

# Analysis of Access Control Management in Ethereum Smart Contracts

Thomas Hain, 20/05/2019, Master Thesis Kick-Off

Chair of Software Engineering for Business Information Systems (sebis)  
Faculty of Informatics  
Technische Universität München  
[www.matthes.in.tum.de](http://www.matthes.in.tum.de)

1. Motivation
2. Applied Access Control
3. Research Questions
4. Approach
5. Timeline

There are varying definitions of Access Control (AC)

## Thesis understanding:

*"Access Control is regarded as limiting a user's access to a resource within a system"*

- Enterprise structures often have **complex hierarchies**
- Current business software based on languages like Java follow **Access Control Patterns**
- Underestimating importance of Access Restrictions leads to **security breaches**

Companies have to follow **security standards** (see certifications → ISO, EU-GDPR\*, etc...)

- How common is Access Control adaptation in Ethereum Smart Contracts?
- Which Types and Software Patterns of Access Control exist?

\*Source: <https://eugdpr.org/>

# Applied Access Control

## **Mandatory Access Control (MAC)<sup>1,3</sup>**

- Subject and object have a security level
- Levels can only be changed by the security officer
- If subject's level is sufficient read and/or write access is granted

## **Discretionary Access Control (DAC)<sup>1,3</sup>**

- Object owner decides about other users' access rights
- Control access based on the identity of the requestor

## **Role-Based Access Control (RBAC)<sup>3</sup>**

- Users are grouped by their roles (e.g. accounting, controlling)
- A users permissions are defined by the roles he holds
- Each role has permissions and / or restrictions
- Permissions can be inherited via the establishment of role hierarchies

# Applied Access Control

A **smart contract's** methods are **called via transactions**

- Sender's **address** is included in transaction
- Can either be **user** or another **contract**

Common Access Control Patterns in Ethereum include:

## RBAC<sup>6</sup>

- Map **roles** to **addresses**
- Introduce **conditional statements**, whether user is **member** of role
- **Annotate methods** with defined conditional checks
- Implemented in OpenZeppelin framework (widely used)

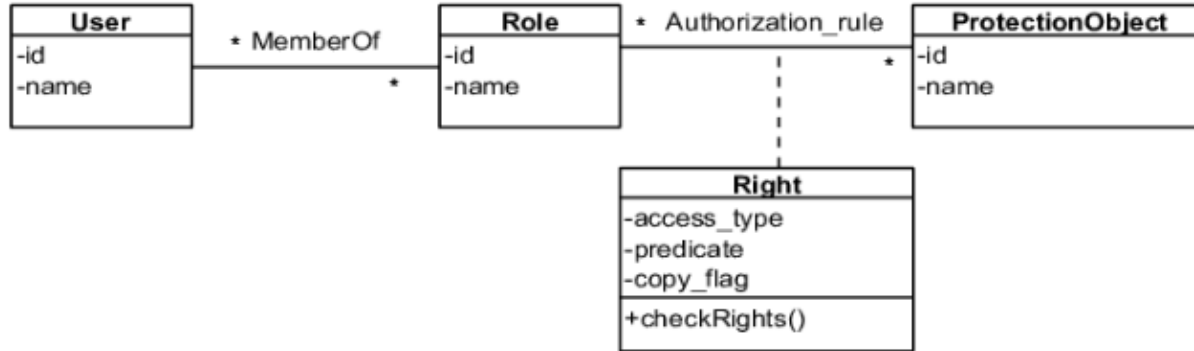
## Ownership-Pattern<sup>7</sup>

- During contract deployment **sender's address** is **stored as owner**
- Introduce **modifier** `onlyOwner`, similar to MAC
- **onlyOwner** checks: transaction sender's address == owner address
- **Annotate methods with modifier** to introduce AC checks

# Research Questions

- 1 How commonly are Access Control frameworks used within Ethereum?
- 2 Which Source Code patterns are frequently used for Access Control?
- 3 How to measure similarity of AC mechanisms of multiple contracts?
- 4 How to recognize high-level Access Control on byte-level?
- 5 Is it possible to find a metric, which indicates AC usage in a contract?

