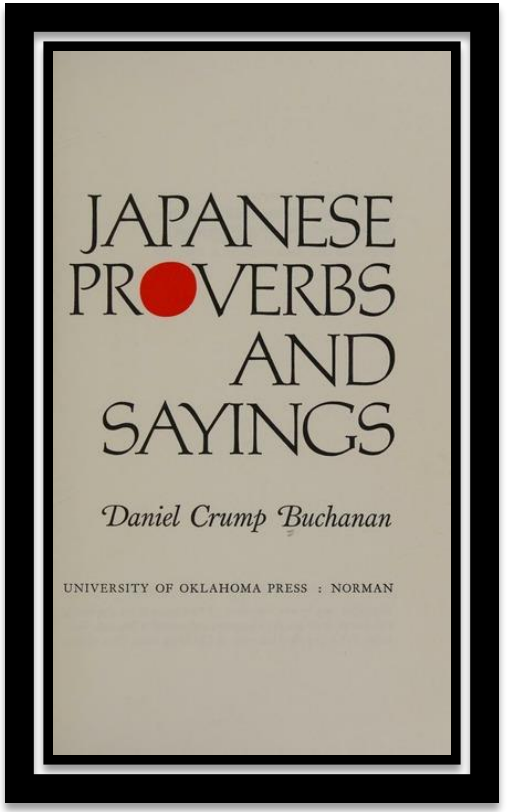
Uncovering Hidden Proxy Smart Contracts for Finding Collision Vulnerabilities in Ethereum

Cheng-Kang Chen, Wen-Yi Chu, **Muoi Tran**, Laurent Vanbever, Hsu-Chun Hsiao

IEEE ICDCS 2025
Glasgow, Scotland, UK
21 July 2025

Two swords cannot be in the same sheath

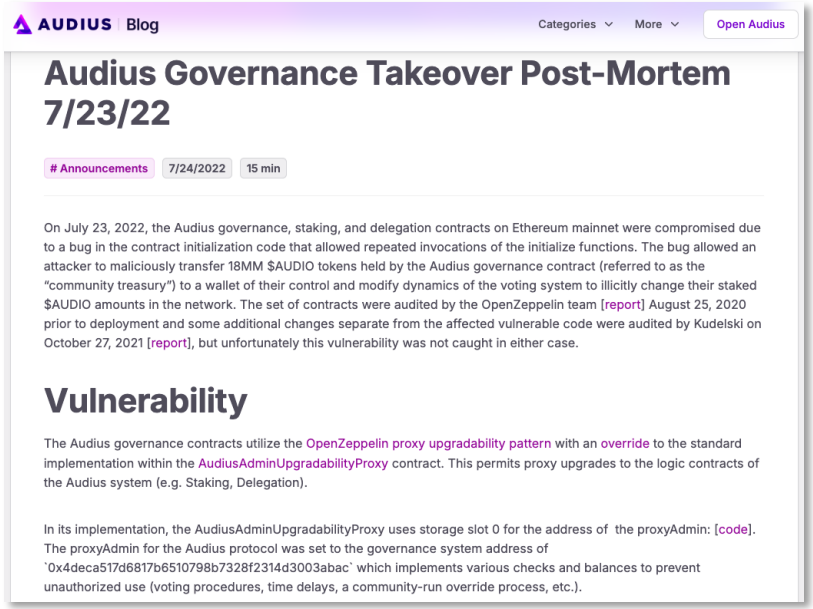


Two smart contracts cannot “be” in the same storage/function

Two smart contracts cannot “be” in the same storage/function

…or is it?

In 2022, Audius lost ~1 Million USDs due to a storage collision exploit



AUDIUS Blog

Categories More Open Audius

Audius Governance Takeover Post-Mortem 7/23/22

Announcements 7/24/2022 15 min

On July 23, 2022, the Audius governance, staking, and delegation contracts on Ethereum mainnet were compromised due to a bug in the contract initialization code that allowed repeated invocations of the initialize functions. The bug allowed an attacker to maliciously transfer 18MM $AUDIO tokens held by the Audius governance contract (referred to as the "community treasury") to a wallet of their control and modify dynamics of the voting system to illicitly change their staked $AUDIO amounts in the network. The set of contracts were audited by the OpenZeppelin team [report] August 25, 2020 prior to deployment and some additional changes separate from the affected vulnerable code were audited by Kudelski on October 27, 2021 [report], but unfortunately this vulnerability was not caught in either case.

Vulnerability

The Audius governance contracts utilize the OpenZeppelin proxy upgradability pattern with an override to the standard implementation within the AudiusAdminUpgradabilityProxy contract. This permits proxy upgrades to the logic contracts of the Audius system (e.g. Staking, Delegation).

In its implementation, the AudiusAdminUpgradabilityProxy uses storage slot 0 for the address of the proxyAdmin: [code]. The proxyAdmin for the Audius protocol was set to the governance system address of `0x4deca517d6817b6510798b7328f2314d3003abac` which implements various checks and balances to prevent unauthorized use (voting procedures, time delays, a community-run override process, etc.).

We will talk more about collision issues in Ethereum

We will talk more about collision issues in Ethereum

How do those collisions occur?

Background on function and storage collisions

We will talk more about collision issues in Ethereum

How do those collisions occur?

Background on function and storage collisions

How to detect those collision issues?

Description of a new tool with wider applicability

We will talk more about collision issues in Ethereum

How do those collisions occur?

Background on function and storage collisions

How to detect those collision issues?

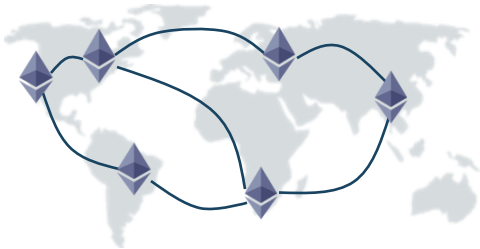
Description of a new tool with wider applicability

Do they affect existing smart contracts?

Insights from analysing all existing contracts

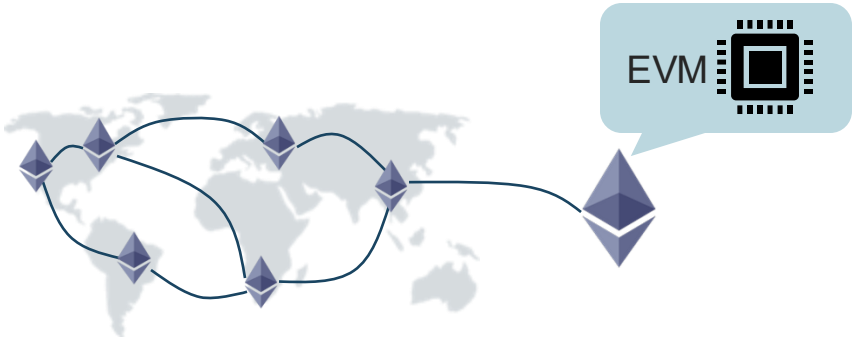
Background

Ethereum is a P2P network of nodes maintaining the blockchain



P2P network

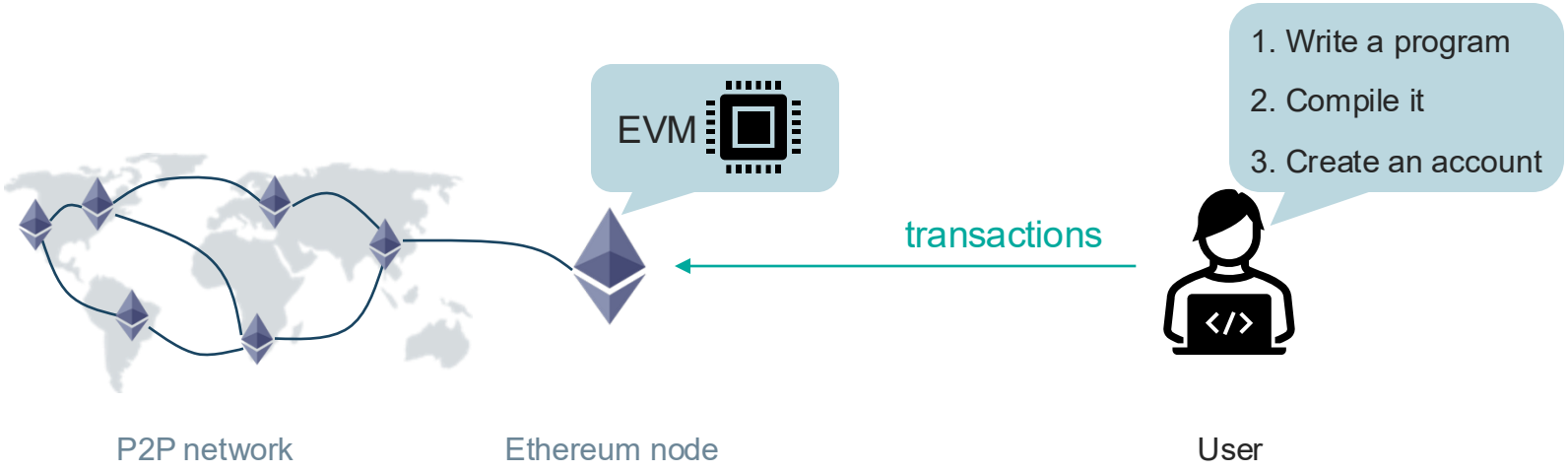
P2P nodes propagate and execute transactions within the EVM



