Abusing the Ethereum Smart Contract Verification Services for Fun and Profit

Pengxiang Ma*[1], **Ningyu He*** [2], Yuhua Huang [1], Haoyu Wang [1], Xiapu Luo [3]

[1] Huazhong University of Science and Technology, China

[2] Peking University, China

[3] The Hong Kong Polytechnic University, China

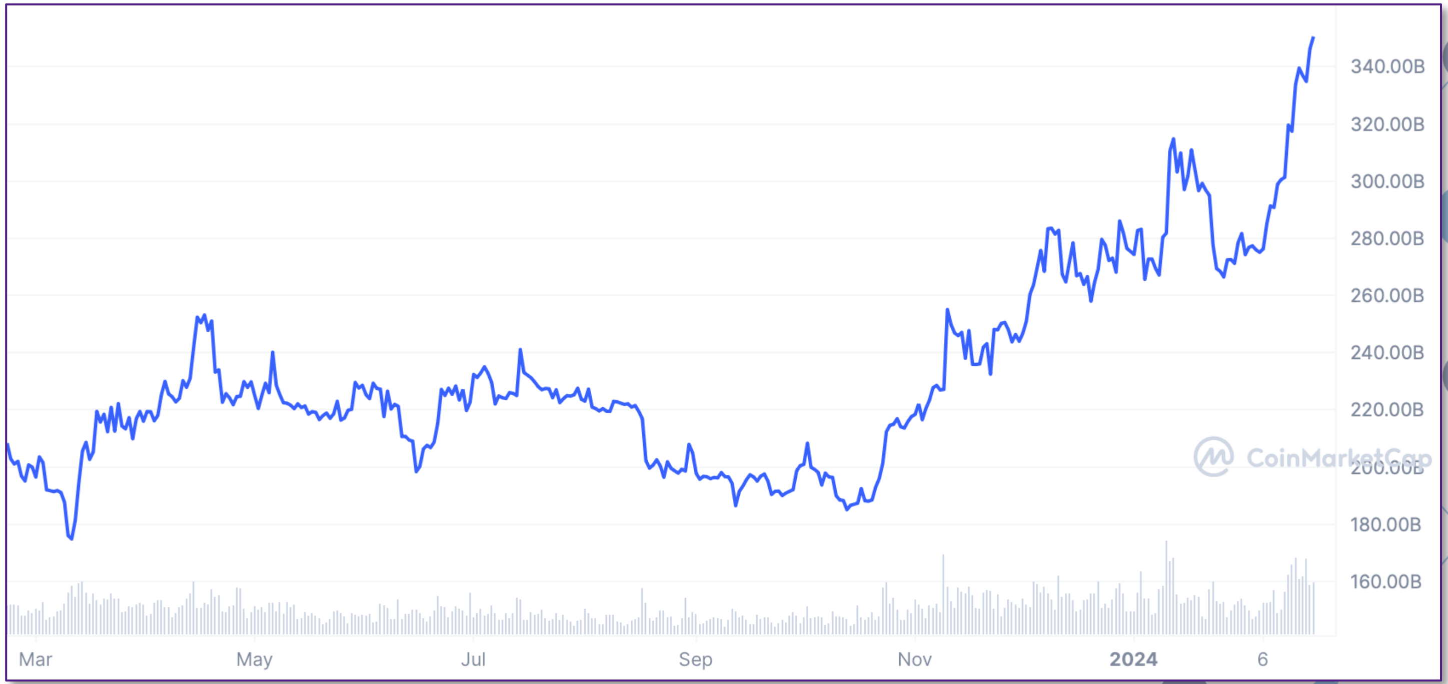
NDSS SYMPOSIUM/2024

Presented by Internet Society

#NDSSSymposium2024

Why Ethereum?

- Market cap of Ethereum has reached 340 billion USD.
- Smart contract is the killer application for Ethereum.