Design Patterns are usually given as UML diagrams\*



→ They are easily implemented as source code

smart contracts are deployed publicly on the blockchain, **but**

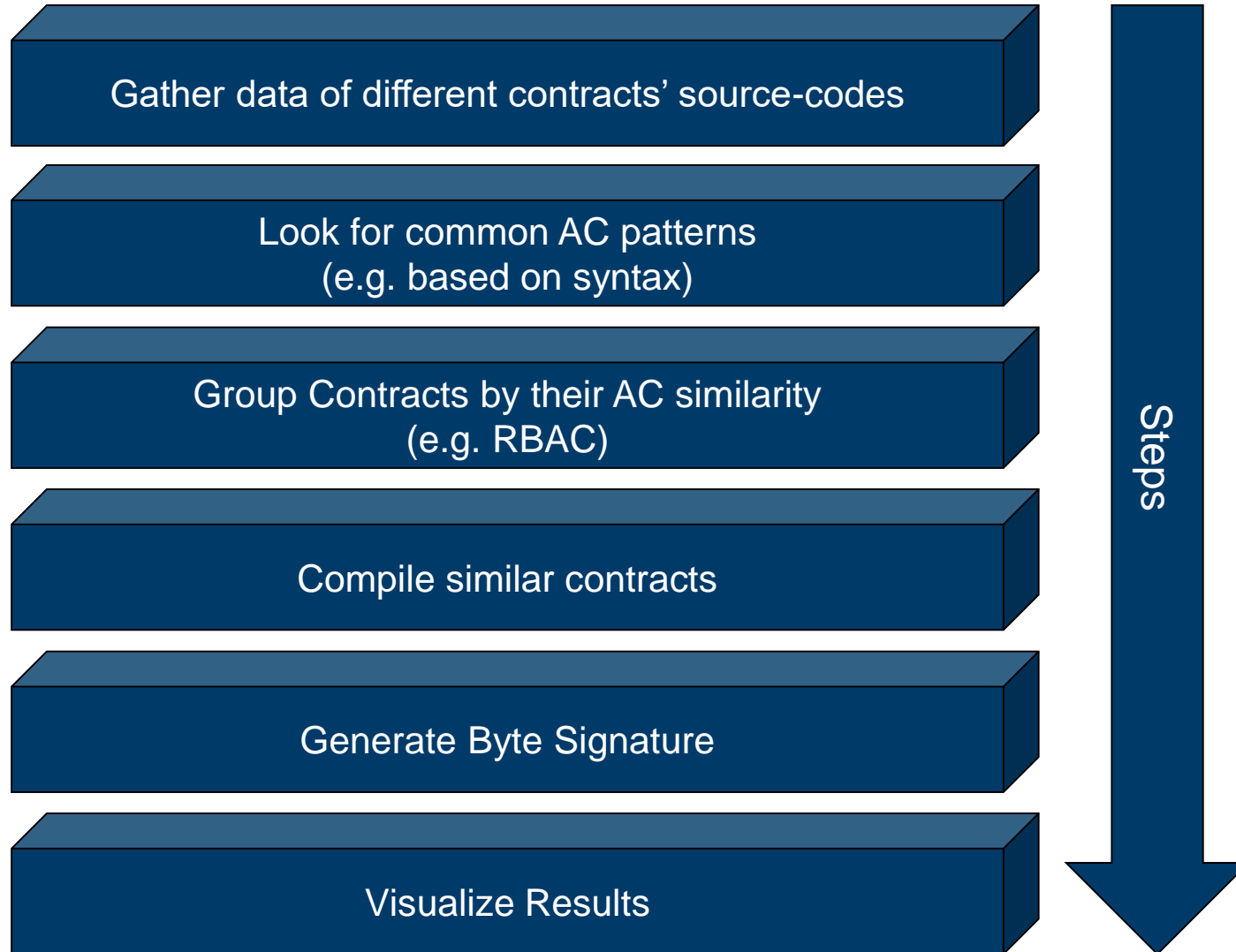
→ **compiled** (therefore only known as binary representation)

**Important question:**

How can we reconstruct information about applied Access Control?