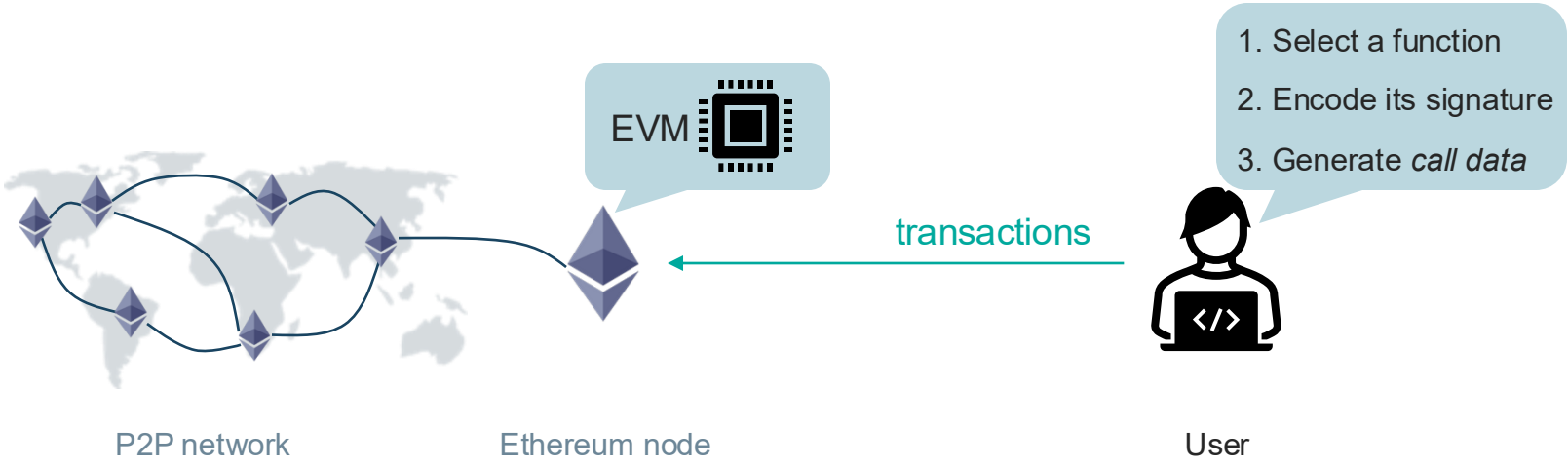
P2P network

Ethereum node

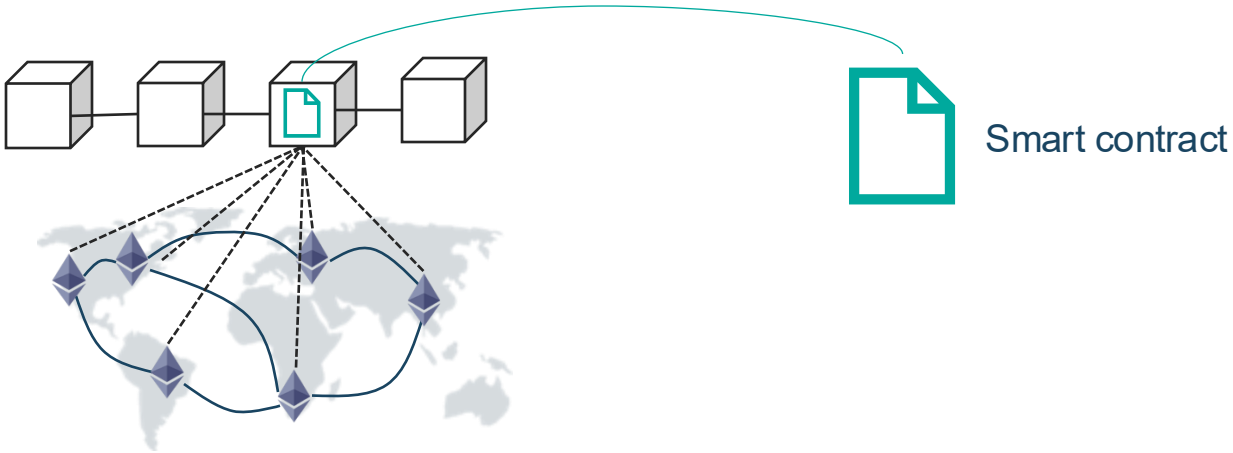
Transactions can be used to
(1) deploy smart contracts



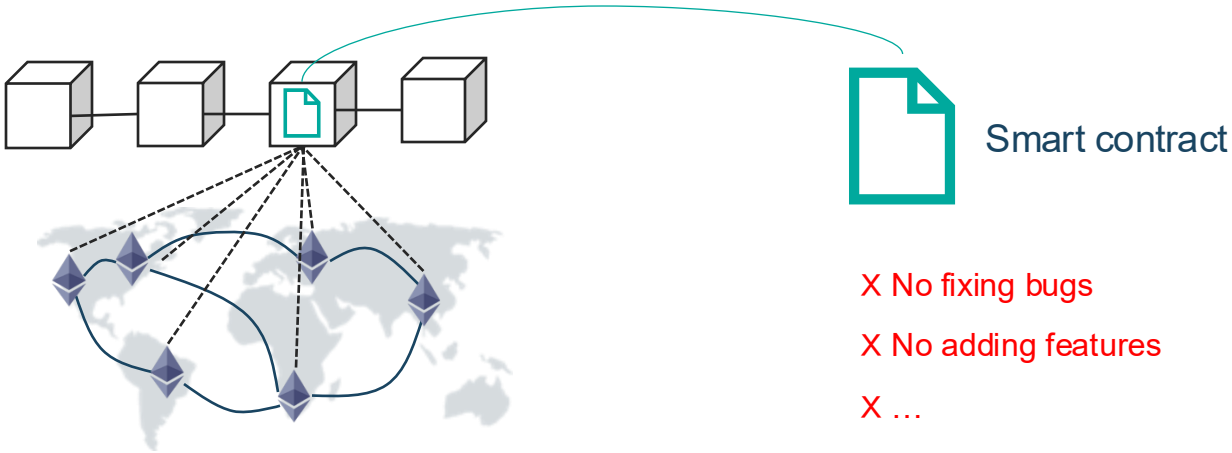
Transactions can also be used to
(2) execute deployed smart contracts