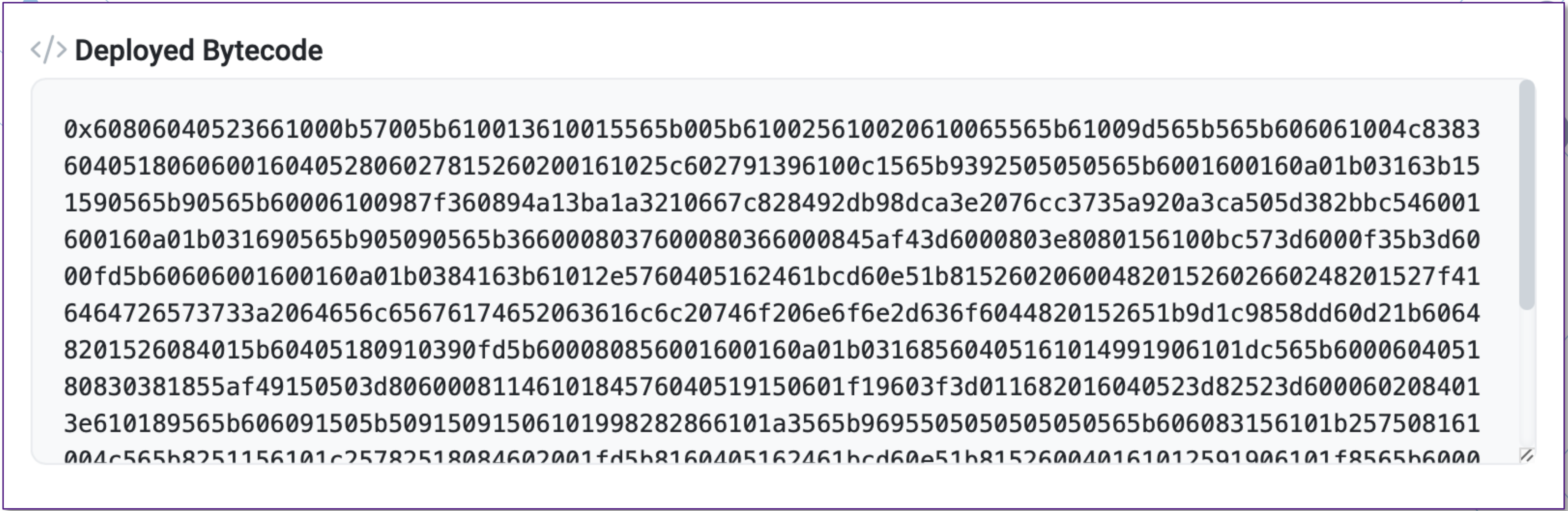
The market cap of Ethereum.

UNISWAP

MAKER

More than tens of millions of smart contracts are deployed on Ethereum!

