MadMax
Surviving Out-of-Gas Conditions in Ethereum Smart Contracts

Neville Grech (Athens/Malta)
Michael Kong (Sydney)
Anton Jurisevic (Sydney)
Lexi Brent (Sydney)
Bernhard Scholz (Sydney)
Yannis Smaragdakis (Athens)

nevillegrech.com/2018/madmax.html
Terminology

Smart Contracts
• Programs running on the Ethereum Blockchain (usually transacting $$$)

Solidity
• The high-level language for writing them

Gas
• Fee paid for running them
• Earned by the miner & bounded/hard coded
Complex contracts, which hold majority of Ether, are ripe targets for attackers.

Market cap > $50B
MadMax is Unique

Cutting-edge (exhaustive) static analysis
- Abstract Interpretation, CFA Flow Analysis, memory modeling

Performs analysis directly on the bytecode
- Source code only available for 0.34% of contracts (Etherscan)
- Developed the Vadnal decompiler for this purpose.

Evaluated on the entire Ethereum blockchain
- Found $5B on vulnerable contracts (81% estimated precision)
Gas-focussed vulnerabilities

nevilegrech.com/2018/madmax.html
Gas Focussed Vulnerabilities

• Gas is needed to execute contracts:
  • Paid for by the account that calls the smart contract.
  • Has monetary value - prevents wasting of resources.
  • If not enough gas is budgeted, transaction is reverted.
  • Possibly blocking forever due to lack of progress.

• Contract susceptible to DoS attacks if attacker can cause it to require unbounded gas.
Vulnerability 1: Unbounded Mass Ops

```
contract NaiveBank {
    struct Account {
        address addr;
        uint balance;
    }
    Account accounts[];

    function applyInterest() returns (uint) {
        for (uint i = 0; i < accounts.length; i++) {
            // apply 5 percent interest
            accounts[i].balance = accounts[i].balance * 105 / 100;
        }
        return accounts.length;
    }
    function openAccount() returns (uint) {
    }
}
```
Vulnerability 2: Wallet Griefing

```solidity
for (uint i = 0; i < investors.length; i++) {
    if (investors[i].invested < min_investment) {
        // Refund, and check for failure.
        // Looks benign but locks entire contract
        // if attacked by a griefing wallet.
        if (!(investors[i].addr.send(investors[i].dividendAmount))) {
            throw;
        }
    }
    investors[i] = newInvestor;
}
```
Vulnerability 3: Integer Overflow

```solidity
contract Overflow {
    Payee payees[];

    function goOverAll() {
        for (var i = 0; i < payees.length; i++) {
            ...
        }
    }
}
```
The Vandal Decompiler

nevillagech.com/2018/madmax.html
Control Flow in EVM Bytecode

PUSH4 <return>  // return address
PUSH4 0xFF      // push data
PUSH4 <foo>     // function address
JUMP            // jumps to ‘foo’

return:  JUMPDEST
...           
...           
foo:        JUMPDEST
POP          // pops data
JUMP         // jumps to ‘return’

Detect flows of addresses
Decomposition in a Nutshell

1. Basic block boundaries
2. Stack shape and data flow
3. Jump targets
4. Function boundaries
5. Conversion to 3-address IR

Why context sensitivity?
Intermediate Language

to := \text{CONST}(c)

where to : Variable, c : Const

\text{JUMPI}(\text{cond, label})

where cond : Variable, label : Statement

to := \text{SHA3}(\text{index, length})

where index, length, to : Variable
Higher level analyses
Higher Level Analyses

Structured loop reconstruction:
- Induction Variables & Loop Exit Conditions

Alias Analyses

High level data structure semantic analysis

Cool concepts such as:
- IncreasedStorageOnPublicFunction
- PossiblyResumableLoop
Modeling Storage & Data Structures

```
contract Foo {
    uint i0;
    uint i1;
    uint [][]a;
    ...
}
```