Smart contracts are immutable once deployed on the blockchain



Smart contracts are immutable once deployed on the blockchain



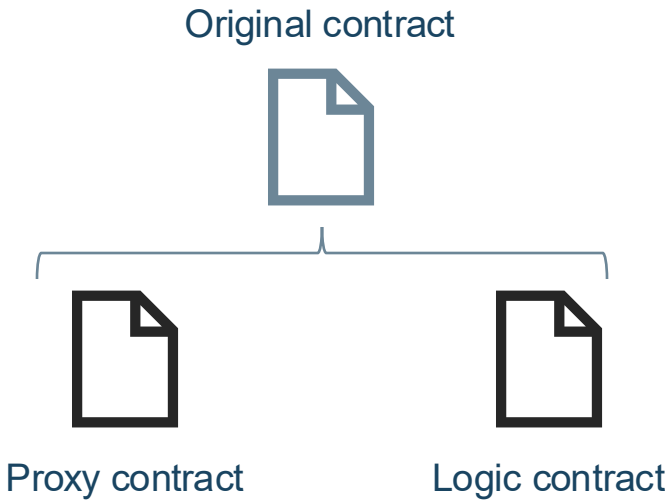
P2P network

X No fixing bugs

X No adding features

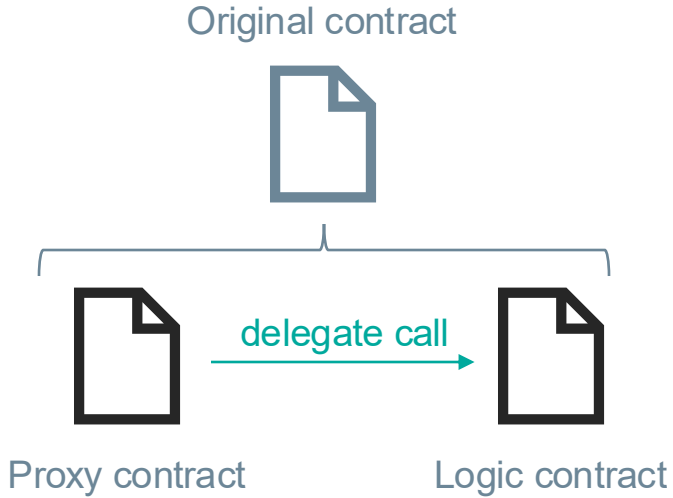
X …

Proxy design pattern enables upgradeability in smart contracts



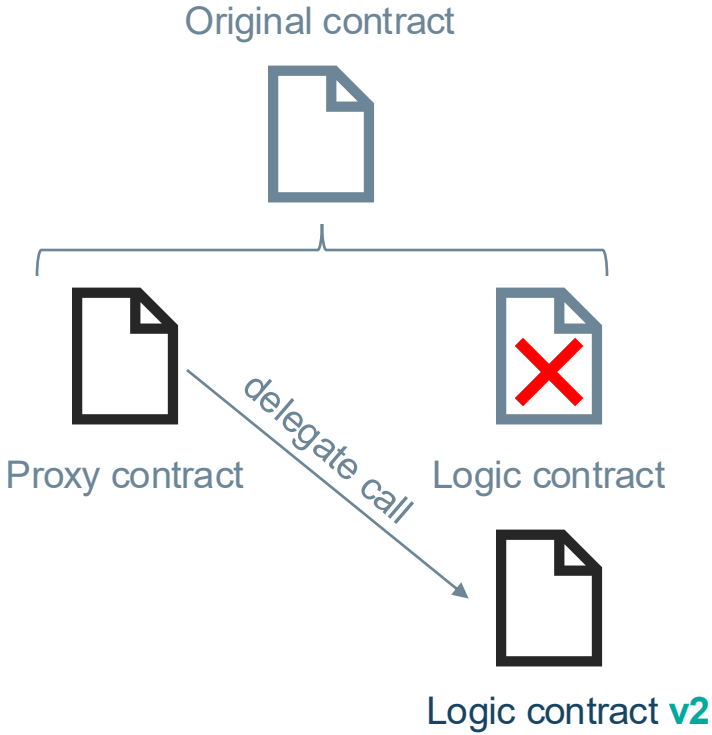
- Proxy contract contains data storage and logic contract contains implementation

Proxy design pattern enables upgradeability in smart contracts



- Proxy contract contains data storage and logic contract contains implementation
- Delegate calls link the two contracts: logic functions run using proxy's storage

Proxy design pattern enables upgradeability in smart contracts



- Proxy contract contains data storage and logic contract contains implementation
- Delegate calls link the two contracts: logic functions run using proxy's storage
- Upgrading now requires deploying a new logic contracts and updating into its address

An example of proxy smart contract

```
contract Proxy {
    address private logic;
    [...]

    function impl_() {
        [...]
    }

    fallback(bytes calldata input) {
        [...]
        logic.delegatecall(input)
    }
```

Storing the address of logic contract

An example of proxy smart contract

```
contract Proxy {
    address private logic;
    [...]

    function impl_() {
        [...]
    }

    fallback(bytes calldata input) {
        [...]
        logic.delegatecall(input)
    }
```

Storing the address of logic contract

If call data's selector doesn't match any function

Executing logic functions via delegate calls

What is the catch?

Functions may collide
when they have the same signature

```
contract Proxy {
   address private logic;
   [...]

   function impl_LUsXCWD2AKCc() {
      [stealing_fund_from_caller]
   }

   fallback(bytes calldata input) {
      [...]
      logic.delegatecall(input)
   }
}
```

```
contract Logic {
   [...]

   function free_eth_withdrawal() {
      [giving_ETH_to_caller]
   }
}
```

Functions may collide
when they have the same signature

```
contract Proxy {
    address private logic;
    [...]

    function impl_LUsXCWD2AKCc() {
        [stealing_fund_from_caller]
    }

    fallback(bytes calldata input) {
        [...]
        logic.delegatecall(input)
    }
}
```

```
contract Logic {
    [...]

    function free_eth_withdrawal() {
        [giving_ETH_to_caller]
    }
}
```

Having the same first 4 bytes in their hashes

Functions may collide
when they have the same signature

```
contract Proxy {
    address private logic;
    [...]

    function impl_LUsXCWD2AKCc() {
        [stealing_fund_from_caller]
    }

    fallback(bytes calldata input) {
        [...]
        logic.delegatecall(input)
    }
}
```

```
contract Logic {
    [...]

    function free_eth_withdrawal() {
        [giving_ETH_to_caller]
    }
}
```

Having the same first 4 bytes in their hashes

=> proxy contract's function is always selected!

Function collisions can be exploited in honeypot contracts

```
contract Proxy {
    address private logic;
    [...]

    function impl_LUsXCWD2AKCc() {
        [stealing_fund_from_caller]
    }

    fallback(bytes calldata input) {
        [...]
        logic.delegatecall(input)
    }
}
```

```
contract Logic {
    [...]

    function free_eth_withdrawal() {
        [giving_ETH_to_caller]
    }
}
```

Having the same first 4 bytes in their hashes

=> proxy contract's function is always selected!

Storage slots may collide
when they are used differently

```
contract Proxy {
   address private owner;
   [...]
   address private logic;

   [...]

   fallback(bytes calldata input) {
      [...]
      logic.delegatecall(input)
   }
}
```

```
contract Logic {
   bool private initialized;
   bool private initializing;

   function initialize() external{
     require (!initialized
          OR  initializing)
     initialized = true;
     initializing = false;
     owner = msg.sender;
   }
   [...]
}
```

Storage slots may collide
when they are used differently

```
contract Proxy {
   address private owner;
   [...]
   address private logic;

   [...]

   fallback(bytes calldata input) {
      [...]
      logic.delegatecall(input)
   }
}
```

Slot 0

```
contract Logic {
   bool private initialized;
   bool private initializing;

   function initialize() external{
     require (!initialized
          OR  initializing)
     initialized = true;
     initializing = false;
     owner = msg.sender;
   }
   [...]
}
```

Storage slots may collide
when they are used differently

```
contract Proxy {
   address private owner;
   [...]
   address private logic;

   [...]

   fallback(bytes calldata input) {
      [...]
      logic.delegatecall(input)
   }
}
```

Slot 0

```
contract Logic {
   bool private initialized;
   bool private initializing;

   function initialize() external{
     require (!initialized
          OR  initializing)
     initialized = true;
     initializing = false;
     owner = msg.sender;
   }
   [...]
}
```

Write to Slot 0

Storage slots may collide
when they are used differently

```
contract Proxy {
   address private owner;
   [...]
   address private logic;

   [...]

   fallback(bytes calldata input) {
      [...]
      logic.delegatecall(input)
   }
}
```

Slot 0

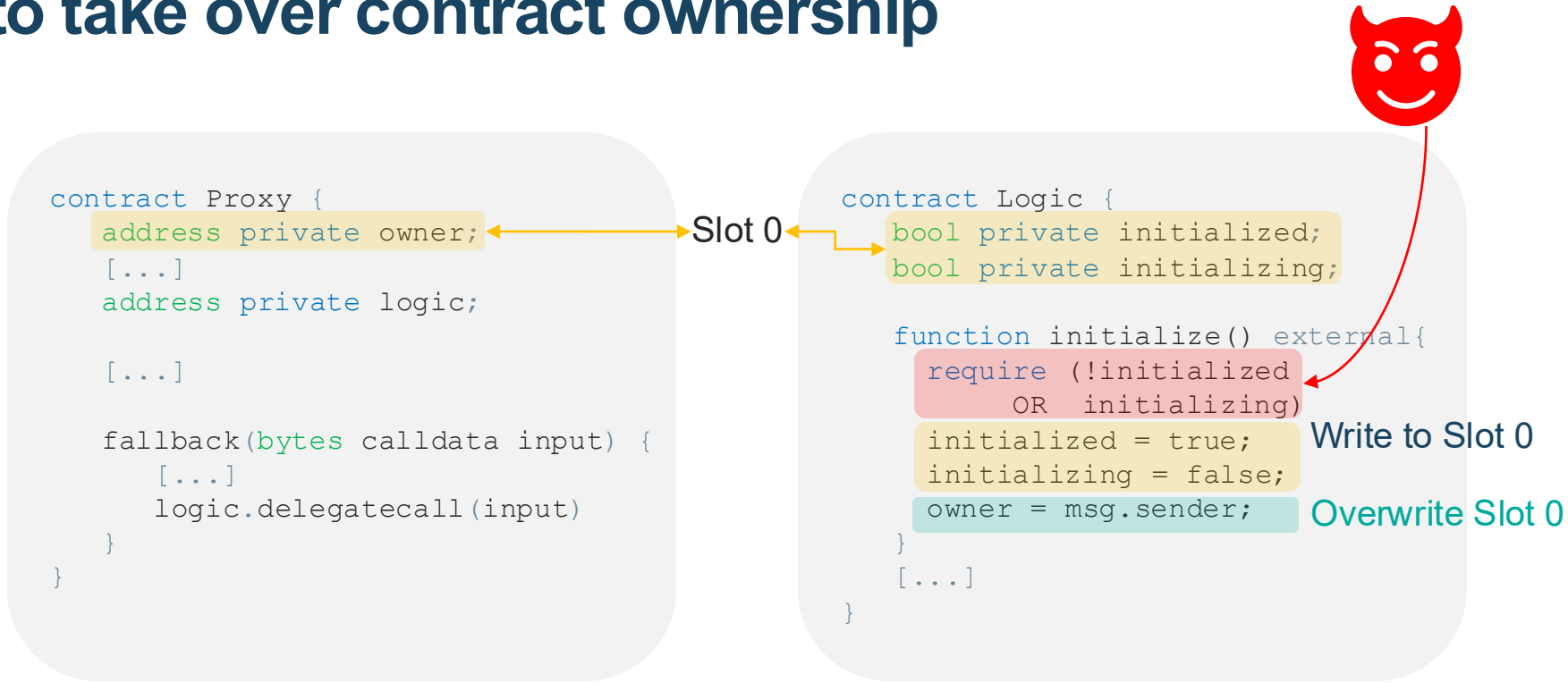
```
contract Logic {
   bool private initialized;
   bool private initializing;

   function initialize() external{
     require (!initialized
          OR  initializing)
     initialized = true;
     initializing = false;
     owner = msg.sender;
   }
   [...]
}
```