Ethereum smart contract

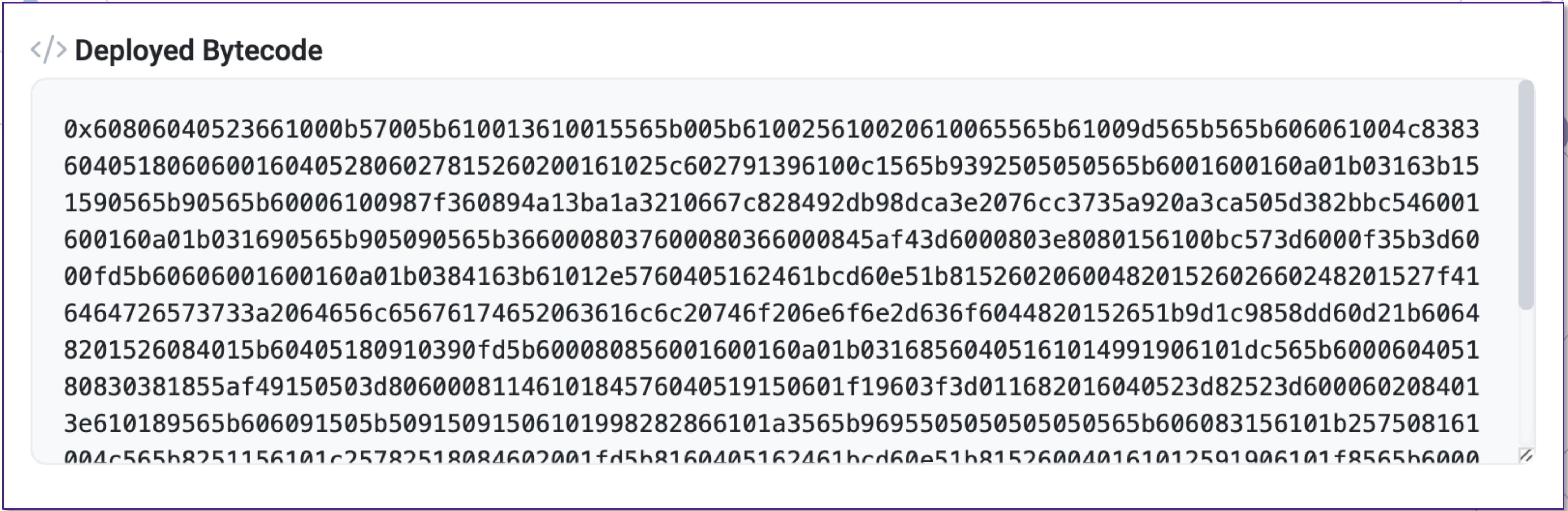


Deployed Bytecode

```
0x60806040523661000b57005b610013610015565b005b610025610020610065565b61009d565b565b606061004c8383
60405180606001604052806027815260200161025c602791396100c1565b939250505050565b6001600160a01b03163b15
1590565b90565b60006100987f360894a13ba1a3210667c828492db98dca3e2076cc3735a920a3ca505d382bbc546001
600160a01b031690565b905090565b3660008037600080366000845af43d6000803e8080156100bc573d6000f35b3d60
00fd5b60606001600160a01b0384163b61012e5760405162461bcd60e51b815260206004820152602660248201527f41
6464726573733a2064656c65676174652063616c6c20746f206e6f6e2d636f6044820152651b9d1c9858dd60d21b6064
8201526084015b60405180910390fd5b600080856001600160a01b03168560405161014991906101dc565b6000604051
80830381855af49150503d8060008114610184576040519150601f19603f3d011682016040523d82523d600060208401
3e610189565b606091505b50915091506101998282866101a3565b9695505050505050565b606083156101b257508161
004c565b8251156101c25782518084602001fd5b8160405162461bcd60e51b815260040161012591906101f8565b6000
```

A piece of deployed smart contract, stored on-chain in the bytecode format.

Ethereum smart contract



Deployed Bytecode

```
0x60806040523661000b57005b6100136100155565b005b610025610020610065565b61009d565b565b606061004c8383
60405180606001604052806027815260200161025c602791396100c1565b939250505050565b6001600160a01b03163b15
1590565b90565b60006100987f360894a13ba1a3210667c828492db98dca3e2076cc3735a920a3ca505d382bbc546001
600160a01b031690565b905090565b3660008037600080366000845af43d6000803e8080156100bc573d6000f35b3d60
00fd5b60606001600160a01b0384163b61012e5760405162461bcd60e51b815260206004820152602660248201527f41
6464726573733a2064656c65676174652063616c6c20746f206e6f6e2d636f6044820152651b9d1c9858dd60d21b6064
8201526084015b60405180910390fd5b600080856001600160a01b03168560405161014991906101dc565b6000604051
80830381855af49150503d8060008114610184576040519150601f19603f3d011682016040523d82523d600060208401
3e610189565b606091505b50915091506101998282866101a3565b9695505050505050565b606083156101b257508161
004c565b8251156101c25782518084602001fd5b8160405162461bcd60e51b815260040161012591906101f8565b6000
```

Unreadable + Unchangeable + Money-related

-> Users don’t trust!

-> Prosperity Issue of Ethereum ☹️

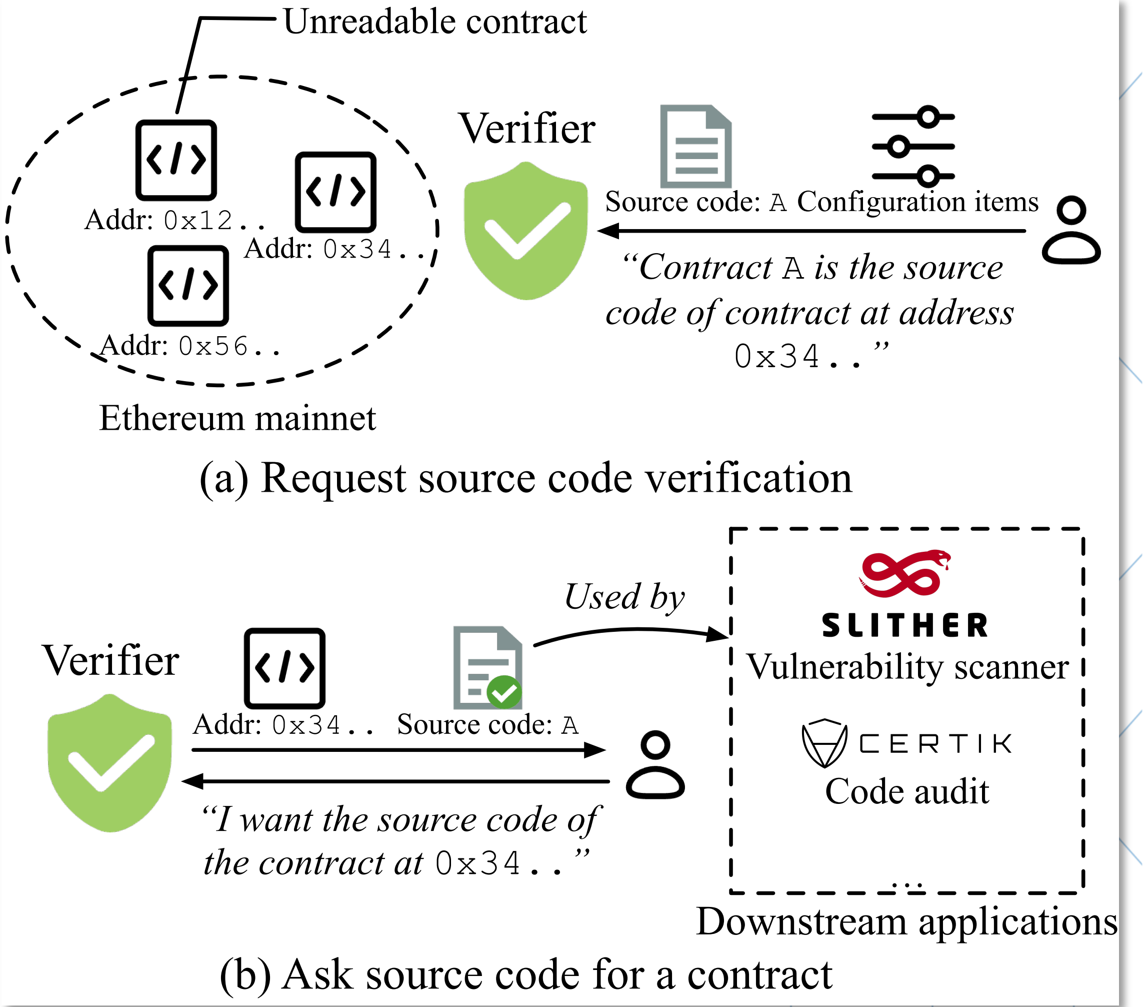
Solution: Source Code Verification Service!

