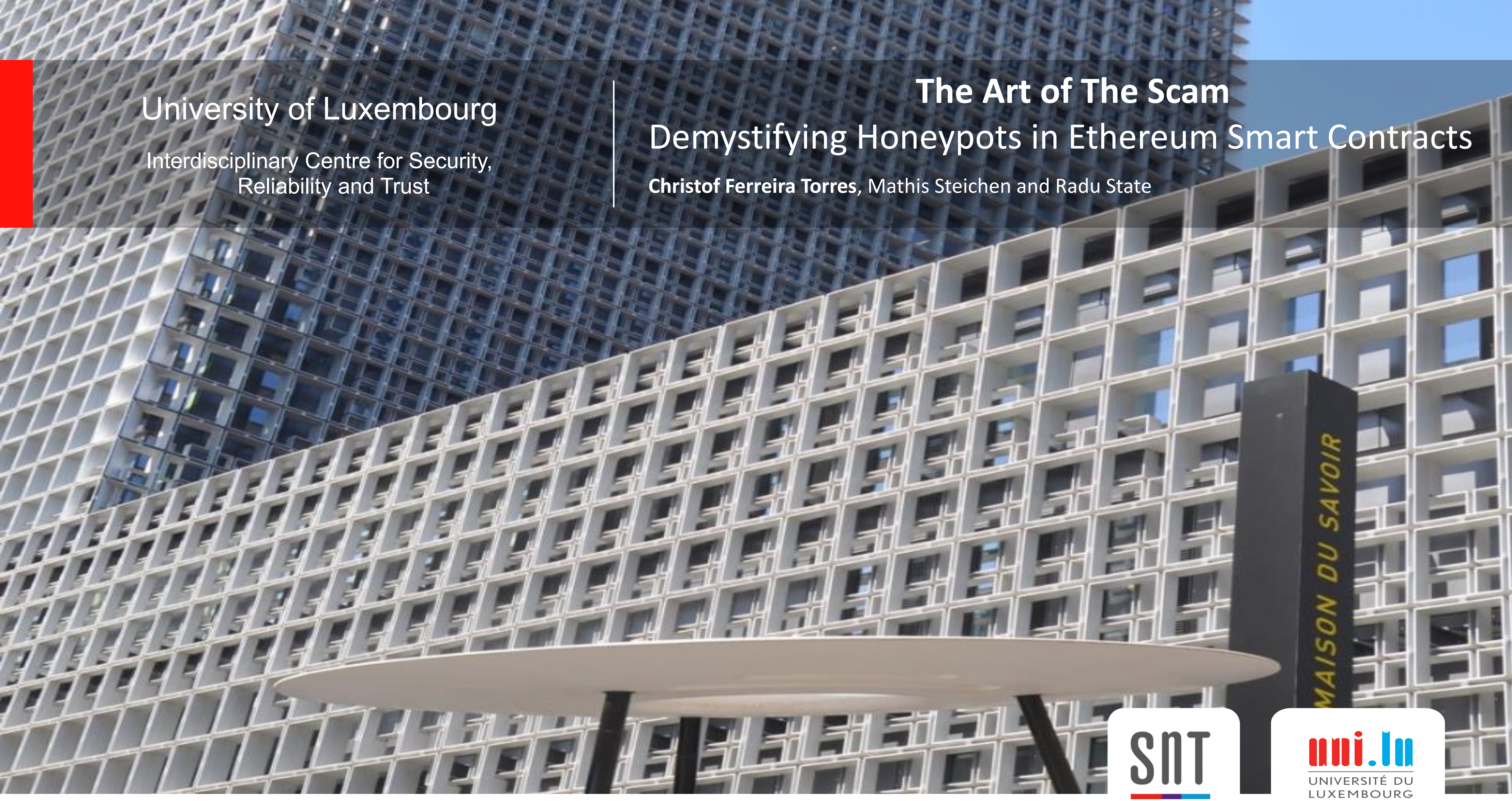
The Art of The Scam

Demystifying Honeypots in Ethereum Smart Contracts

Christof Ferreira Torres, Mathis Steichen and Radu State

University of Luxembourg

Interdisciplinary Centre for Security, Reliability and Trust

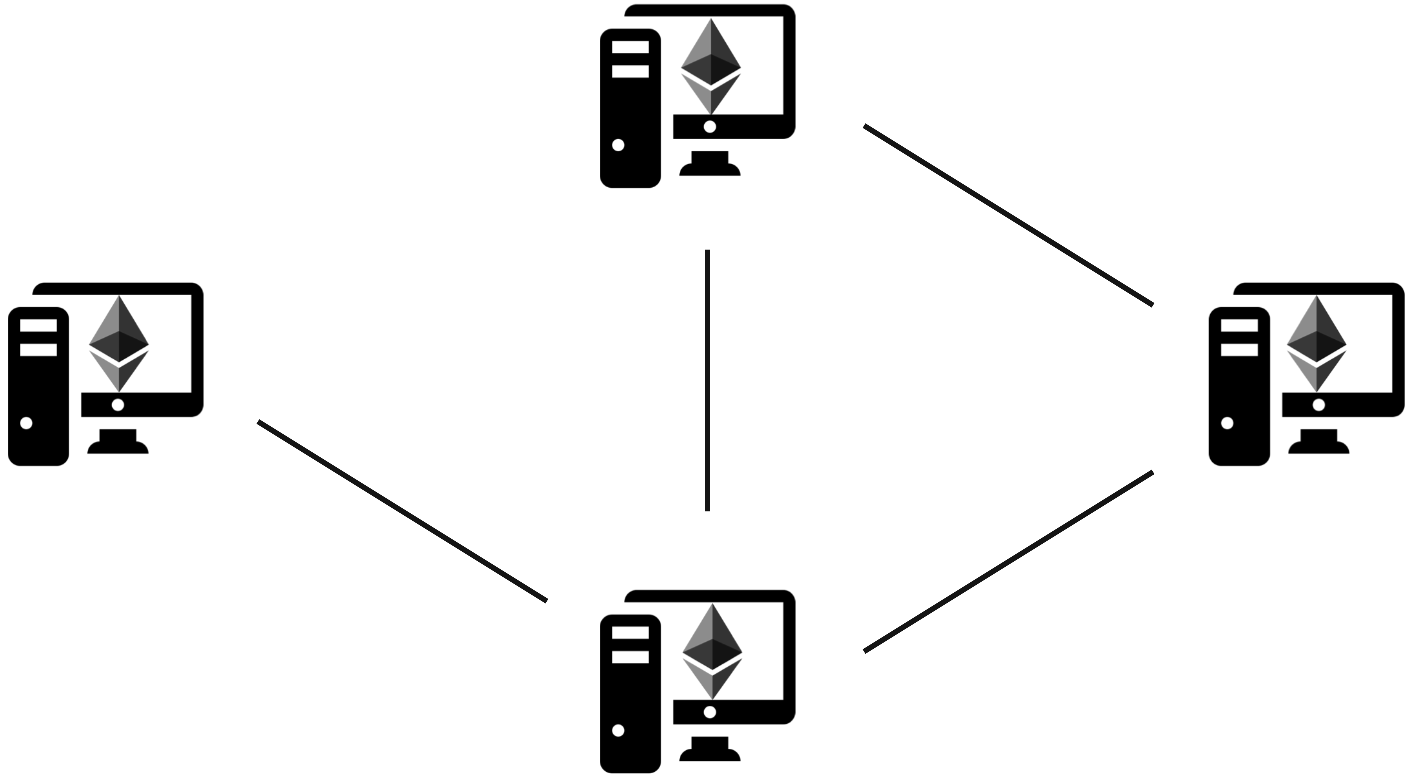


SnT

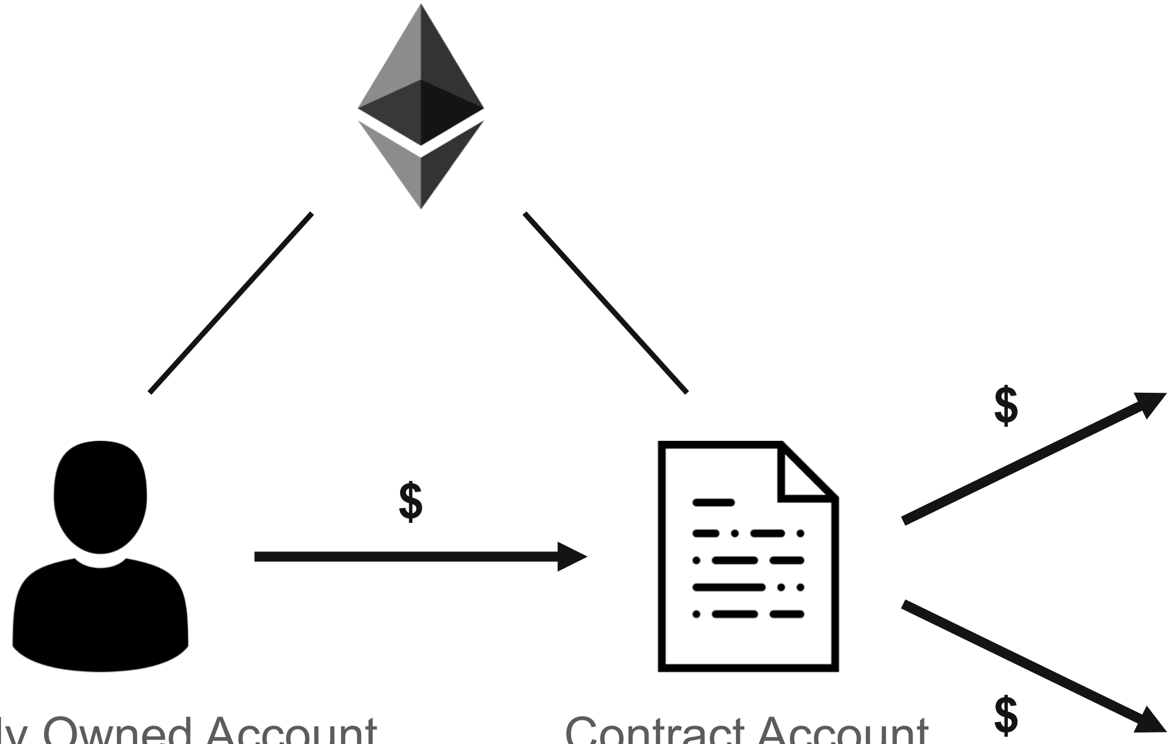
uni.lu
UNIVERSITÉ DU LUXEMBOURG

Ethereum
Crash Course

Ethereum Blockchain



Ethereum Accounts



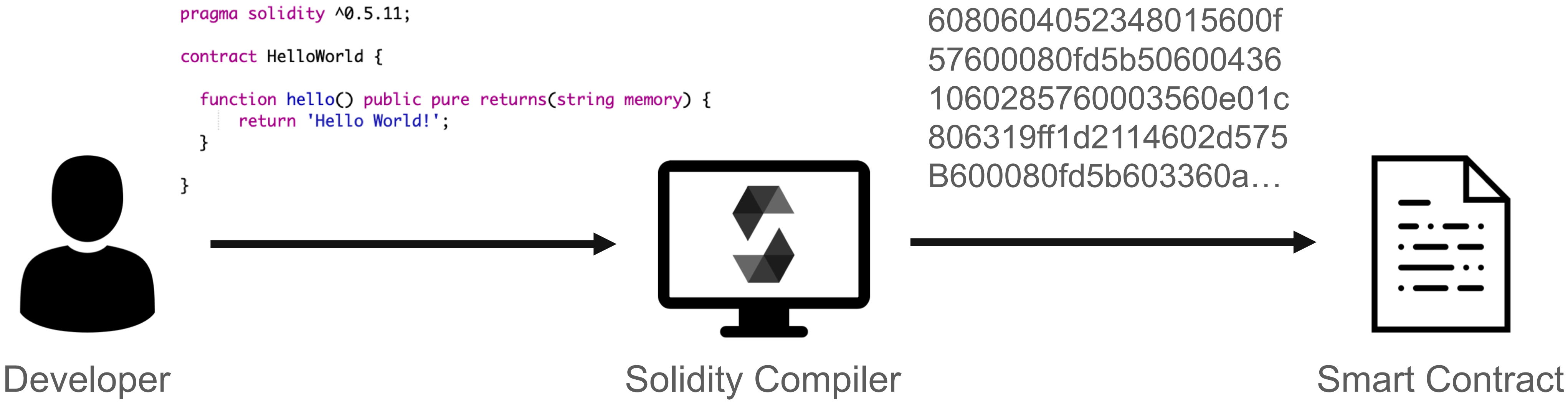
Externally Owned Account

- Address
- Balance

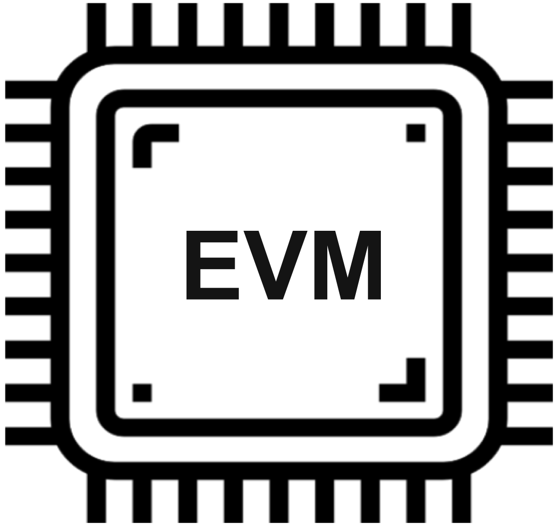
Contract Account

- Address
- Balance
- Code
- Storage

Ethereum Smart Contracts



Ethereum Virtual Machine

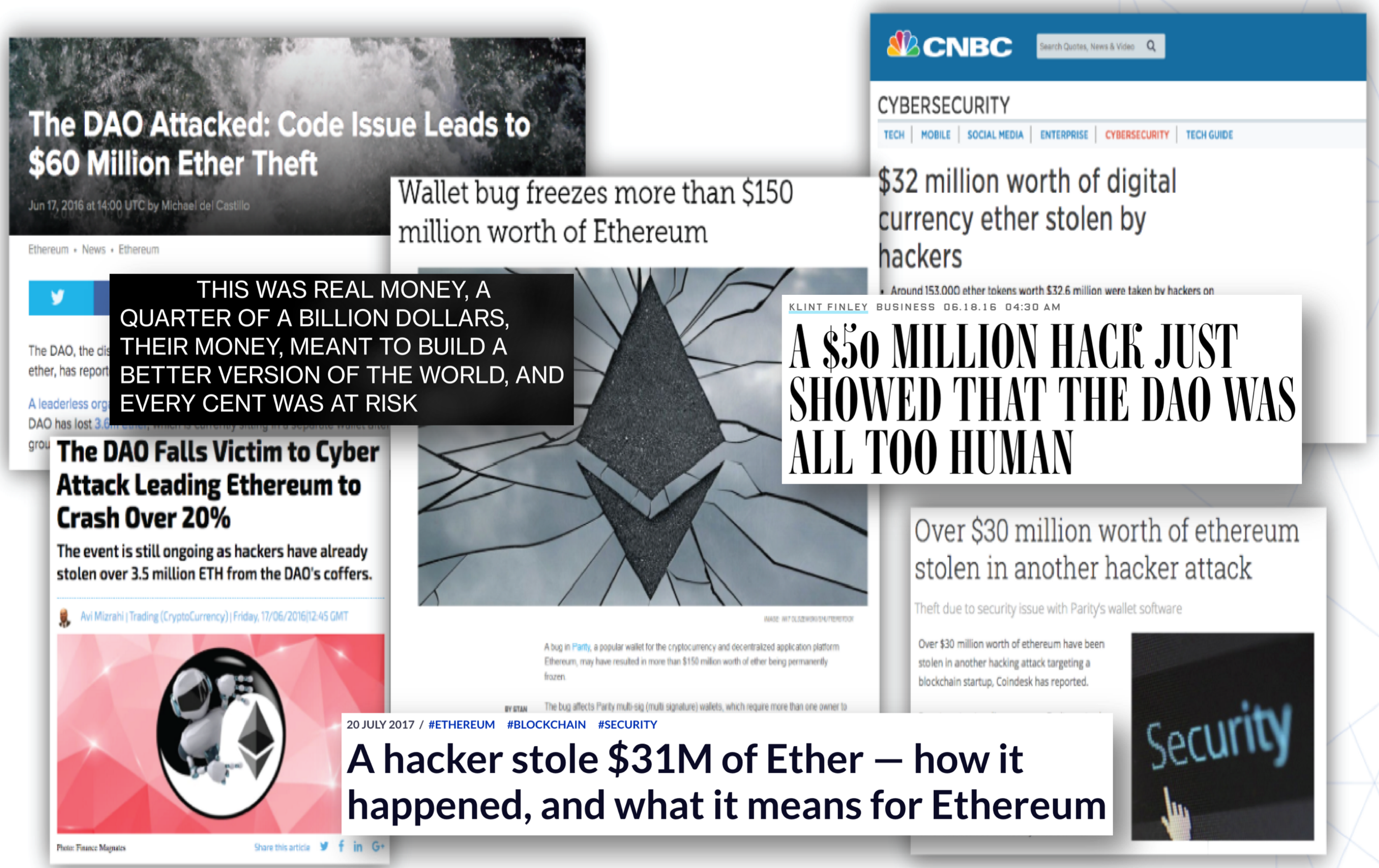


- Turing complete
- Register-less, 256-bit, stack-based VM

- Over **100** instructions:
  - Stack instructions:
    PUSH, SWAP, …