\*UML Source: [https://www.researchgate.net/figure/Structure-of-RBAC-Figures-2-shows-structure-of-Role-Based-Access-Control-RBAC-as\\_fig4\\_274720701](https://www.researchgate.net/figure/Structure-of-RBAC-Figures-2-shows-structure-of-Role-Based-Access-Control-RBAC-as_fig4_274720701)

# Approach





# Approach

## Step: Gather data of different contracts' source-codes

Website **Etherscan**\* allows programmers to **publish** source-codes

**Motivation**: Companies prove, that a contract is doing what it's intended to do

By using web scraping, a **MySQL database** of contracts was generated

About **150k** contracts, including:

- Source code
- Name
- Address
- ...



➔ Serves as testing ground for pattern identification

\*Source: <https://etherscan.io/contractsVerified>

# Approach

**Step: Look for common AC patterns (e.g. based on syntax)**

As already mentioned, **two\*** prominent patterns emerge:

## **Ownership and RBAC**

Both require little implementation effort

→ few syntactic elements

→ serve as **good starting point** for further analysis

\*based on literature research (e.g. sources 6,7,8) and own analysis

# Approach

**Step: Group Contracts by their AC similarity (e.g. RBAC contracts)**

Reference Literature<sup>4,5</sup> compare their **Abstract Syntax Trees (AST)**.

**ASTs are based on a language's Grammar.**

**Most common compiler Solidity can generate ASTs from source-code**

Shortened excerpt from Ethereum's Official Grammar Definition\*

*PrimaryExpression = BooleanLiteral*

*| NumberLiteral*

*| StringLiteral*

*| Identifier*

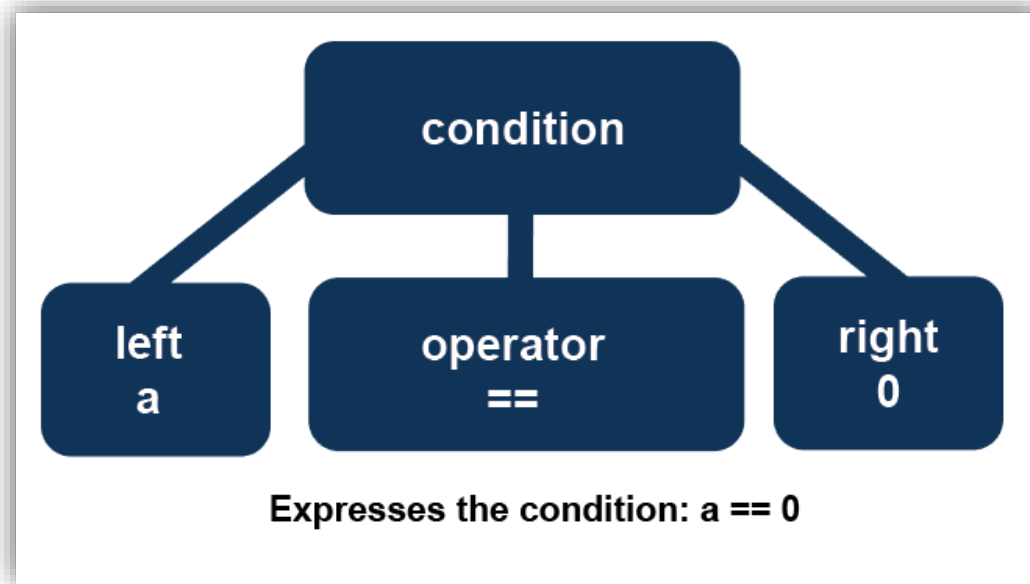
*BooleanLiteral = 'true' | 'false'*

\*Source: <https://github.com/ethereum/solidity/blob/develop/docs/grammar.txt>

# Approach

```
1 {  
2   "type": "IfStatement",  
3   "condition": {  
4     "type": "BinaryOperation",  
5     "operator": "==",  
6     "left": {  
7       "type": "Identifier",  
8       "name": "a"  
9     },  
10    "right": {  
11      "type": "NumberLiteral",  
12      "number": "0",  
13      "subdenomination": null  
14    }  
15  }
```

