- Global optimizations (1973 G. Kildall)
  - Constant propagation
  - Copy propagation
  - Dead code elimination
- Local optimizations

```
--
SLOAD $s4, [0x0]
--
--
GEQ $s4, $s2, $s4
--
JUMPI 0x93, $s4
```
Optimization: Removing Redundancy

- Global optimizations (1973 G. Kildall)
  - Constant propagation
  - Copy propagation
  - Dead code elimination
- Local optimizations

```
SLOAD     $s4, [0x0]
GEQ       $s4, $s2, $s4
JUMPI     0x93, $s4
```
Aggregation: Condensing the Output

- Convert register-based instructions into three address form

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Operands</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLOAD</td>
<td>$s4, [0x0]</td>
</tr>
<tr>
<td>GEQ</td>
<td>$s4, $s2, $s4</td>
</tr>
<tr>
<td>JUMPI</td>
<td>0x93, $s4</td>
</tr>
</tbody>
</table>
Aggregation: Condensing the Output

- Convert register-based instructions into three address form

$s4 = S[0x0]$
$s4 = s2 \geq s4$
if ($s4$) goto 0x93
Aggregation: Condensing the Output

- Convert register-based instructions into three address form
- Aggregate instructions into nested expressions (R. Vallee-Rai 1999)

```
$s4 = S[0x0]
$s4 = $s2 ≥ $s4
if ($s4) goto 0x93
```
Aggregation: Condensing the Output

- Convert register-based instructions into three address form
- Aggregate instructions into nested expressions (R. Vallee-Rai 1999)

```plaintext
--
$s4 = $s2 ≥ S[0x0]
if ($s4) goto 0x93
```
Aggregation: Condensing the Output

- Convert register-based instructions into three address form
- Aggregate instructions into nested expressions (R. Vallee-Rai 1999)

```
--
--
if ($s2 \geq S[0x0]) goto 0x93
```
Aggregation: Condensing the Output

- Convert register-based instructions into three address form
- Aggregate instructions into nested expressions (R. Vallee-Rai 1999)

```plaintext
if ($s2 \geq S[0x0]) \text{ goto } 0x93
```
Control Flow Structure Recovery

- Separate each public function subgraph
- Use structural analysis (M. Sharir 1980)
  - Match subgraphs to control constructs (while, if then else)
  - Collapse matched subgraphs
CONTROL_FLOW_STRUCTURES

CONTROL_FLOW_STRUCTURE_RECOVERY

ASSERT(0 == msg.value)
$s2 = C[0x4]

if ($s2 <= 0xa) goto 0x88
$s2 = 0x1 + $s2
goto 0x8f

if ($s2 >= 0x0) goto 0x93

M[$m] = $s2
RETURN($m, 0x20)

$s2 = 0x2 + $s2
goto 0x6d
ASSERT(0 == msg.value)
$s2 = C[0x4]$

if ($s2 <= 0xa) {
    $s2 = 0x2 + $s2
} else {
    $s2 = 0x1 + $s2
}

if ($s2 >= S[0x0]) goto 0x93

$\textbf{Control Flow Structure Recovery}$

M[$m$] = $s2$
RETURN($m$, 0x20)

goto 0x6d
ASSERT(0 == msg.value)
$s2 = C[0x4]$

if ($s2 <= 0xa) {
    $s2 = 0x2 + $s2
} else {
    $s2 = 0x1 + $s2
}
goto 0x6d

if ($s2 >= S[0x0]) goto 0x93

M[$m] = $s2
RETURN($m, 0x20)
ASSERT(0 == msg.value)
$s2 = C[0x4]

while (0x1) {
    if ($s2 >= S[0x0])
        break
    if ($s2 <= 0xa) {
        $s2 = 0x2 + $s2
    } else {
        $s2 = 0x1 + $s2
    }
}

M[$m] = $s2
RETURN($m, 0x20)
ASSERT(0 == msg.value)
$s2 = C[0x4]
while (0x1) {
    if ($s2 >= S[0x0])
        break
    if ($s2 <= 0xa) {
        $s2 = 0x2 + $s2
    } else {
        $s2 = 0x1 + $s2
    }
}
M[$m] = $s2
RETURN($m, 0x20)
Validation