  - Arithmetic instructions:
    ADD, SUB, MUL, …
  - Memory instructions:
    SLOAD, SSTORE, …
  - Control-flow instructions:
    JUMP, JUMPI, …
  - Contract instructions:
    CALL, SELFDESTRUCT, …
  - Error handling instructions:
    REVERT, INVALID, …

Exploiting Smart Contracts

Attacks on Smart Contracts



Automated Exploit Generation

TEETHER: Gnawing at Ethereum to Automatically Exploit Smart Contracts

Johannes Krupp
CISPA, Saarland University, Saarland Informatics Campus

Christian Rossow
CISPA, Saarland University, Saarland Informatics Campus

Abstract

Cryptocurrencies like Bitcoin not only provide a decentralized currency, but also provide a programmatic way to process transactions. Ethereum, the second largest cryptocurrency next to Bitcoin, is the first to provide a Turing-complete language to specify transaction processing, thereby enabling so-called *smart contracts*. This provides an opportune setting for attackers, as security vulnerabilities are tightly intertwined with financial gain. In this paper, we consider the problem of automatic vulnerability identification and exploit generation for smart contracts. We develop a generic definition of vulnerable contracts and use this to build TEETHER, a tool that allows creating an exploit for a contract given only its binary bytecode. We perform a large-scale analysis of all 38,757 unique Ethereum contracts, 815 out of which our tool finds working exploits for—completely automated.

lion USD [1]. Although Bitcoin remains the predominant cryptocurrency, it also inspired many derivative systems. One of the most popular of these is Ethereum, the second largest cryptocurrency by overall market value as of mid 2018 [1].