Excerpt of Etherscan Contract AST



Corresponding Sub-AST

# Approach

**Step: Group Contracts by their AC similarity (e.g. RBAC contracts)**

**Problem: How do we compare the similarity of multiple given source codes / programs?**

Reference Literature<sup>4,5</sup> **compare** their **Abstract Syntax Trees (AST)**.

**How are ASTs compared?**

**Bottom-up Maximum Common Subtree Isomorphism<sup>4</sup>**

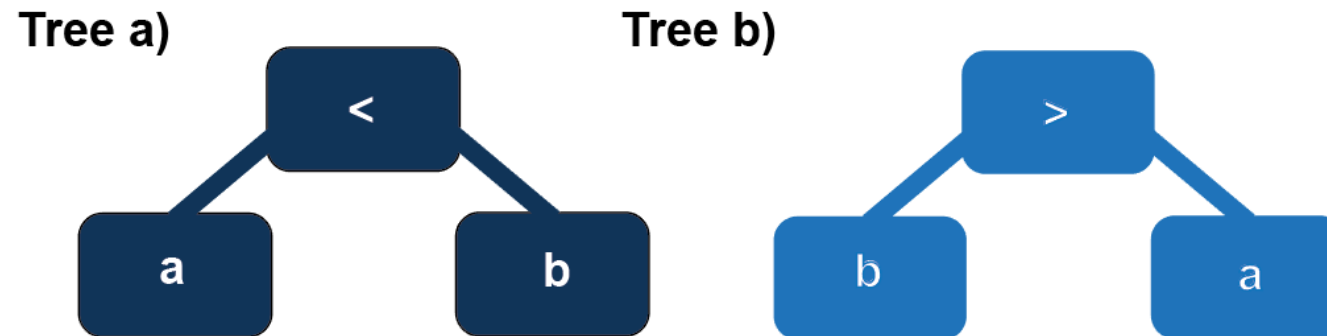
→ Find the largest (most nodes) subtree within two syntax trees, which are equal

**Tree Edit Distance<sup>4</sup>**

→ how many trivial (insertion, deletion, ...) steps does it take to transform one tree into another tree

**Step: Group Contracts by their AC usage similarity (e.g. RBAC contracts)**

**ASTs can differ while expressing the same AC patterns,**  
e.g. same semantics  $\Leftrightarrow$  different syntactic order



- Algorithms need to accommodate this behavior to a certain extent
- **But:** Not a perfect (i.e. 100%) match is required
- **Goal I:** Implement an **AST parser**, which measures the AC usage similarity of contracts

# Approach

## Step: Compile similar contracts

After grouping contracts by their AC type, they are compiled

### Source-Code:

```
2 import "remix_tests.sol"; // this import is automatically injected by Remix.
3 import "./ballot.sol";
4
5 contract test3 {
6
7     Ballot ballotToTest;
8     function beforeAll () public {
9         ballotToTest = new Ballot(2);
10    }
11
12    function checkWinningProposal () public {
13        ballotToTest.vote(1);
14        Assert.equal(ballotToTest.winningProposal(), uint(1), "1 should be the winning proposal");
15    }
16
17    function checkWinninProposalWithReturnValue () public view returns (bool) {
18        return ballotToTest.winningProposal() == 1;
19    }
20 }
```