Core idea:

Source code + Compiling options =

On-chain bytecode?

- Two steps:
  - **Request**: Anyone can claim he/she has the source code of any unverified on-chain contract;
  - **Ask**: Anyone can ask for the source code of any address if the verification is passed.



(a) Request source code verification

(b) Ask source code for a contract

Example of source code verification service.

Threat Model

Due to the anonymity of blockchains, source code verifiers allow **anyone** requesting the verification of **any** unverified contract.

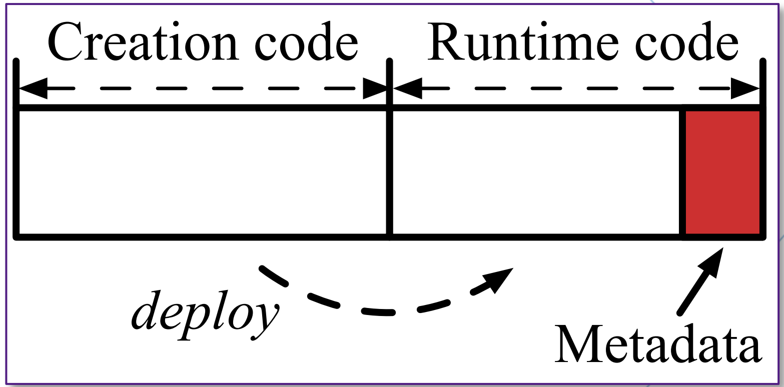
What if the source code verifier is exploited …

Source code provider Actual deployer	**Normal***	**Malicious**
Normal	-	**Discredit (e.g., add fraud or phishing info)**
Malicious	-	**Cover malicious intent (e.g., hide the backdoor)**

* Assume normal users will not exploit source code verifiers.