Ethereum heavily extends the way consensus protocols handle transactions: While Bitcoin allows to specify simple checks that are to be performed when processing a transaction, Ethereum allows these rules to be specified in a Turing-complete language. This makes Ethereum the number one platform for so-called *smart contracts*.

A smart contract can be seen quite literally as a contract that has been formalized in code. As such, smart contracts can for example be used to implement fundraising schemes that automatically refund contributions unless a certain amount is raised in a given time, or shared wallets that require transactions to be approved of by

Limitations of a Reactive Approach

- Attackers are required to scan millions of smart contracts to finds bugs.
- Finding exploitable bugs in smart contracts is becoming more challenging.

"Why should I spend time looking for victims, if I can just let the victims come to me?"

Smart Contract Honeypots

What are Smart Contract Honeypots?

- Smart contracts that look vulnerable but actually are not.

- **Idea:**

  Make users believe that they can exploit a smart contract by sending funds to it, while in reality only the smart contract creator is be able to retrieve them.

Multiplicator Honeypot

```
contract MultiplicatorX3
{
    address public Owner = msg.sender;

    function() public payable{}

    function withdraw()
    payable
    public
    {
        require(msg.sender == Owner);
        Owner.transfer(this.balance);
    }

    function Command(address adr,bytes data)
    payable
    public
    {
        require(msg.sender == Owner);
        adr.call.value(msg.value)(data);
    }

    function multiplicate(address adr)
    public
    payable
    {
        if(msg.value>=this.balance)
        {
            adr.transfer(this.balance+msg.value);
        }
    }
}
```

Trap

Bait

!! Balance = Previous Balance + Transaction Value !!

Multiplicator Honeypot

Etherscan
The Ethereum Block Explorer

LOGIN | Search by Address / Txhash / Block / Token / Ens | GO | Language

HOME | BLOCKCHAIN | TOKENS | RESOURCES | MORE

Contract 0x5aA88d2901

Home / Accounts / Address

Sponsored: W12 -1st place at World Blo

Contract Overview

Buy

Balance:

Ether Value:

Transactions:

0x58f3e1b933ab27…

Transactions | Internal Txns

Latest 4 txns

TxHash	Block	Age	From		To	Value	[TxFee]
0xbf4930b18953d0…	4750052	331 days 18 hrs ago	0x4b3df879896c90…	IN	0x5aa88d2901c68f…	0 Ether	0.0002713529
0x7d306043874efb…	4749394	331 days 21 hrs ago	0x22dc0138701299…	IN	0x5aa88d2901c68f…	1.1 Ether	0.0011632
0x4e4eb36057822e…	4569381	362 days 2 hrs ago	0x4b3df879896c90…	IN	0x5aa88d2901c68f…	1.00000000000001 Ether	0.0000002104
0x58f3e1b933ab27…	4569361	362 days 2 hrs ago	0x4b3df879896c90…	IN	Contract Creation	0 Ether	0.0000034583