## Step: Compile similar contracts

After grouping contracts by their AC type, they are compiled

**Binary representation as HEX:**

```
608060405234801561001057600080FD5B506040516020806108668339810180604052602081101561003057600080FD5B8101908080519060200190929190505050336000806101
000A81548173FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF021916908373FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF160217905550600180600080600090549
06101000A900473FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF1673FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF1673FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFF168152602001908152602001600020600001819055508060FF166002816100FA9190610101565B5050610154565B815481835581811115610128578183600052602060
00209182019101610127919061012D565B5B505050565B61015191905B8082111561014D5760008082016000905550600101610133565B5090565B90565B610703806101636000
396000F3FE608060405234801561001057600080FD5B506004361061004C5760003560E01C80635C19A95C14610051578063609FF1BD146100955780639E7B8D61146100B957
8063B3F98ADC146100FD575B600080FD5B6100936004803603602081101561006757600080FD5B81019080803573FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF169060
20019092919050505061012E565B005B61009D610481565B604051808260FF1660FF16815260200191505060405180910390F35B6100FB600480360360208110156100CF576000
80FD5B81019080803573FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF1690602001909291905050506104F9565B005B61012C6004803603602081101561011357600080F
D5B81019080803560FF1690602001909291905050506105F6565B005B6000600160003373FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF1673FFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF16815260200190815260200160002090508060010160009054906101000A900460FF161561018E575061047E565B5B600073FFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF16600160008473FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF1673FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF16815260200190815
260200160002060010160029054906101000A900473FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF1673FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF1614158015
6102BC57503373FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF16600160008473FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF1673FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF16815260200190815260200160002060010160029054906101000A900473FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF1673FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF1614155B1561032B57600160008373FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF1673FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFF16815260200190815260200160002060010160029054906101000A900473FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF16915061018F565B3373FFFFFFFFFFFF
FFFFFFFFFFFFFFFF168273FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF161415610365575061047E565B60018160010160006101000A81548160FF0219
16908315150217905550818160010160026101000A81548173FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF021916908373FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFF1602179055506000600160008473FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF1673FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF16815260200190815260
2001600020905080600101
```



# Approach

## Step: Compile similar contracts

After categorizing contracts by their AC type, they are **compiled**

Binary Representation **encodes OP-Codes, e.g.:**

**PUSH1 0x80 PUSH1 0x40 MSTORE CALLVALUE DUP1 ISZERO...**

OP-Codes are commands, which instruct a stack-based virtual machine

→ Ethereum Virtual Machine (EVM)

**This way contracts are executed!**

# Approach

## Step: Generate Byte Signature

### Binary Code

```
04E86B70 6A 01  
04E86B72 E8 C95C1200  
04E86B77 83 C4 04
```

### OP-Code Representation

```
push 01  
call 04FAC840  
add esp,04
```

Generated Signature:

```
6A 01 E8 ?? ?? ?? ?? 83 C4 04
```

→ Important: Signature needs to be robust

### Balance between:

→ too short / too much wildcarding → false positives

→ too long / too few wildcarding → false negatives

**Goal I:** Implement an AST parser, which measures the AC usage similarity of contracts

**Goal II:** Generate stable AC byte signatures for contracts with non-public source code

**If the stated approach succeeds, it will then be possible to answer:**

How commonly are Access Control frameworks used within Ethereum?

*E.g. OpenZeppelin RBAC signature*

Which Source Code patterns are frequently used for Access Control?

*RBAC, Ownership, and **possibly more?***

How to measure similarity of AC mechanisms of multiple contracts?