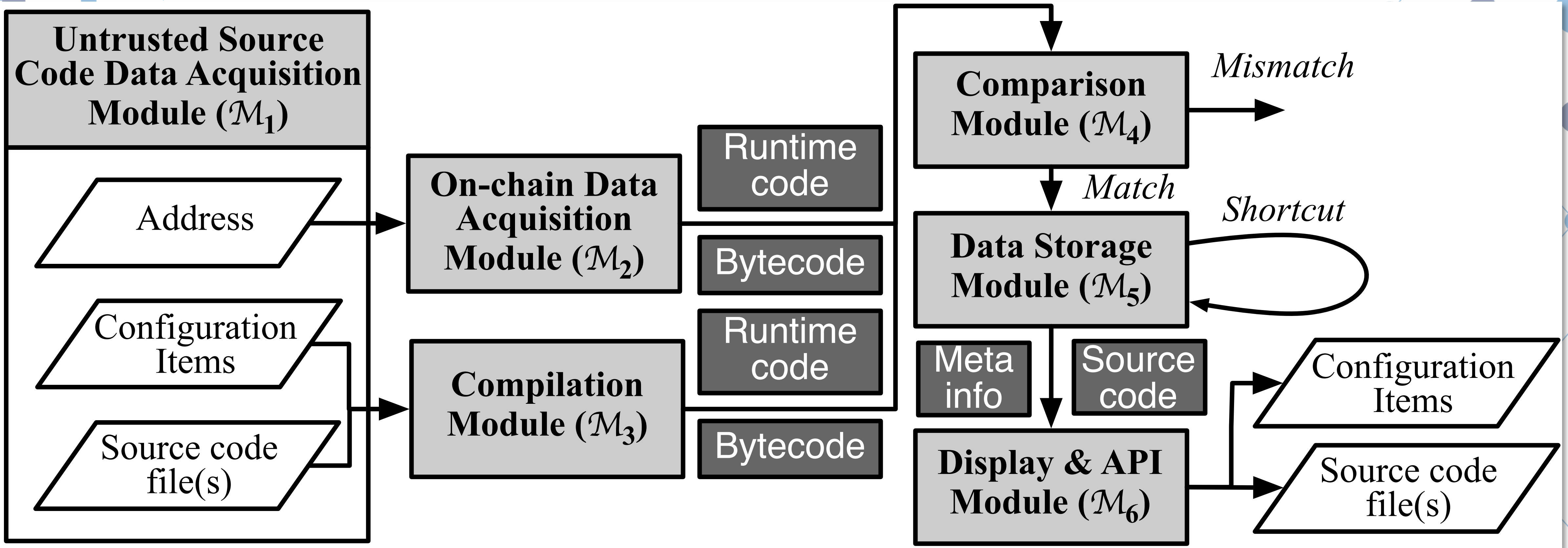
Background Knowledge

- Smart contract **bytecode** can be divided into: **creation code**, **runtime code**, and **metadata**.
  - Creation code: deploy and initialize the runtime code;
  - Runtime code: runtime logic;
  - Metadata: index this contract.
- Three mainstream source code verifiers: **Etherscan**, **Sourcify**, and **Blockscout.**



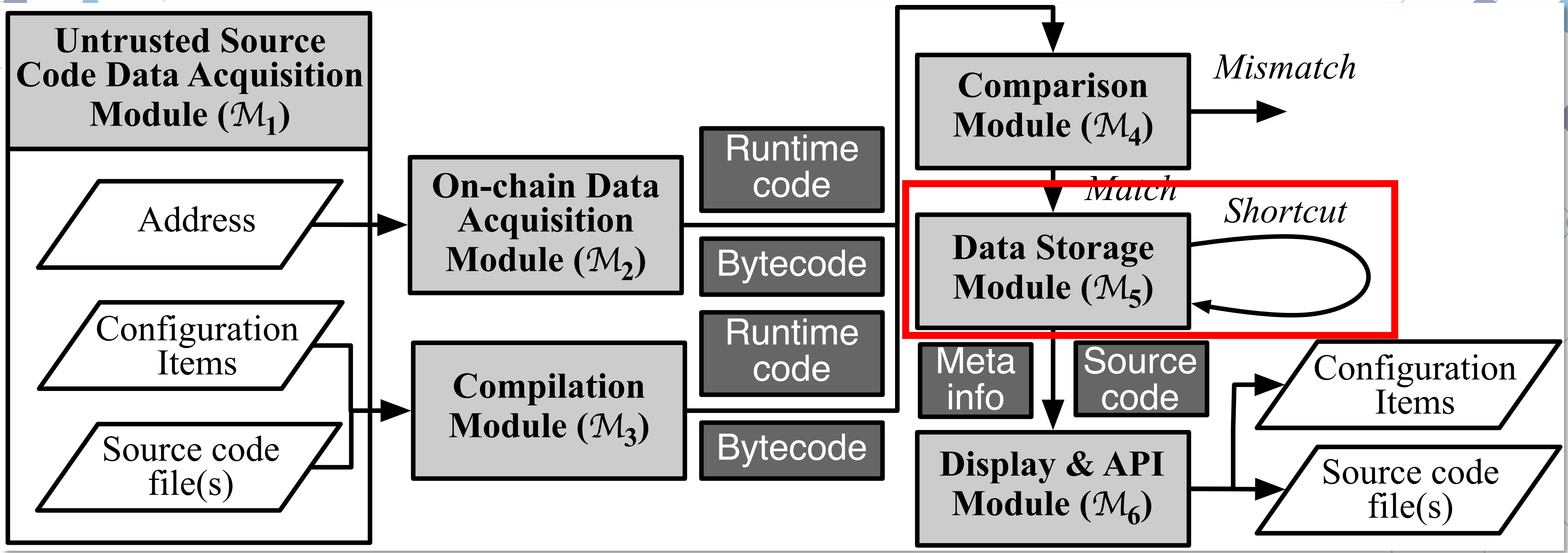
Structure of Ethereum smart contract.