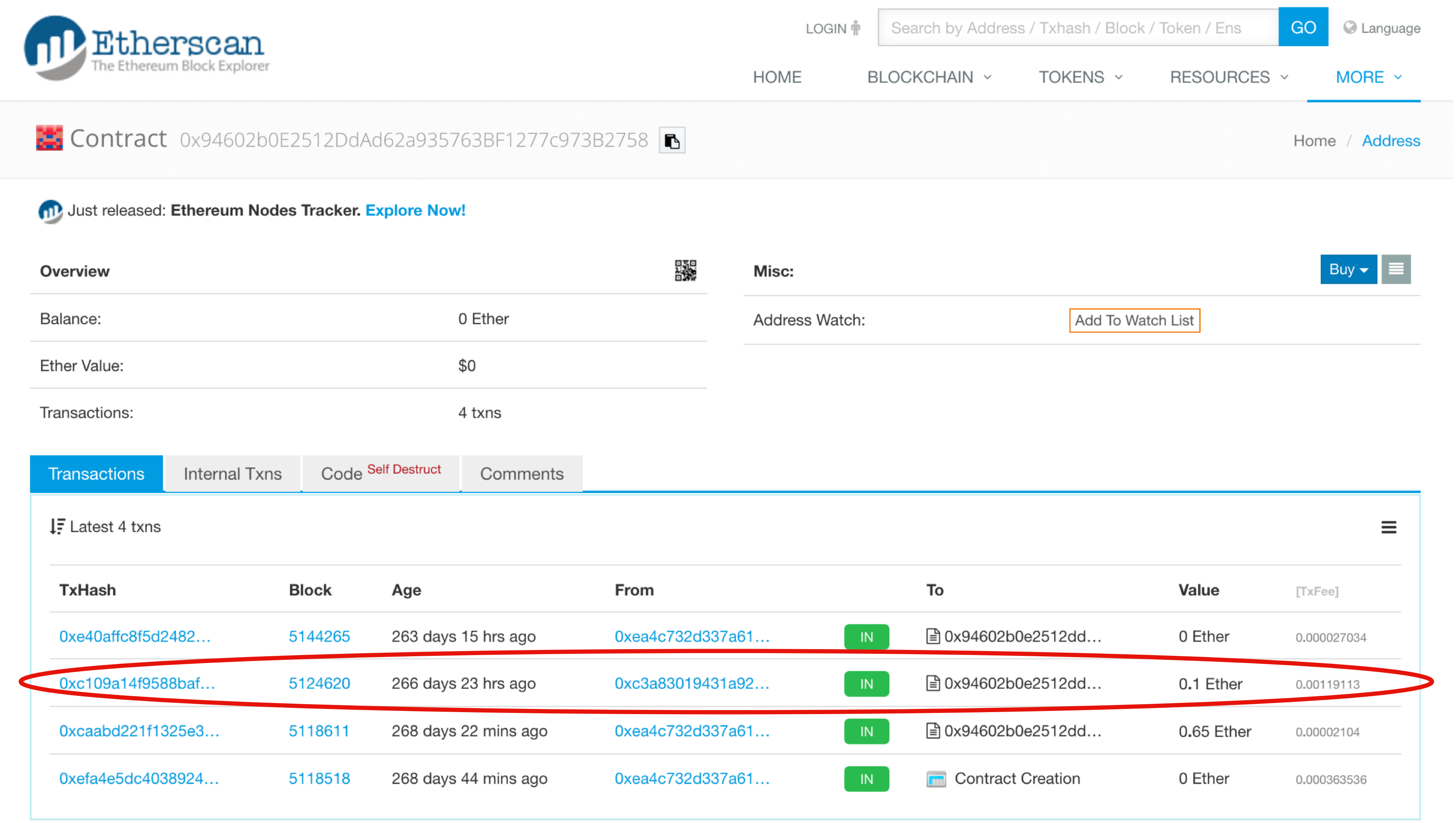
CryptoRoulette Honeypot

```
contract CryptoRoulette {

    uint256 private secretNumber;
    uint256 public lastPlayed;
    uint256 public betPrice = 0.1 ether;
    address public ownerAddr;

    struct Game {
        address player;
        uint256 number;
    }
    Game[] public gamesPlayed;

    function CryptoRoulette() public {
        ownerAddr = msg.sender;
        shuffle();
    }

    function shuffle() internal {
        // randomly set secretNumber with a value between 1 and 20
        secretNumber = uint8(sha3(now, block.blockhash(block.number-1))) % 20 + 1;
    }

    function play(uint256 number) payable public {
        require(msg.value >= betPrice && number <= 20);

        Game game;
        game.player = msg.sender; // this line
        game.number = number;
        gamesPlayed.push(game);


        if (number == secretNumber) {
            // win!
            msg.sender.transfer(this.balance);
        }

        shuffle();
        lastPlayed = now;
    }

    function kill() public {
        if (msg.sender == ownerAddr && now > lastPlayed + 1 days) {
            suicide(msg.sender);
        }
    }
```

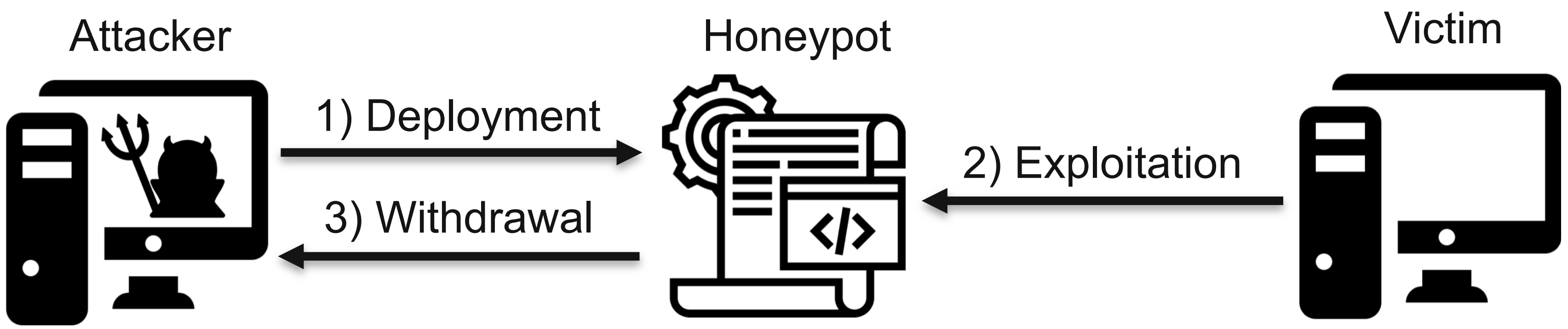
Trap

Bait

CryptoRoulette Honeypot



Honeypot Phases



Why Do Honeypots Work?

- People actively look for exploitable smart contracts.
- Complexity of the Ethereum ecosystem.