- Construct test cases using historical transactions
- Leverage Geth to generate the expected transaction output
- “Execute” our representation and compare the output
Validation

- Construct test cases using historical transactions
- Leverage Geth to generate the expected transaction output
- “Execute” our representation and compare the output

<table>
<thead>
<tr>
<th>Transactions</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>15,855 (100.0 %)</td>
</tr>
</tbody>
</table>
Validation

- Construct test cases using historical transactions
- Leverage Geth to generate the expected transaction output
- “Execute” our representation and compare the output

<table>
<thead>
<tr>
<th>Transactions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>15,855 (100.0 %)</td>
</tr>
<tr>
<td>Success</td>
<td>15,345 (96.8%)</td>
</tr>
</tbody>
</table>
Validation

- Construct test cases using historical transactions
- Leverage Geth to generate the expected transaction output
- “Execute” our representation and compare the output

<table>
<thead>
<tr>
<th>Transactions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>15,855 (100.0 %)</td>
</tr>
<tr>
<td>Success</td>
<td>15,345 (96.8%)</td>
</tr>
<tr>
<td>Failures</td>
<td>510 (3.2%)</td>
</tr>
</tbody>
</table>
Validation

- Construct test cases using historical transactions
- Leverage Geth to generate the expected transaction output
- “Execute” our representation and compare the output

<table>
<thead>
<tr>
<th></th>
<th>Transactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>15,855 (100.0 %)</td>
</tr>
<tr>
<td>Success</td>
<td>15,345 (96.8%)</td>
</tr>
<tr>
<td>Failures</td>
<td>510 (3.2%)</td>
</tr>
<tr>
<td>Construction Failures</td>
<td>196 (1.2%)</td>
</tr>
</tbody>
</table>
- Construct test cases using historical transactions
- Leverage Geth to generate the expected transaction output
- “Execute” our representation and compare the output

<table>
<thead>
<tr>
<th></th>
<th>Transactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>15,855 (100.0 %)</td>
</tr>
<tr>
<td>Success</td>
<td>15,345 (96.8%)</td>
</tr>
<tr>
<td>Failures</td>
<td>510 (3.2%)</td>
</tr>
<tr>
<td>Construction Failures</td>
<td>196 (1.2%)</td>
</tr>
<tr>
<td>Comparison Failures</td>
<td>314 (2.0%)</td>
</tr>
</tbody>
</table>
Use Case
Erays: Function Fuzzy Hash

Binary X

Function A

Function B

Function C
Erays: Function Fuzzy Hash

Binary X

- Function A
- Function B
- Function C

Hash A
0x746f7563...

Compute Fuzzy Hash
Erays: Function Fuzzy Hash

Binary X
- Function A
- Function B
- Function C

Hash A
0x746f7563...

Hash B
0x6865646d...

Hash C
0x79737061...
Erays: Code Sharing

Binary X
- Function A
- Function B
- Function C

Hash A
0x746f7563...

Hash B
0x6865646d...

Hash C
0x79737061...

Hash D
0x67686574...

Binary Y
- Function B
- Function D
Case Studies
Case Study: High Value Contracts

- Look for opaque contracts with large Ether balance ~ $590M
- **Multi-signature** wallets likely used by the *Gemini* exchange

**Multi-Signature Wallet**: signature scheme requiring k-of-N signatures.
- Security best practice for large sums of money
Case Study: High Value Contracts

- Look for opaque contracts with large Ether balance ~ $590M / 3 contracts
- **Multi-signature** wallets likely used by the Gemini exchange
- Interesting, time-dependent withdrawal policies

**Multi-Signature Wallet**: signature scheme requiring k-of-N signatures.
- Security best practice for large sums of money
Time Dependency Hazard