Structure of Source Code Verifier

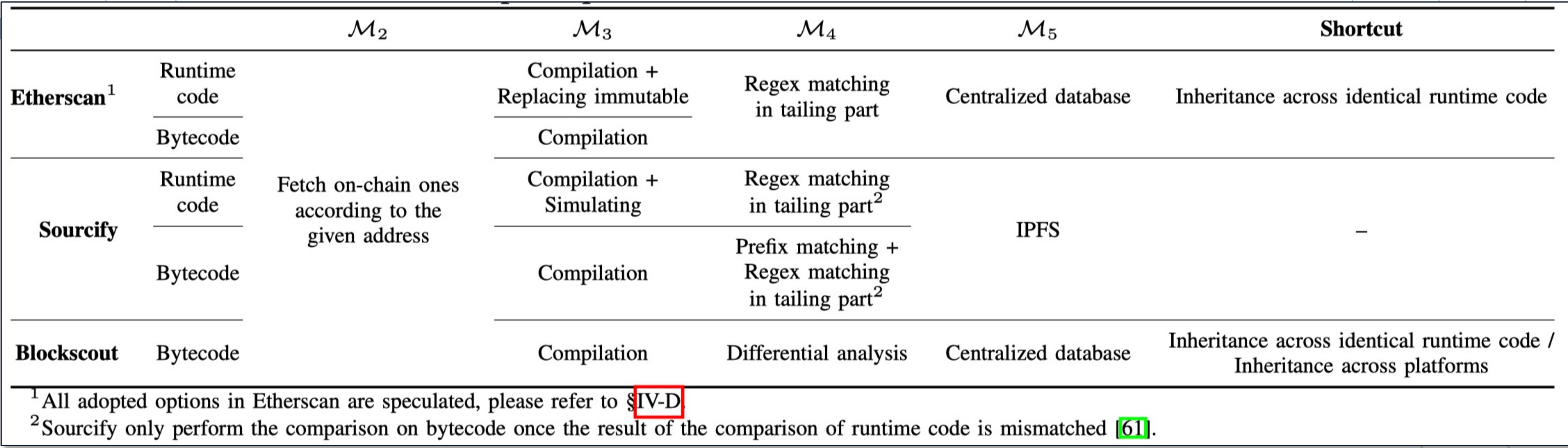


Untrusted Source Code Data Acquisition Module ($\mathcal{M}_1$)
Address
Configuration Items
Source code file(s)
On-chain Data Acquisition Module ($\mathcal{M}_2$)
Compilation Module ($\mathcal{M}_3$)
Runtime code
Bytecode
Runtime code
Bytecode
Comparison Module ($\mathcal{M}_4$)
Mismatch
Match
Shortcut
Data Storage Module ($\mathcal{M}_5$)
Meta info
Source code
Display & API Module ($\mathcal{M}_6$)
Configuration Items
Source code file(s)

Structure of Source Code Verifier


Untrusted Source Code Data Acquisition Module ($\mathcal{M}_1$)
Address
Configuration Items
Source code file(s)
On-chain Data Acquisition Module ($\mathcal{M}_2$)
Compilation Module ($\mathcal{M}_3$)
Runtime code
Bytecode
Runtime code
Bytecode
Comparison Module ($\mathcal{M}_4$)
Mismatch
Match
Data Storage Module ($\mathcal{M}_5$)
Shortcut
Meta info
Source code
Display & API Module ($\mathcal{M}_6$)
Configuration Items
Source code file(s)