Detecting Honeypots

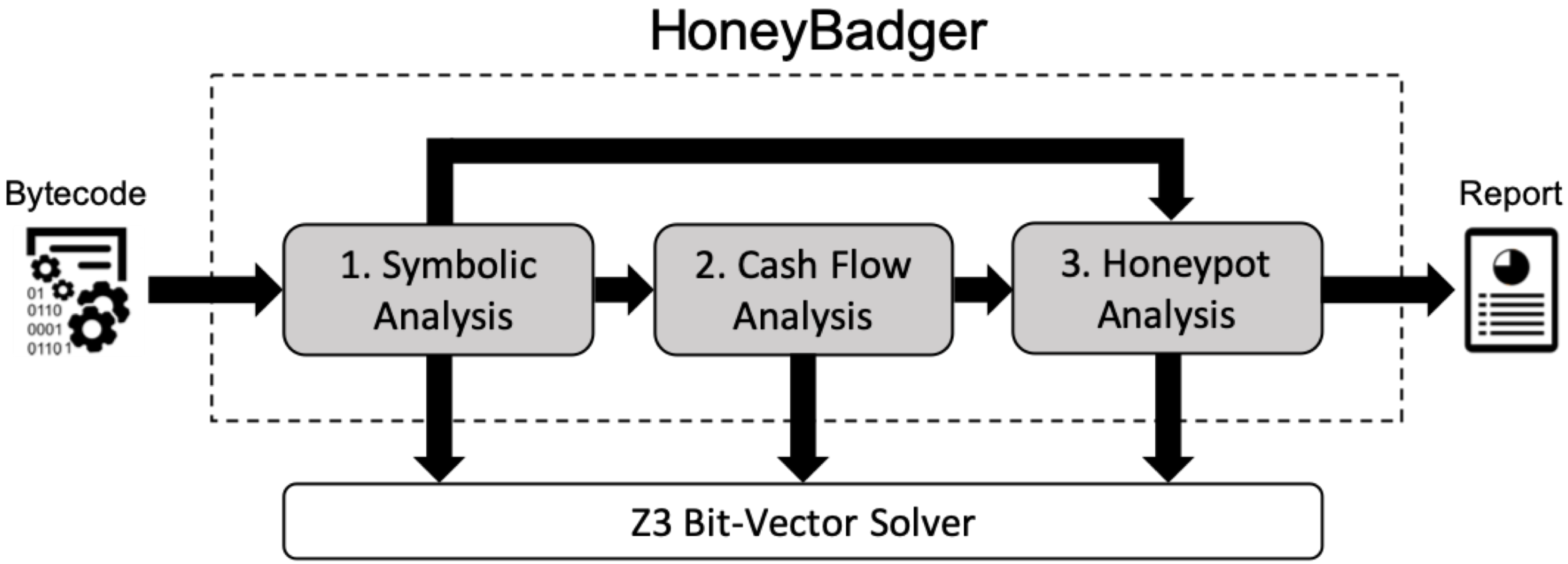
Taxonomy of Honeypots

- Collected **24** honeypot smart contracts from public sources on the Internet.
- Extracted **8** different techniques, each exploiting a feature (“bug”) on a particular level of Ethereum.

Level	Technique
Ethereum Virtual Machine	Balance Disorder
Solidity Compiler	Inheritance Disorder
	Skip Empty String Literal
	Type Deduction Overflow
	Uninitialised Struct
Etherscan Blockchain Explorer	Hidden State Update
	Hidden Transfer
	Straw Man Contract

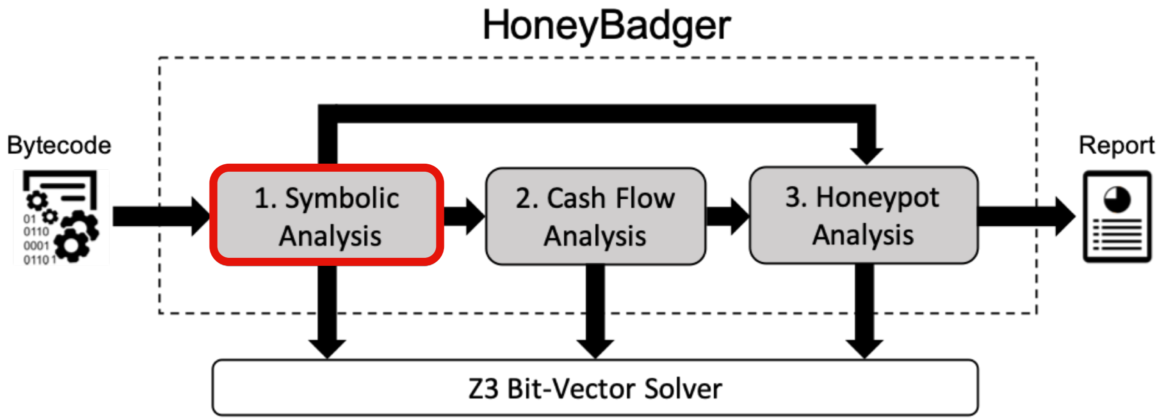
Table 1: A taxonomy of honeypot techniques in Ethereum smart contracts.

HoneyBadger Design



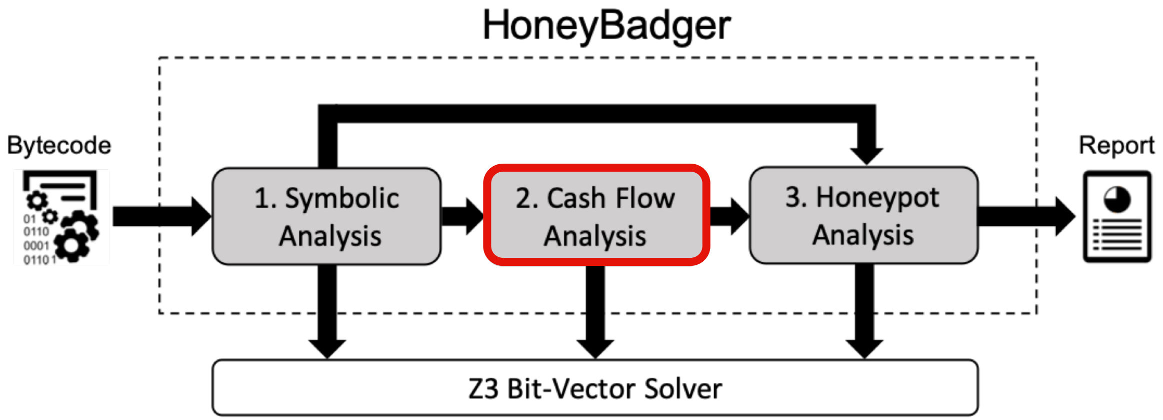
Symbolic Analysis

- Based on Luu et al.'s symbolic execution engine *Oyente* [CCS '16].
- Constructs control flow graph and executes every instruction symbolically.
- Our symbolic execution does not ignore infeasible paths.
- Collects meta information about:
  - Storage writes S
  - Infeasible basic blocks IB
  - Arithmetic operations A
  - Execution paths P
  - Feasible basic blocks FB
  - Contract calls C



Cash Flow Analysis

- Discard contracts that cannot receive and transfer funds.
- **Receiving** funds:
  - $\exists\, p \in P : REVERT \notin p$
  - We use Z3 to verify that $I_v > 0$ is satisfiable under p.
- **Transferring** funds:
  - Explicitly (e.g. *transfer*): $\exists\, c \in C:\ c_v > 0 \lor c_v\ is\ symbolic$
  - Implicitly (i.e. *selfdestruct*): $\exists\, p \in P : SELFDESTRUCT \in p$