*ASTs + ByteCode, possibly assisted by analysing ControlFlowGraphs?*

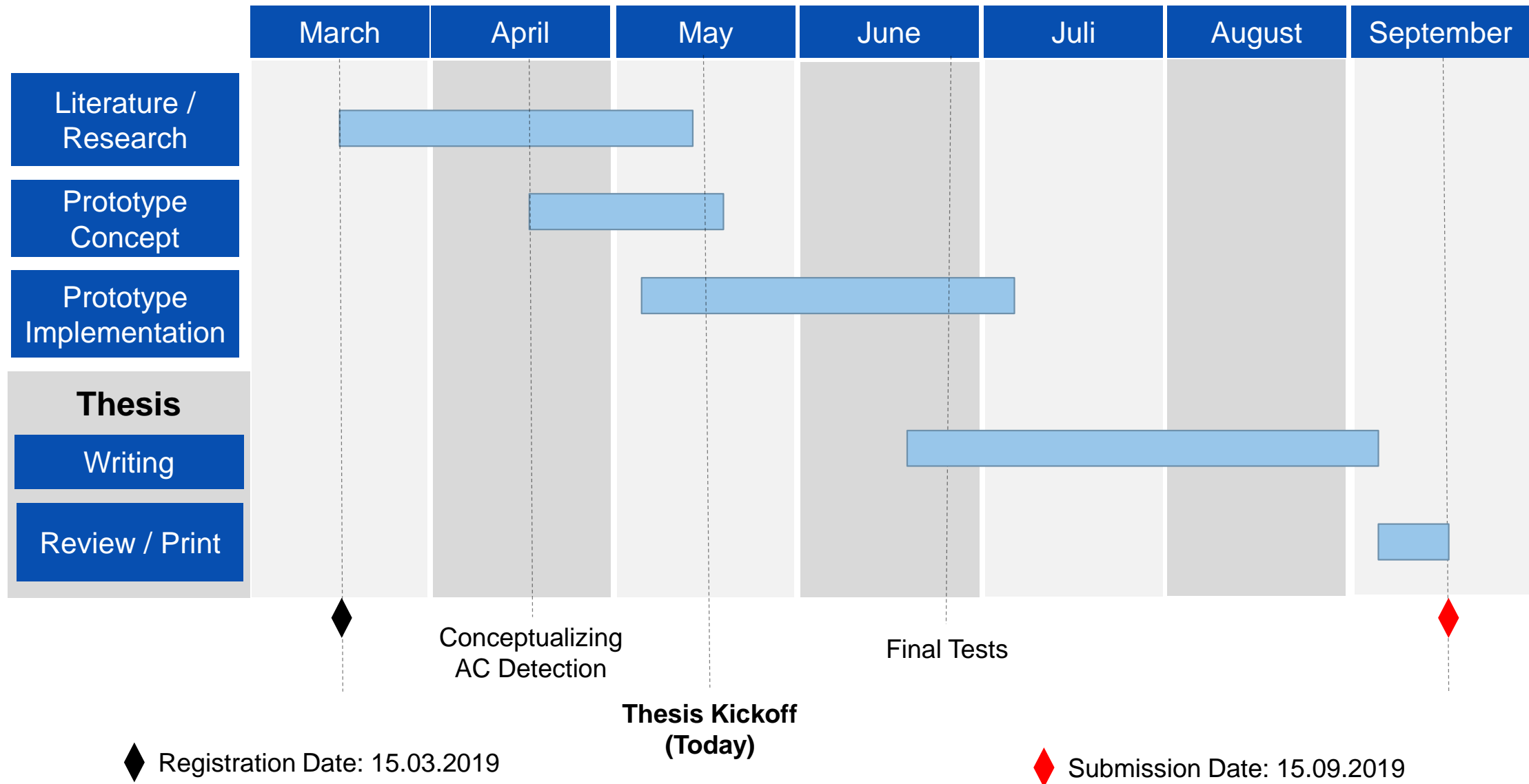
How to recognize high-level Access Control on byte-level?

*By inferring byte-level patterns from source-code patterns*

Is it possible to find a metric, which indicates AC usage in a contract?