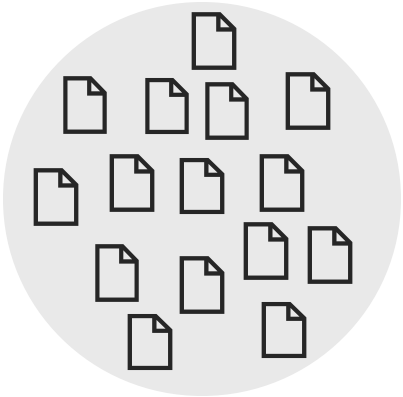
Write to Slot 0

Overwrite Slot 0

Storage slots can be exploited to take over contract ownership

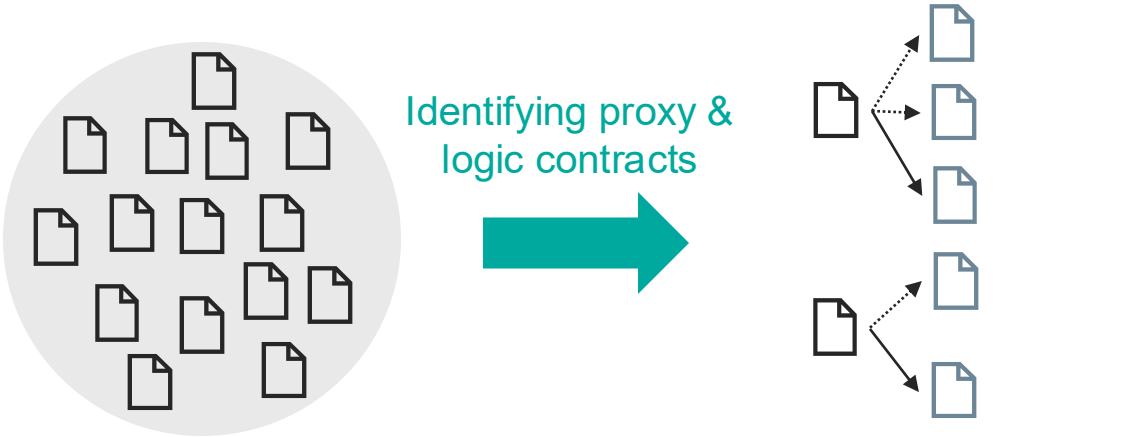


Existing tools typically detect collisions in two phases

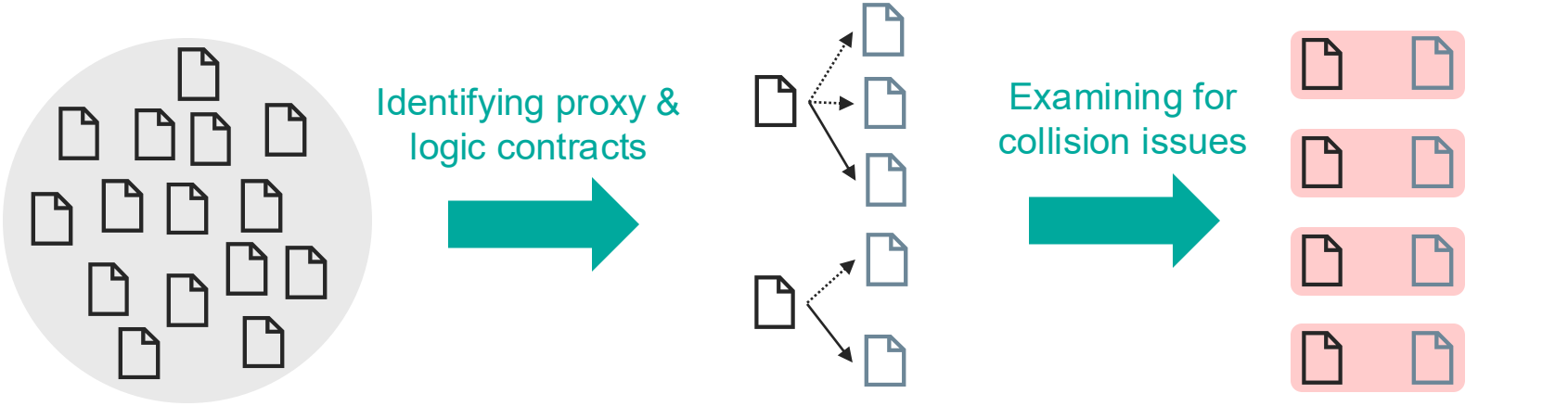


All smart contracts

Existing tools typically detect collisions in two phases



Existing tools typically detect collisions in two phases

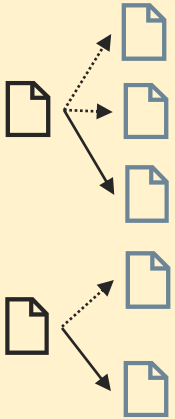


Existing tools typically detect collisions in two phases

Requiring contracts' **source code** or **historical data of transactions**

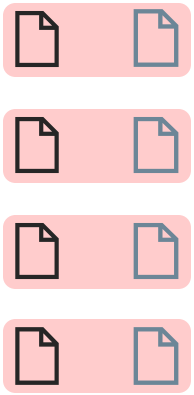
All smart contracts

Identifying proxy & logic contracts



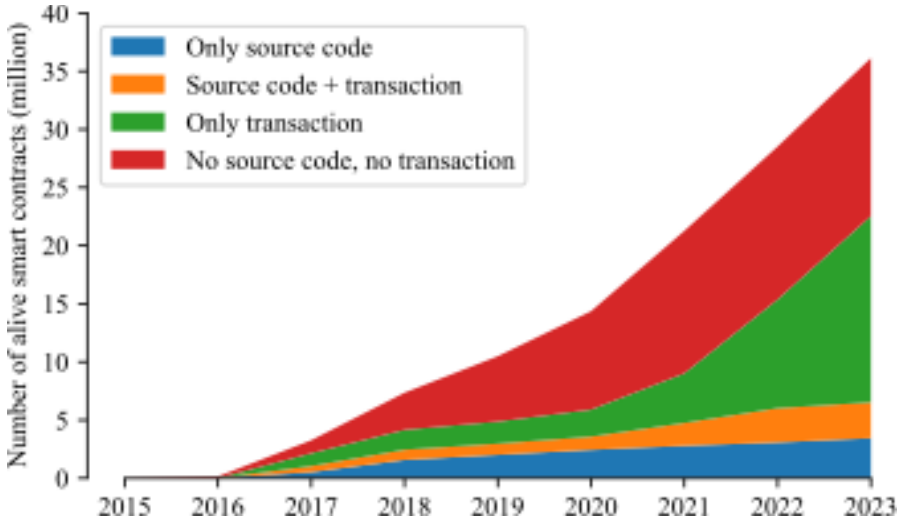
Pairs of proxy/logic contracts

Examining for collision issues

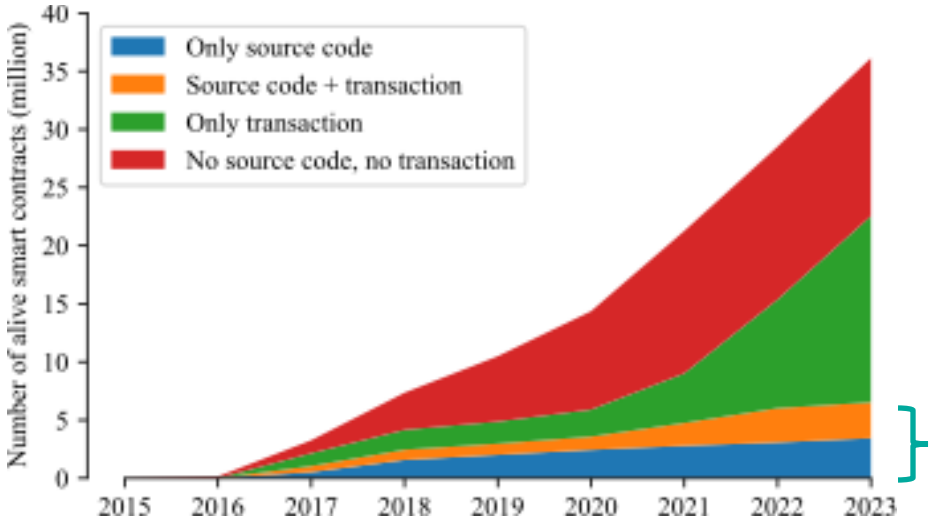


Function/Storage collisions

Challenges: Source code and historical transactions may be unavailable

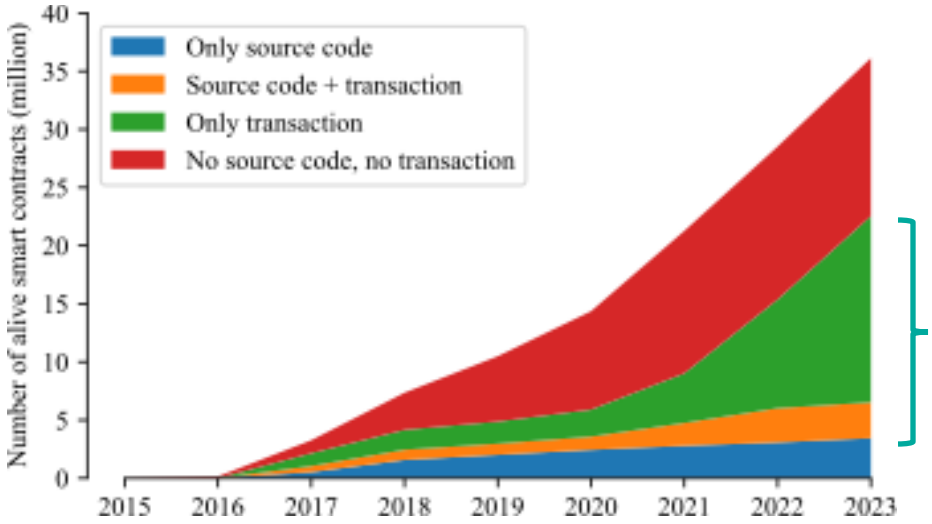


Challenges: Source code and historical transactions may be unavailable



18% contracts have source code available

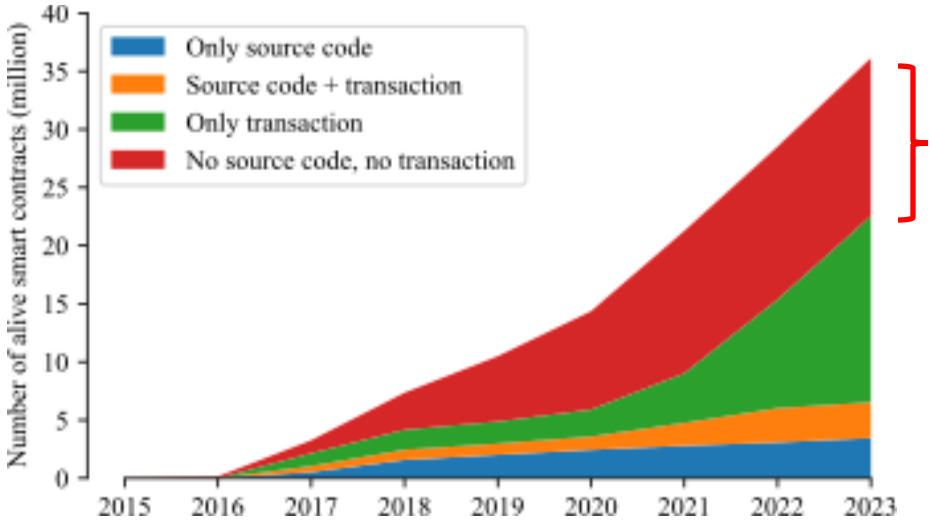
Challenges: Source code and historical transactions may be unavailable



53% contracts have transactions available

18% contracts have source code available

Challenges: Source code and historical transactions may be unavailable



"hidden" smart contracts

53% contracts have transactions available

18% contracts have source code available

Can we uncover *all* proxy contracts?

Proxion

Proxion uncovers more proxy contracts than previous works