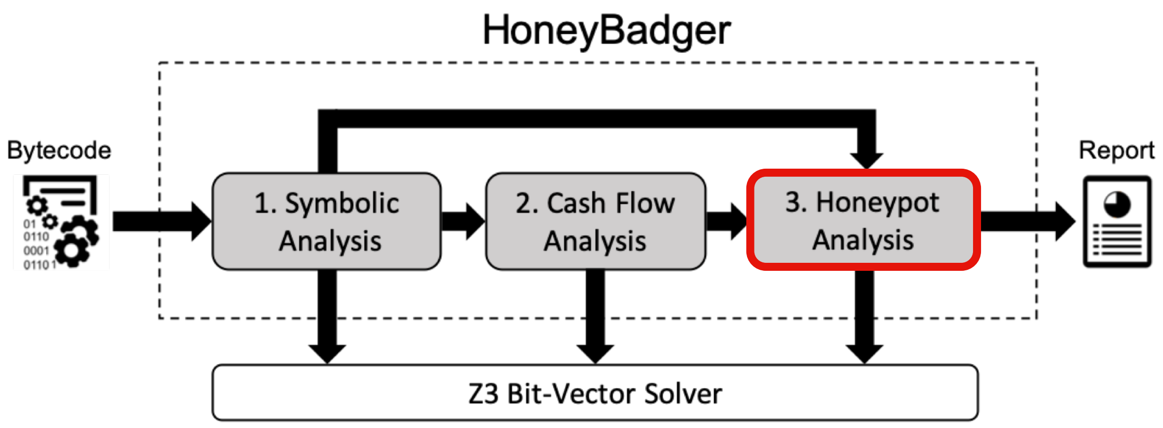


Honeypot Analysis

- Consists of several sub-components.
- Each sub-component is responsible for the detection of a particular technique.
- Honeypot techniques are detected via simple heuristics:
  - Ex.: Balance Disorder

$$\exists\, c \in C{:}\, c \in IB \land c_v = \sigma[I_a]_b + I_v$$

```
contract MultiplicatorX3 {
  ...
  function multiplicate(address adr) payable {
    if (msg.value >= this.balance)
      adr.transfer(this.balance+msg.value);
  }
}
```



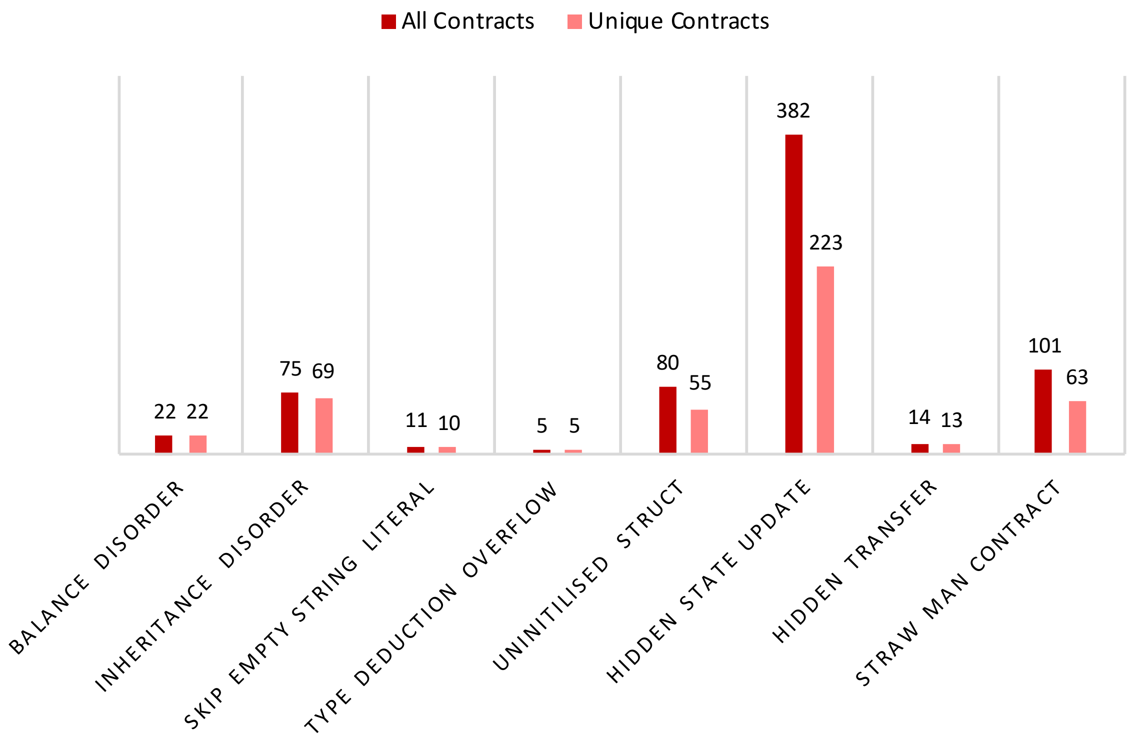
Evaluation

Dataset

- We crawled 2,019,434 contracts from August 7, 2015 to October 12, 2018.
- **151,935** contracts are unique in terms of bytecode (7.52%).
- We run HoneyBadger on the set of unique smart contracts.

Results

- 48,487 contracts have been identified as cash flow contracts (32%).
- Our tool detected **460** unique honeypots (690 on the 2 million).
- Analysis took about 2 minutes per contract (91% code coverage).



Validation

- Manual inspection of the source code for the 460 flagged contracts.
- We managed to collect the source code for 323 contracts (70% of 460).
- Validation shows that **282** contracts are true positives (87% precision).

	Balance Disorder	Inheritance Disorder	Skip Empty String Literal	Type Deduction Overflow	Uninitialised Struct	Hidden State Update	Hidden Transfer	Straw Man Contract
TP	20	41	9	4	32	134	12	30
FP	0	7	0	0	0	30	0	4
p	100	85	100	100	100	82	100	88

Table 2: Number of true positives (TP), false positives (FP) and precision *p* (in %) per detected honeypot technique for contracts with source code.