<table>
<thead>
<tr>
<th>address</th>
<th>contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>i0</td>
</tr>
<tr>
<td>1</td>
<td>i1</td>
</tr>
<tr>
<td>2</td>
<td>a.length</td>
</tr>
<tr>
<td>SHA3(2)</td>
<td>a[0].length</td>
</tr>
<tr>
<td>SHA3(2) + 1</td>
<td>a[1].length</td>
</tr>
<tr>
<td>SHA3(SHA3(2))</td>
<td>a[0][0]</td>
</tr>
<tr>
<td>SHA3(SHA3(2)) + 1</td>
<td>a[0][1]</td>
</tr>
<tr>
<td>SHA3(SHA3(2) + 1)</td>
<td>a[1][0]</td>
</tr>
<tr>
<td>SHA3(SHA3(2) + 1) + 1</td>
<td>a[1][1]</td>
</tr>
</tbody>
</table>
Example top-level query

UnboundedMassOp(loop) ←
  IncreasedStorageOnPublicFunction(arrayId) □
  ArrayIdToStorageIndex(arrayId, storeOffsetVar) □
  Flows(storeOffsetVar, index) □
  VarIndexesStorage(storeOrLoadStmt, index) □
  InLoop(storeOrLoadStmt, loop) □
  ArrayIterator(loop, arrayId) □
  InductionVar(i, loop) □
  Flows(i, index) □
  !PossiblyResumableLoop(loop).
Experimental Evaluation
Results: Effectiveness

Analysed entire blockchain:
6.33M contracts (90k unique) in 10 hours
4.1% susceptible to unbounded iteration.
0.12% susceptible to wallet griefing.
1.2% susceptible to loop overflows.

Combined holding of 7.07 million ETH ~ $5B

81% estimated precision
Insights: Iteration and Data Structures

Reconstructing high level data structure semantics critical for low false positive rate.
## Related work

<table>
<thead>
<tr>
<th>Approach</th>
<th>Works</th>
<th>Soundy</th>
<th>Automated</th>
<th>Bytecode</th>
<th>General</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symbolic Execution</td>
<td>- Oyente by Luu et al. (2016)</td>
<td>❌</td>
<td>✅</td>
<td>✅</td>
<td>❌</td>
</tr>
<tr>
<td></td>
<td>- Maian by Nikolic et al. (2018)</td>
<td></td>
<td>✅</td>
<td>✅</td>
<td>❌</td>
</tr>
<tr>
<td></td>
<td>- gasper by Chen et al. (2017)</td>
<td></td>
<td></td>
<td>✅</td>
<td>❌</td>
</tr>
<tr>
<td></td>
<td>- Grossman et al. (2017)</td>
<td></td>
<td></td>
<td></td>
<td>❌</td>
</tr>
<tr>
<td></td>
<td>- Proofs in the K framework by Hildenbrandt et al. (2017)</td>
<td></td>
<td></td>
<td>❌</td>
<td>❌</td>
</tr>
<tr>
<td></td>
<td>- Formalism of EVM in F* by Bhargavan et al. (2016)</td>
<td></td>
<td></td>
<td>❌</td>
<td>❌</td>
</tr>
<tr>
<td>Abstract interpretation on Solidity</td>
<td>- Zeus by Kalra et al. (2018)</td>
<td></td>
<td></td>
<td>❌</td>
<td>❌</td>
</tr>
<tr>
<td></td>
<td>- FSolidM by Mavridou and Laszka (2018)</td>
<td></td>
<td></td>
<td>❌</td>
<td>❌</td>
</tr>
<tr>
<td>Abstract interpretation on EVM bytecode</td>
<td>MadMax (OOPSLA'18) (Our Approach)</td>
<td></td>
<td></td>
<td>❌</td>
<td>❌</td>
</tr>
</tbody>
</table>
Conclusions

MadMax, a vulnerability detection tool:
- Scales to the entire Blockchain
- Interesting results, practical impact

Datalog lends itself well to:
- Program analyzers (even flow sensitive ones)
- High level decompilers

Decomposition is a very important step & current work focuses on this
Current work: Fully declarative decompilation

1. Whole contract ctx & flow sensitive analysis

2. Function extraction algorithms

3. Function argument inference with flow sensitive analysis

Original Bytecode → Whole program 3 address IR + CFG → 3 address IR + function bounds + local CFGs → Functional 3-address code