*Matching Ratio to Byte Signature and / or subtrees of ASTs*

# Timeline



1. Osborn, Sylvia, Ravi Sandhu, and Qamar Munawer. "Configuring role-based access control to enforce mandatory and discretionary access control policies." *ACM Transactions on Information and System Security (TISSEC)* 3.2 (2000): 85-106.
2. Samarati, Pierangela, and Sabrina Capitani de Vimercati. "Access control: Policies, models, and mechanisms." *International School on Foundations of Security Analysis and Design*. Springer, Berlin, Heidelberg, 2000.
3. Osborn, Sylvia. "Mandatory access control and role-based access control revisited." *IN PROCEEDINGS OF THE 2ND ACM WORKSHOP ON ROLE-BASED ACCESS CONTROL*. 1997.
4. Sager, Tobias, et al. "Detecting similar Java classes using tree algorithms." Proceedings of the 2006 international workshop on Mining software repositories. ACM, 2006.
5. Chilowicz, Michel, Etienne Duris, and Gilles Roussel. "Syntax tree fingerprinting for source code similarity detection." *2009 IEEE 17th International Conference on Program Comprehension*. IEEE, 2009
6. Cruz, Jason Paul, Yuichi Kaji, and Naoto Yanai. "RBAC-SC: Role-based access control using smart contract." *IEEE Access* 6 (2018): 12240-12251.
7. Wöhler, Maximilian, and Uwe Zdun. "Design patterns for smart contracts in the ethereum ecosystem." (2018).
8. [https://docs.openzeppelin.org/docs/ownership\\_rbac\\_rbac](https://docs.openzeppelin.org/docs/ownership_rbac_rbac), Accessed 05/02/2019
9. Ferraiolo, David F., John F. Barkley, and D. Richard Kuhn. "A role-based access control model and reference implementation within a corporate intranet." *ACM Transactions on Information and System Security (TISSEC)* 2.1 (1999): 34-64.
10. Klarl, Heiko, et al. "Extending Role-based Access Control for Business Usage." *2009 Third International Conference on Emerging Security Information, Systems and Technologies*. IEEE, 2009.
11. KOCH, Manuel; MANCINI, Luigi V.; PARISI-PRESICCE, Francesco. Graph-based specification of access control policies. *Journal of Computer and System Sciences*, 2005, 71. Jg., Nr. 1, S. 1-33.



B. Sc.

**Thomas Hain**

Technische Universität München  
Faculty of Informatics  
Chair of Software Engineering for Business  
Information Systems

Boltzmannstraße 3  
85748 Garching bei München

Tel +49.89.289. 17132  
Fax +49.89.289.17136

matthes@in.tum.de  
[wwwmatthes.in.tum.de](http://wwwmatthes.in.tum.de)



Symbolic Execution: Find necessary inputs to cover all unique paths

Example from Wikipedia: Program **fails** when  $y = 6$

```
1 int f() {  
2   ...  
3   y = read();  
4   z = y * 2;  
5   if (z == 12) {  
6     fail();  
7   } else {  
8     printf("OK");  
9   }  
10 }
```

**Two Paths:** If (Line 5) and else (Line 7)

3) set  $y = \lambda$

4) set  $z = 2*\lambda$

5) Fork

5-True) fail()  $\rightarrow$  constraint  $2*\lambda = 12 \rightarrow \lambda = 6$

5-False) "OK"  $\rightarrow$  constraint  $2*\lambda \neq 12 \rightarrow \lambda \neq 6 \rightarrow \lambda = 1$

**Governance scenario:** changing ownership is access restricted

Generate different userlevel inputs:

if userlevel == admin  $\rightarrow$  success

else  $\rightarrow$  fail



<https://github.com/trailofbits/manticore/>

## Manticore



build passing pypi package 0.2.4 slack 40/1928 docs passing maintainability A test coverage 71%

Manticore is a symbolic execution tool for analysis of smart contracts and binaries.

Note: Beginning with version 0.2.0, Python 3.6+ is required.

### Features

- **Input Generation:** Manticore automatically generates inputs that trigger unique code paths
- **Error Discovery:** Manticore discovers bugs and produces inputs required to trigger them
- **Execution Tracing:** Manticore records an instruction-level trace of execution for each generated input
- **Programmatic Interface:** Manticore exposes programmatic access to its analysis engine via a Python API

Manticore can analyze the following types of programs:

- Ethereum smart contracts (EVM bytecode)

Zooming out // Post-Processing:

Idea: Labelling clustered contracts and applying distance measures

Data

- Contracts grouped in sets based on their AC similarity
- Labels from scraping + fall-back manual labelling

Additional information (e.g. from internet research):

- Their institutional models (e.g. GmbH, ...)
- Known important stakeholders, CTOs, etc...