	Smart contract coverage			
	With source code		Without source code	
	With tx	Without tx	With tx	Without tx
EtherScan	✅	✅		
Slither	✅	✅		
Salehi et al.	✅		✅	
USCDetector	✅		✅	
USCHunt	✅	✅		
CRUSH	✅		✅	
Proxion	✅	✅	✅	✅

Proxion also detects more collision issues as a result

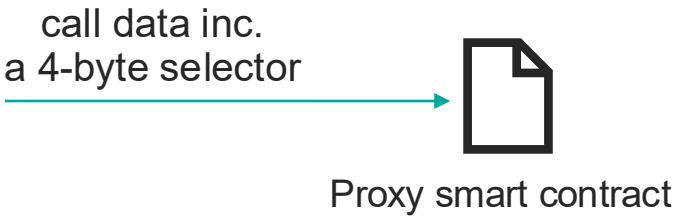
	Smart contract coverage				Collision coverage			
	With source code		Without source code		With source code		Without source code	
	With tx	Without tx	With tx	Without tx	Function	Storage	Function	Storage
EtherScan	✅	✅						
Slither	✅	✅			✅	✅		
Salehi et al.	✅		✅					
USCDetector	✅		✅					
USCHunt	✅	✅			✅	✅		
CRUSH	✅		✅			✅		✅
Proxion	✅	✅	✅	✅	✅	✅	✅	✅

Proxion's novel coverage is about hidden smart contracts

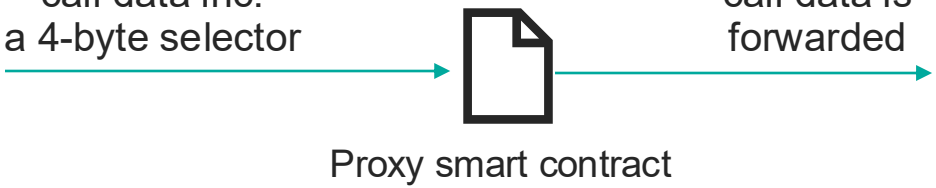
	Smart contract coverage				Collision coverage			
	With source code		Without source code		With source code		Without source code	
	With tx	Without tx	With tx	Without tx	Function	Storage	Function	Storage
EtherScan	✅	✅						
Slither	✅	✅			✅	✅		
Salehi et al.	✅		✅					
USCDetector	✅		✅					
USCHunt	✅	✅			✅	✅		
CRUSH	✅		✅			✅		✅
Proxion	✅	✅	✅	✅	✅	✅	✅	✅

Key idea: the distinct behaviors of proxy contracts can be triggered in EVM

- We trigger fallback function, by using a 4-byte selector matching no function

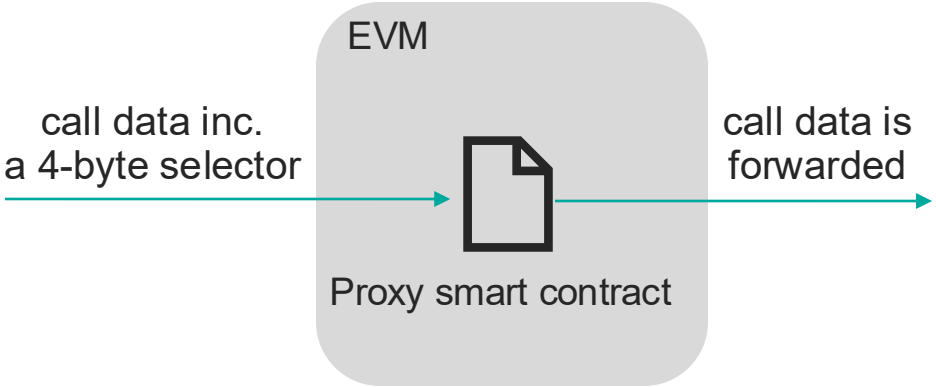


Key idea: the distinct behaviors of proxy contracts can be triggered in EVM



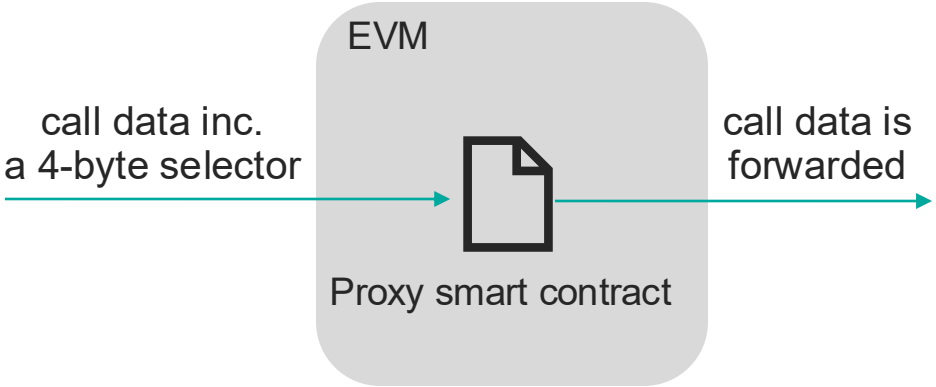
- We trigger fallback function, by using a 4-byte selector matching no function
- We observe if delegate calls trigger forwarding the transaction's call data