- Found `block.timestamp` used in contract
- Erays reveals it is used to control the delay of withdrawal requests
- Useful auditing tool, even for opaque contracts

```solidity
$s10 = sha3(0x0, 0x40)
$s8 = $s10
$s10 = (ad_mask & $s3) | (0xffffffff)
$s4 = $s01 + $s10
$s7 = $s02 + $s10
$s9 = block.timestamp
$s9 = $s03 + $s10
if (msg.sender == ad_mask & s[0x0]){
    $s9 = s[0x1] + $s9
```
Case Study: Duplicate Contracts

- Look for opaque contracts with the most instances
- Exchange user wallets
  - Poloniex: ~350,000 contracts
  - Yunbi: ~90,000 contracts
- A different approach to handling user funds
Case Study: EtherDelta Arbitrage

- Decentralized token exchanges (DEX) operate entirely on-chain
  - Etherdelta
Case Study: EtherDelta Arbitrage

- Decentralized token exchanges (DEX) operate entirely on-chain
  - EtherDelta
- Evidence of arbitrageurs

Order Book:

<table>
<thead>
<tr>
<th>PPT/ETH</th>
<th>PPT</th>
<th>ETH</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.01000</td>
<td>6000.000</td>
<td>60.000</td>
</tr>
<tr>
<td>0.01000</td>
<td>100.000</td>
<td>1.000</td>
</tr>
<tr>
<td>0.00999</td>
<td>50.000</td>
<td>0.499</td>
</tr>
<tr>
<td>0.00998</td>
<td>100.000</td>
<td>0.998</td>
</tr>
</tbody>
</table>

Arbitrageur Behavior:

1. Buy @0.009
2. Sell @0.01
Case Study: EtherDelta Arbitrage

- Decentralized token exchanges (DEX) operate entirely on-chain
  - Etherdelta
- Evidence of arbitrageurs
- Executing a buy/sell mismatch for a profit
Case Study: EtherDelta Arbitrage Bots

- Arbitrageurs must publish *gadgets* to facilitate arbitrage
- Create functions to validate the order and new trade
- Implement atomic batch trades (or fail)
Case Study: CryptoKitties
Founder Cat #6
Kitty #6 · Gen 0
Fast

Lucky 7 | Founder Cat #7
Kitty #7 · Gen 0
Fast

For sale: €197.65
For sale: €350.12
For sale: €75.329
For sale: €75.247
Case Study: CryptoKitties

- On-chain game code is published with source code
- Game mechanism well understood

```solidity
// Call the sooper-sekret gene mixing operation.
uint256 childGenes = geneScience.mixGenes(matron.genes, sire.genes, matron.cooldownEndBlock - 1);
```
Case Study: CryptoKitties

- Developers who know the algorithm aren’t allowed to play the game!
Case Study: CryptoKitties

- Developers who know the algorithm aren’t allowed to play the game!
- So obviously we had to target this function
Case Study: CryptoKitties

- The block hash is used to inject random mutations into genes and to select a parent for a gene.
Case Study: CryptoKitties

- The block hash is used to inject random mutations into genes and to select a parent for a gene.
- Found a more effective breeding strategy.
Case Study: CryptoKitties

- The block hash is used to inject random mutations into genes and to select a parent for a gene.
- Found a more effective breeding strategy.
- Don’t rely on security through obscurity!

```c
// Call the sooper-sekret gene mixing operation.
uint256 childGenes = geneScience.mixGenes(matron.genes, sire.genes, matron.cooldownEndBlock - 1);
```
Conclusion

- Ethereum smart contract ecosystem is largely opaque
  - ~1M contracts, 34K unique, 77.5% unique opaque
Conclusion

- Ethereum smart contract ecosystem is largely opaque
  - ~1M contracts, 34K unique, 77.5% unique opaque

- Erays converts EVM bytecode into higher level representations
  - https://github.com/teamnsrg/erays
  - yizhou7@illinois.edu
Conclusion

- Ethereum smart contract ecosystem is largely opaque
  - ~1M contracts, 34K unique, 77.5% unique opaque

- Erays converts EVM bytecode into higher level representations
  - [https://github.com/teamnsrg/erays](https://github.com/teamnsrg/erays)
  - yizhou7@illinois.edu

- The utility of Erays is demonstrated in several case studies
  - High value wallets, exchange user wallets, arbitrage bots, CryptoKitties secret algorithm