→ Further assessment of the power a respective role holds

## 1) Ownership:

### OWNERSHIP PATTERN

**Problem** By default any party can call a contract method, but it must be ensured that sensitive contract methods can only be executed by the owner of a contract.

**Solution** Store the contract creator's address as owner of a contract and restrict method execution dependent on the callers address.

It is very common that only the owner of a contract should be eligible to call functions, which are sensitive and crucial for the correct operation of the contract. This pattern limits access to certain functions to only the owner of the contract, an example is shown in Listing 7. A typical application of this pattern is demonstrated in the Mortal pattern.

```
pragma solidity ^0.4.17;
contract Owned {
    address public owner;

    event LogOwnershipTransferred(address indexed
        previousOwner, address indexed newOwner);

    modifier onlyOwner() {
        require(msg.sender == owner);
        _;
    }

    function Owned() public {
        owner = msg.sender;
    }

    function transferOwnership(address newOwner) public
        onlyOwner {
        require(newOwner != address(0));
        LogOwnershipTransferred(owner, newOwner);
        owner = newOwner;
    }
}
```

## 1) Ownership:

### OWNERSHIP PATTERN

**Problem** By default any party can call a contract method, but it must be ensured that sensitive contract methods can only be executed by the owner of a contract.

**Solution** Store the contract creator's address as owner of a contract and restrict method execution dependent on the callers address.

It is very common that only the owner of a contract should be eligible to call functions, which are sensitive and crucial for the correct operation of the contract. This pattern limits access to certain functions to only the owner of the contract, an example is shown in Listing 7. A typical application of this pattern is demonstrated in the Mortal pattern.

```
pragma solidity ^0.4.17;
contract Owned {
    address public owner;

    event LogOwnershipTransferred(address indexed
        previousOwner, address indexed newOwner);

    modifier onlyOwner() {
        require(msg.sender == owner);
        _;
    }

    function Owned() public {
        owner = msg.sender;
    }

    function transferOwnership(address newOwner) public
        onlyOwner {
        require(newOwner != address(0));
        LogOwnershipTransferred(owner, newOwner);
        owner = newOwner;
    }
}
```

[https://docs.openzeppelin.org/docs/ownership\\_rbac\\_rbac](https://docs.openzeppelin.org/docs/ownership_rbac_rbac)

## Modifiers

### onlyRole

```
modifier onlyRole(string _role)
```

Modifier to scope access to a single role (uses msg.sender as addr).

#### Parameters:

`_role` - the name of the role // reverts

## Functions

### addRole

```
function addRole(address _operator, string _role) internal
```

Add a role to an address.


#### Parameters:

`_operator` - address

`_role` - the name of the role

# Backup

<https://etherscan.io/address/0xd1faa35d9ffda86f9804dc764d02256919ccb8f9#code>

✔ Contract Source Code Verified (Exact Match) 

Contract Name: RTDAirDrop

Optimization Enabled: Yes with 200 runs

Compiler Version: v0.4.25+commit.59dbf8f1

Evm Version: default

Contract Source Code (Solidity)

Find Similar Contracts



```
1- |/**
2  | * Source Code first verified at https://etherscan.io on Monday, May 20, 2019
3  | (UTC) */
4  |
5  | pragma solidity ^0.4.16;
6  |
7- | contract owned {
8  |     address public owner;
9  |
10- |     constructor() public {
11  |         owner = msg.sender;
12  |     }
13  |
14- |     modifier onlyOwner {
15  |         require(msg.sender == owner);
16  |         _;
17  |     }
18  |
19- |     function transferOwnership(address newOwner) onlyOwner public {
20  |         owner = newOwner;
21  |     }
22  | }
23  |
24  |
25  |
```