Key idea: the distinct behaviors of proxy contracts can be triggered in EVM



- We trigger fallback function, by using a 4-byte selector matching no function
- We observe if delegate calls trigger forwarding the transaction's call data
- These behaviors can be done in an (emulated) EVM

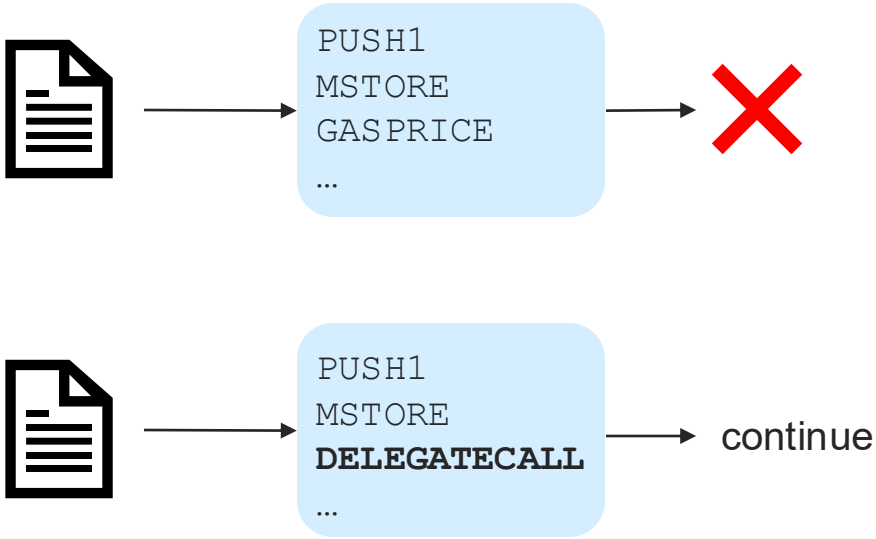
Key idea: the distinct behaviors of proxy contracts can be triggered in EVM



- We trigger fallback function, by using a 4-byte selector matching no function
- We observe if delegate calls trigger forwarding the transaction's call data
- These behaviors can be done in an (emulated) EVM

=> No source code analysis or real transactions are needed!

Given a smart contract, Proxion first disassembles it into opcodes

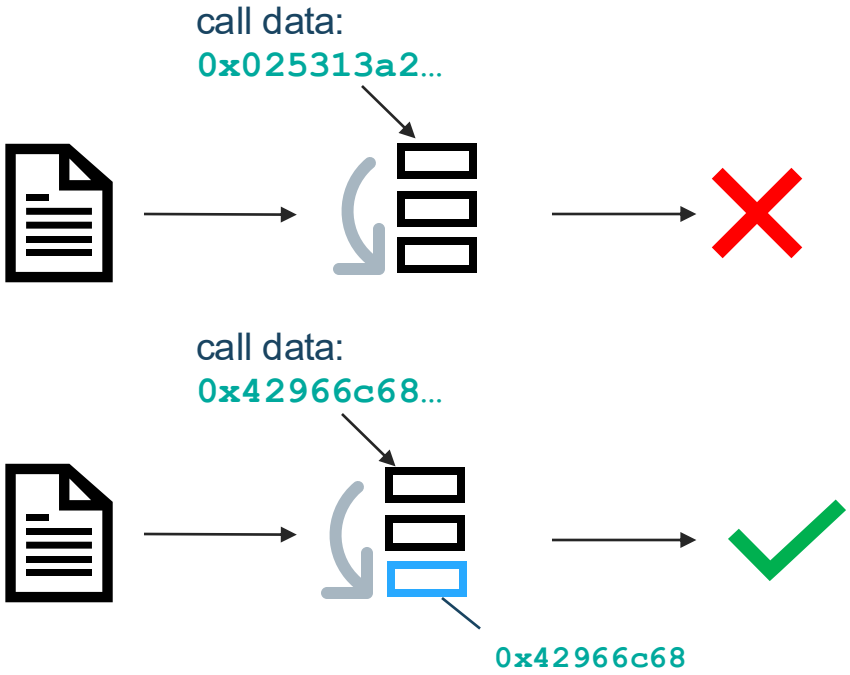


- Proxy contracts must contain a `DELETEGATECALL` opcode

Proxion then emulates EVM execution with generated *calldata*

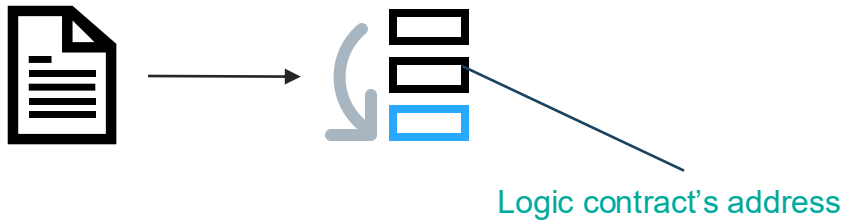
- To generate the calldata:
  - Identify *all* data following `PUSH4` opcodes
  - Avoid all these potential signatures
  - Generate random function selector

Proxion then emulates EVM execution with generated *calldata*



- To generate the calldata:
  - Identify *all* data following `PUSH4` opcodes
  - Avoid all these potential signatures
  - Generate random function selector
- Proxy contract must push calldata after executing DELEGATECALL instruction

Proxion then finds logic contracts associated with the proxy contracts



- The address of logic contract can also be found in the EVM stack
- All associated logic contracts in the past can be discovered in the bytecode or in the same identified storage slot
  - See our full paper for a heuristic

Proxion finally tests if a pair of proxy & logic contracts have collisions

- Storage collisions:
  - Proxion re-uses solutions from CRUSH[1]
  - (Proxion still covers more contracts)

[1]Not your Type! Detecting Storage Collision Vulnerabilities in Ethereum Smart Contracts, Ruaro et al., NDSS '24.

Proxion finally tests if a pair of proxy & logic contracts have collisions

```
0000  60  PUSH1 0x80
0002  60  PUSH1 0x40
0004  52  MSTORE
0005  34  CALLVALUE
0006  80  DUP1
0007  15  ISZERO
0008  60  PUSH1 0x0e
000A  57  *JUMPI
[...]
0018  35  CALLDATALOAD
0019  60  PUSH1 0xe0
001B  1C  SHR
001C  80  DUP1
001D  63  PUSH4 0xdf4a3106
0022  14  EQ
0023  60  PUSH1 0x2a
0025  57  *JUMPI
[...]
002A  5B  JUMPDEST
[...]
```

- Storage collisions:
  - Proxion re-uses solutions from CRUSH
  - (Proxion still covers more contracts)

- Function collisions:
  - Proxion analyses the opcodes to find potential function signatures

Proxion finally tests if a pair of proxy & logic contracts have collisions

```
0000  60  PUSH1 0x80
0002  60  PUSH1 0x40
0004  52  MSTORE
0005  34  CALLVALUE
0006  80  DUP1
0007  15  ISZERO
0008  60  PUSH1 0x0e
000A  57  *JUMPI
[...]
0018  35  CALLDATALOAD
0019  60  PUSH1 0xe0
001B  1C  SHR
001C  80  DUP1
001D  63  PUSH4 0xdf4a3106
0022  14  EQ
0023  60  PUSH1 0x2a
0025  57  *JUMPI
[...]
002A  5B  JUMPDEST
[...]
```

Pattern:

```
If matched_selector:
  jump to a function
```

- Storage collisions:
  - Proxion re-uses solutions from CRUSH
  - (Proxion still covers more contracts)
- Function collisions:
  - Proxion analyses the opcodes to find potential function signatures