Adopted Strategies in Different Modules

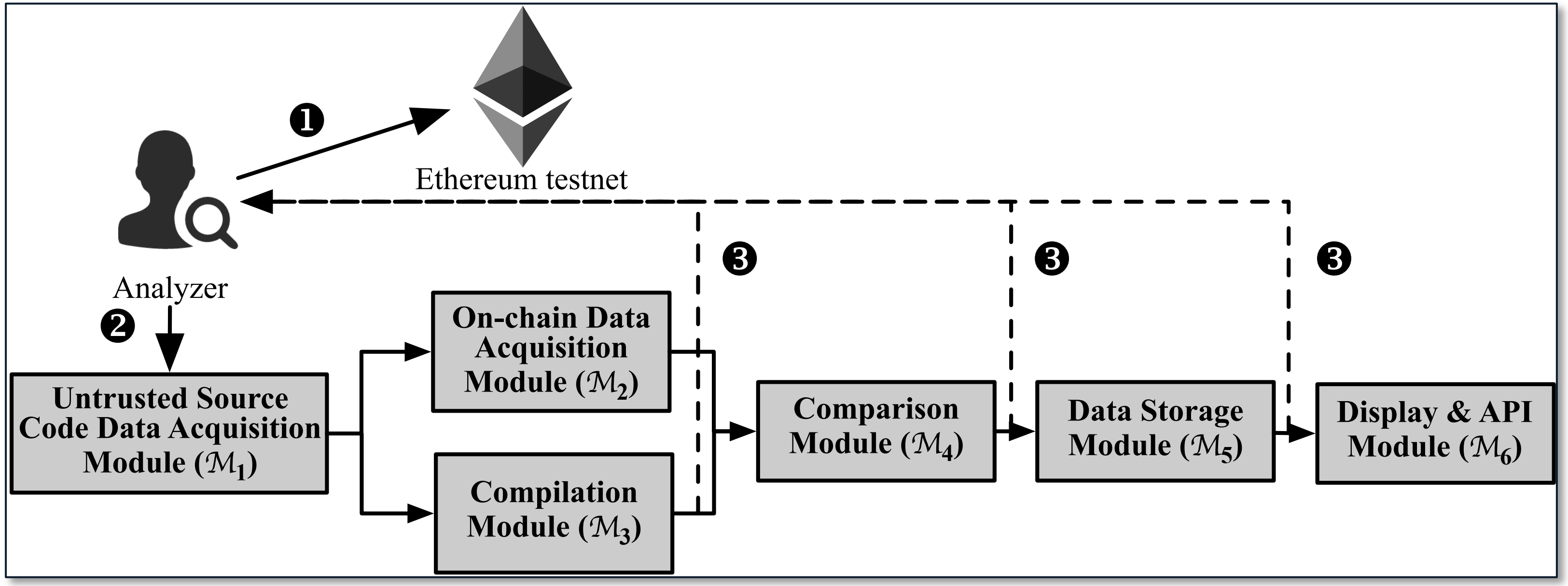


		$\mathcal{M}_2$	$\mathcal{M}_3$	$\mathcal{M}_4$	$\mathcal{M}_5$	**Shortcut**
Etherscan[1]	Runtime code	Fetch on-chain ones according to the given address	Compilation + Replacing immutable	Regex matching in tailing part	Centralized database	Inheritance across identical runtime code
	Bytecode		Compilation			
Sourcify	Runtime code		Compilation + Simulating	Regex matching in tailing part[2]	IPFS	–
	Bytecode		Compilation	Prefix matching + Regex matching in tailing part[2]		
Blockscout	Bytecode		Compilation	Differential analysis	Centralized database	Inheritance across identical runtime code / Inheritance across platforms

[1] All adopted options in Etherscan are speculated, please refer to §IV-D.

[2] Sourcify only perform the comparison on bytecode once the result of the comparison of runtime code is mismatched [61].

How to identify vulnerabilities?



❶
Ethereum testnet
Analyzer
❷
Untrusted Source Code Data Acquisition Module ($\mathcal{M}_1$)
On-chain Data Acquisition Module ($\mathcal{M}_2$)
Compilation Module ($\mathcal{M}_3$)
❸
Comparison Module ($\mathcal{M}_4$)
❸
Data Storage Module ($\mathcal{M}_5$)
❸
Display & API Module ($\mathcal{M}_6$)

For Sourcify and Blockscout, which are open-sourced:

Step 0: Performing code audit according to principles of **unrestorability** and **consistency;**

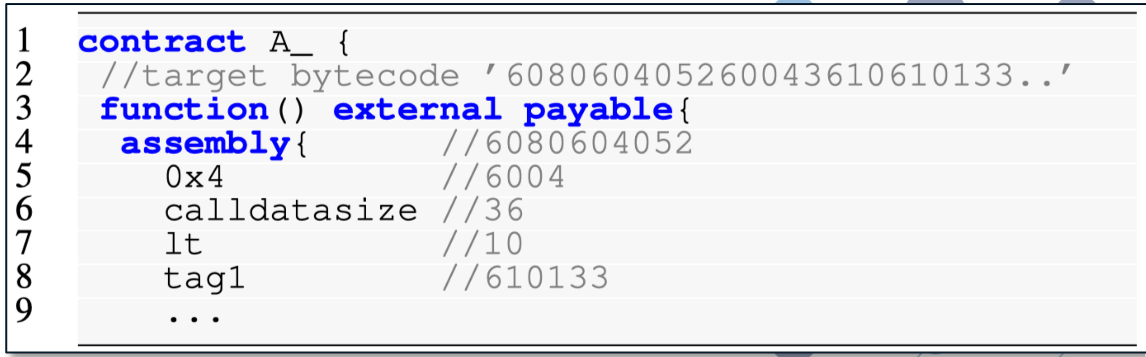
Step 1: Deploying contracts on testnet;

Step 2: Constructing source code and requesting source code verification service;

Step 3: Investigating the outputs of each module to see if they are expected.

Example 1: Exploitable Compiler Features

- Ethereum smart contracts allow **inline assembly**, which can be utilized to embed opcode sequence into the source code;
- **Detection**: Compose only a fallback function, in which it only has a piece of inline assembly. Then, observe if the compilation result is the opcode sequence.