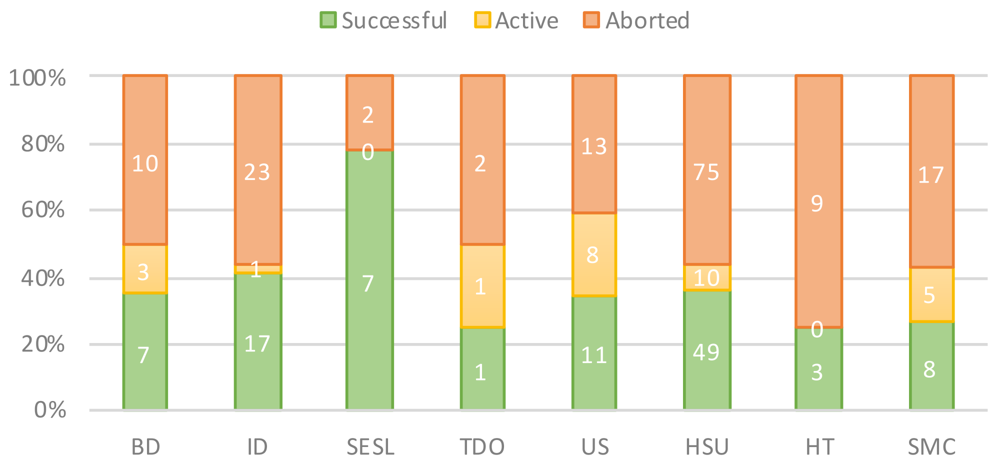
Honeypot Insights

Methodology

- Analyzed all transactions of the 282 true positives.
- Used simple heuristics to label addresses as:
  - **Attacker**:
    1) created the contract.
    2) first to send funds to the contract.
    3) received more funds than actually spent.
  - **Victim:** not labeled as attacker and spent more funds than actually received.
- Used this to label honeypots as:
  - **Successful:** a victim has been detected.
  - **Aborted:** the balance is zero and no victim has been detected.
  - **Active:** the balance is larger than zero and no victim has been detected.

Success Rate

- **71% manage to trap only one victim.**
  - Users potentially look at transactions.
- **Majority are successful within the first 24 hours.**
  - Users quickly attempt to exploit honeypots.



Diversity

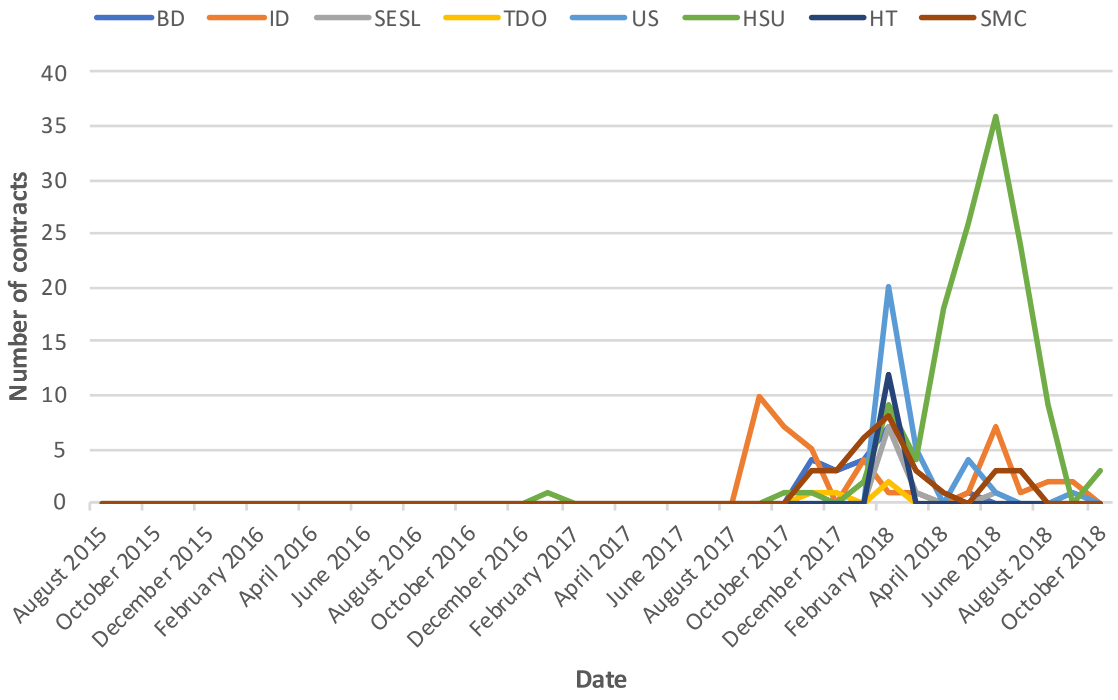
- **Bytecode of honeypots is vastly different even within the same technique.**
  - Signature-based detection methods are rather ineffective.

	BD	ID	SESL	TDO	US	HSU	HT	SMC
Min.	27	14	22	88	25	11	28	26
Max.	97	96	98	95	98	98	98	98
Mean	50	40	47	90	52	49	71	53
Mode	35	35	28	89	45	36	95	49

Table 3: Bytecode similarity (in %) per honeypot technique.

Popularity

- First deployment in January 2017 with highest activity in February 2018.
  - Honeypots are an emerging and increasing trend.



Profitability

- **A total profit of 257.25 ether has been made through honeypots.**
  - An accumulated profit of $90,118 at the time of withdrawal.

	Min.	Max.	Mean	Mode	Median	Sum
BD	0.01	1.13	0.5	0.11	0.11	3.5
ID	0.004	6.41	1.06	0.1	0.33	17.02
SESL	0.584	4.24	1.59	1.0	1.23	9.57
TDO	-	-	-	-	-	-
US	0.009	1.1	0.46	0.1	0.38	6.44
HSU	0.00002	11.96	1.44	0.1	1.02	171.22
HT	1.009	1.1	1.05	1.0	1.05	2.11
SMC	0.399	4.94	1.76	2.0	1.99	47.39
Overall	0.00002	11.96	1.35	1.0	1.01	257.25

Table 4: Statistics on the profitability of each honeypot technique in ether.

Conclusion

Conclusion

SnT

uni.lu UNIVERSITÉ DU LUXEMBOURG

- Honeypots are an emerging new type of fraud and requires further investigation.
- We propose a taxonomy and a tool called *HoneyBadger,* that detects honeypots at a large scale.
- We identified 690 honeypots with a precision of 87%.
- We provide interesting insights: 240 victims and $90,000 profit.

Questions?

All **code** & **data** is available on GitHub:

https://github.com/christoftorres/HoneyBadger

More information at:

https://honeybadger.uni.lu

Supported by:

Luxembourg National Research Fund