Proxion finally tests if a pair of proxy & logic contracts have collisions

```
0000  60  PUSH1 0x80
0002  60  PUSH1 0x40
0004  52  MSTORE
0005  34  CALLVALUE
0006  80  DUP1
0007  15  ISZERO
0008  60  PUSH1 0x0e
000A  57  *JUMPI
[...]
0018  35  CALLDATALOAD
0019  60  PUSH1 0xe0
001B  1C  SHR
001C  80  DUP1
001D  63  PUSH4 0xdf4a3106
0022  14  EQ
0023  60  PUSH1 0x2a
0025  57  *JUMPI
[...]
002A  5B  JUMPDEST
[...]
```

Function signature

Pattern:

```
If matched_selector:
  jump to a function
```

- Storage collisions:
  - Proxion re-uses solutions from CRUSH
  - (Proxion still covers more contracts)
- Function collisions:
  - Proxion analyses the opcodes to find potential function signatures

Proxion finally tests if a pair of proxy & logic contracts have collisions

```
0000  60  PUSH1 0x80
0002  60  PUSH1 0x40
0004  52  MSTORE
0005  34  CALLVALUE
0006  80  DUP1
0007  15  ISZERO
0008  60  PUSH1 0x0e
000A  57  *JUMPI
[...]
0018  35  CALLDATALOAD
0019  60  PUSH1 0xe0
001B  1C  SHR
001C  80  DUP1
001D  63  PUSH4 0xdf4a3106
0022  14  EQ
0023  60  PUSH1 0x2a
0025  57  *JUMPI
[...]
002A  5B  JUMPDEST
[...]
```

Function signature

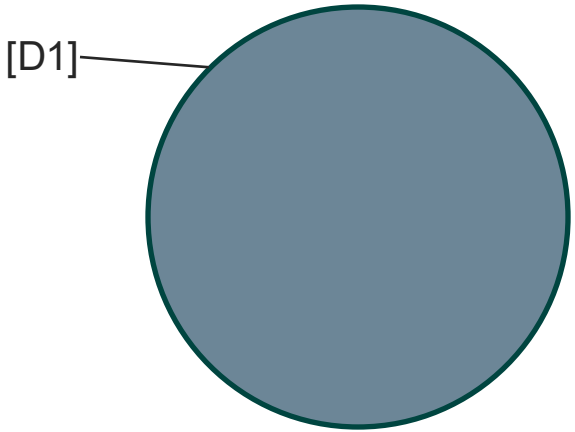
Pattern:

```
If matched_selector:
  jump to a function
```

- Storage collisions:
  - Proxion re-uses solutions from CRUSH
  - (Proxion still covers more contracts)
- Function collisions:
  - Proxion analyses the opcodes to find potential function signatures
  - Collision occurs when a function signature appears in both contracts

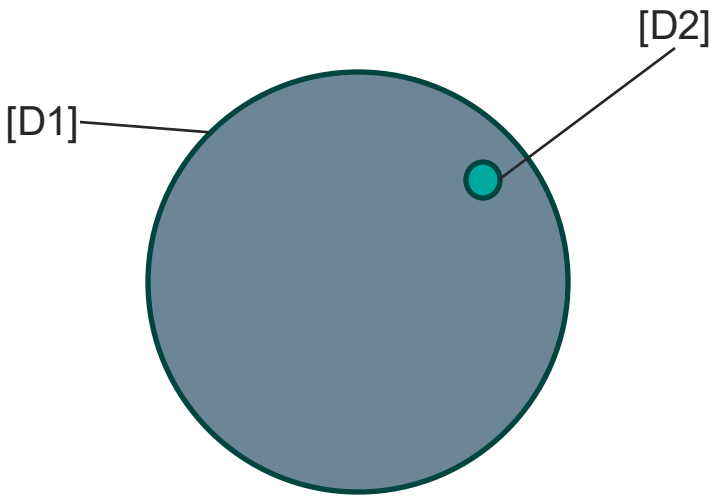
Results

We use different datasets collected independently



- Dataset **[D1]** (2015 – Oct 2023)
  - 36.1M active contracts

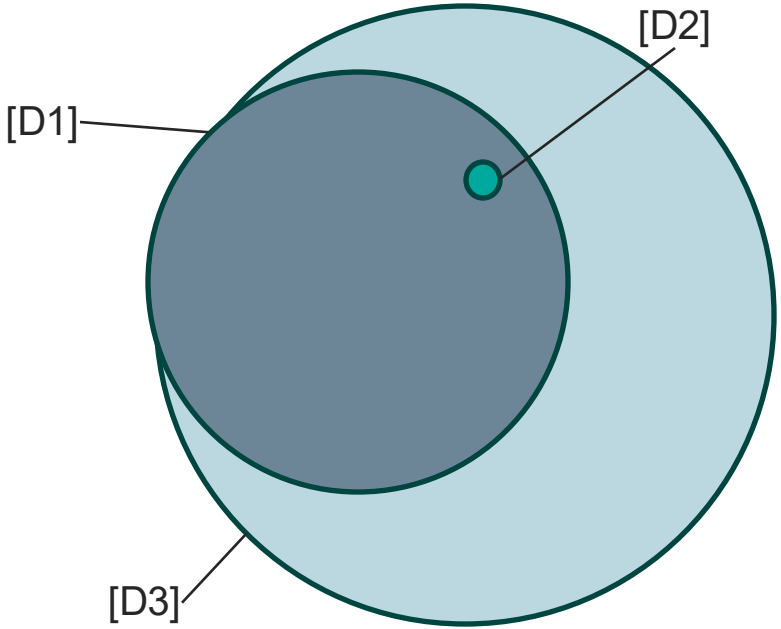
We use different datasets collected independently



- Dataset **[D1]** (2015 – Oct 2023)
  - 36.1M active contracts
- Dataset **[D2]** (2017 – 2022)
  - 330K contracts with source code available
  - Used by USCHunt[1]

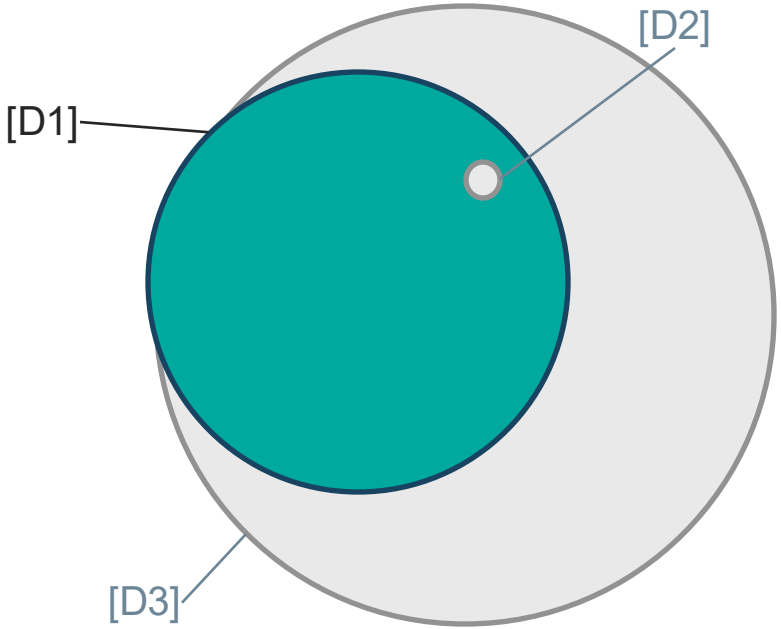
[1]Proxy Hunting: Understanding and Characterizing Proxy-based Upgradeable Smart Contracts in Blockchains, Bodell et al., Usenix Sec '23.

We use different datasets collected independently



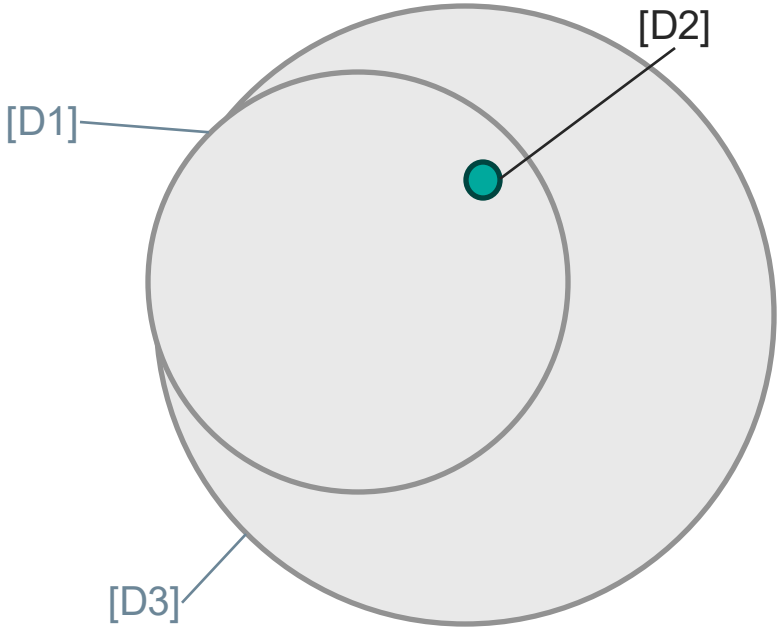
- Dataset **[D1]** (2015 – Oct 2023)
  - 36.1M active contracts
- Dataset **[D2]** (2017 – 2022)
  - 330K contracts with source code available
  - Used by USCHunt
- Dataset **[D3]** (2015 – April 2023)
  - All 53.5M contracts
  - Used by CRUSH

Proxion is effective in identifying proxy smart contracts



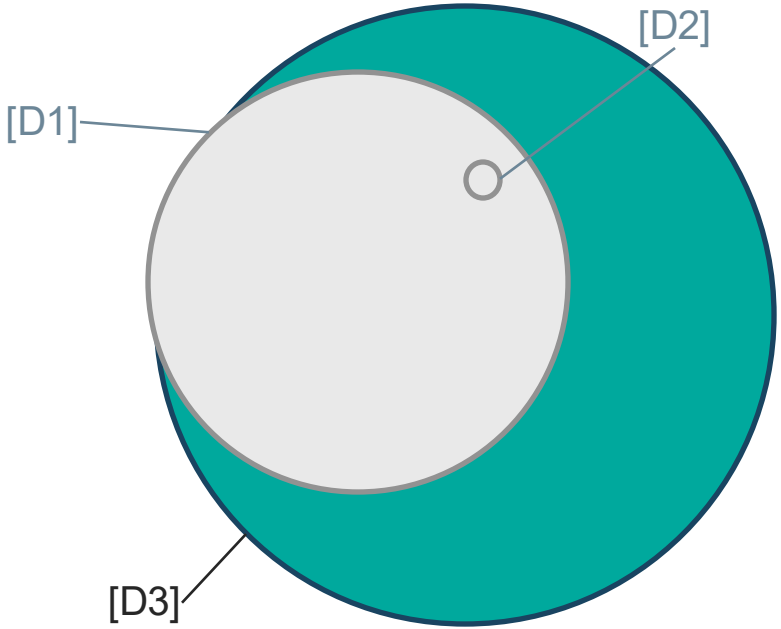
- Dataset **[D1]**
  - 54.2% contracts are proxy
  - 1.5M proxy contracts are hidden

Proxion is effective in identifying proxy smart contracts



- Dataset **[D1]**
  - 54.2% contracts are proxy
  - 1.5M proxy contracts are hidden
- Dataset **[D2]**
  - +7,000 contracts compared to USCHunt

Proxion is effective in identifying proxy smart contracts



- Dataset **[D1]**
  - 54.2% contracts are proxy
  - 1.5M proxy contracts are hidden
- Dataset **[D2]**
  - +7,000 contracts compared to USCHunt
- Dataset **[D3]**
  - +1.6M contracts compared to CRUSH

Proxion is accurate in detecting collision issues

		TP	FP	TN	FN	Accuracy
Storage collision	USCHunt	33	83	79	11	54.4%
	CRUSH	26	76	86	18	54.4%
	Proxion	27	28	134	17	78.2%

Proxion is accurate in detecting collision issues

		TP	FP	TN	FN	Accuracy
Storage collision	USCHunt	33	83	79	11	54.4%
	CRUSH	26	76	86	18	54.4%
	Proxion	27	28	134	17	78.2%

Higher accuracy than prior works

Proxion is accurate in detecting collision issues

		TP	FP	TN	FN	Accuracy
Storage collision	USCHunt	33	83	79	11	54.4%
	CRUSH	26	76	86	18	54.4%
	Proxion	27	28	134	17	78.2%

Same collision detector but different proxy identifications

Proxion is accurate in detecting collision issues

		TP	FP	TN	FN	Accuracy
Storage collision	USCHunt	33	83	79	11	54.4%
	CRUSH	26	76	86	18	54.4%
	Proxion	27	28	134	17	78.2%
Function collision	USCHunt	299	1	0	261	53.3%
	Proxion	557	0	1	3	99.5%

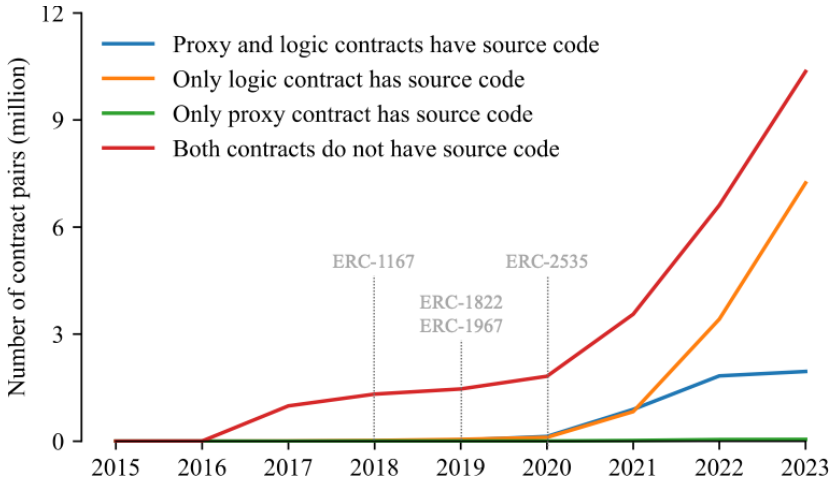
Proxion is accurate in detecting collision issues

		TP	FP	TN	FN	Accuracy
Storage collision	USCHunt	33	83	79	11	54.4%
	CRUSH	26	76	86	18	54.4%
	Proxion	27	28	134	17	78.2%
Function collision	USCHunt	299	1	0	261	53.3%
	Proxion	557	0	1	3	99.5%

No false positive

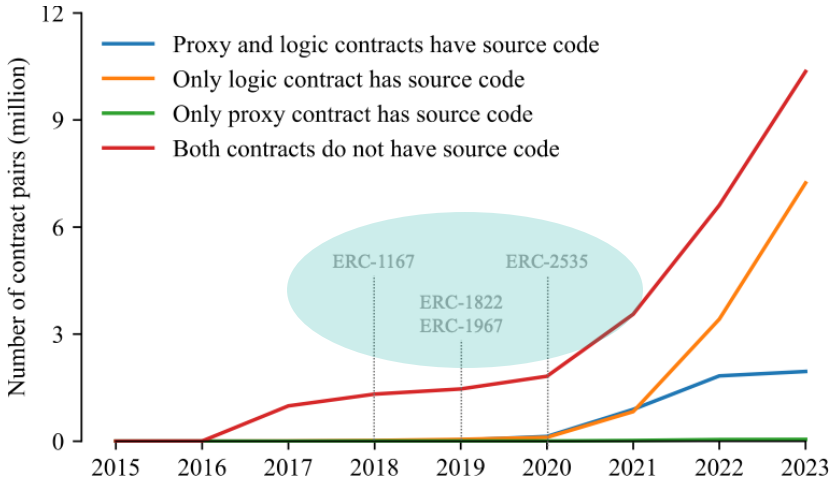
Higher accuracy

Proxion reveals insights about proxy smart contracts' landscape



- Over half of the active contracts are proxy or logic contracts
- The majority of proxy contracts do not publish their source code

Proxion reveals insights about proxy smart contracts' landscape



- Over half of the active contracts are proxy or logic contracts
- The majority of proxy contracts do not publish their source code
- The growth shows demand, testing, and mainstream periods of proxy contracts

Proxion reveals insights about proxy smart contracts' landscape

Year	Function collision
2017	24
2018	5,341
2019	16,136
2020	28,448
2021	705,801
2022	808,493
2023	2,541
Total	**1,566,784**

- 98.7% of function collisions are duplicated from the `OwnableDelegateProxy` contract

Proxion reveals insights about proxy smart contracts' landscape

Year	Function collision	Storage collision
2017	24	0
2018	5,341	7
2019	16,136	37
2020	28,448	34
2021	705,801	725
2022	808,493	2082
2023	2,541	137
Total	**1,566,784**	**3,022**

- 98.7% of function collisions are duplicated from the `OwnableDelegateProxy` contract
- 3,000 storage collisions are exploitable, affecting several staking entities like Compound, Curve, Poly,…

Takeaways

How do function and storage collisions occur?

Due to the emerging proxy contract setups.

Takeaways

How do function and storage collisions occur?

Due to the emerging proxy contract setups.

How to detect those collision issues?

Use Proxion to uncover all proxy smart contracts!

Takeaways

How do function and storage collisions occur?

Due to the emerging proxy contract setups.

How to detect those collision issues?

Use Proxion to uncover all proxy smart contracts!

Do they affect existing smart contracts?

Yes! Millions are already vulnerable (and counting).

Last but not least:
We are hiring PhDs & Postdoc!

… or mail to *muoi@chalmers.se*

CHALMERS

UNIVERSITY OF TECHNOLOGY