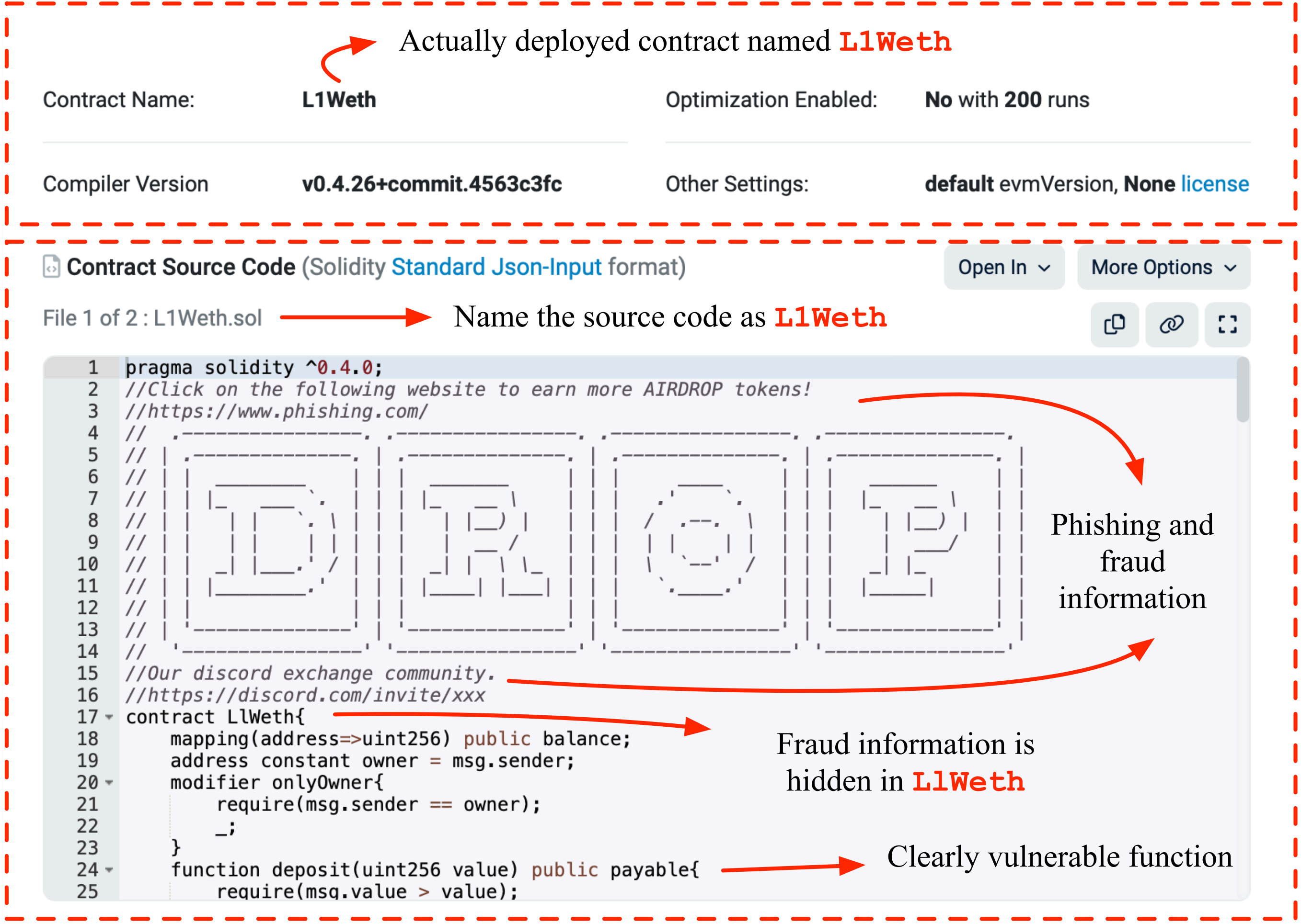


```
contract A_ {
 //target bytecode ’60806040526004361061013..’
 function() external payable{
  assembly{        //6080604052
    0x4            //6004
    calldatasize //36
    lt             //10
    tag1           //610133
    ...
```

Embed victim’s opcode into inline assembly directly.

Example 1: Exploitable Compiler Features

- **PoC:**
  - Construct a contract (`foo`), and put some malicious info in the contract, like fraud information;
  - Construct another contract (`bar`) with victim's opcode by inline assembly;
  - Put `bar` behind `foo`, but take `bar` as the main contract when requesting source code verification.



Actually deployed contract named **L1Weth**

Contract Name:	**L1Weth**	Optimization Enabled:	**No** with **200** runs
Compiler Version	**v0.4.26+commit.4563c3fc**	Other Settings:	**default** evmVersion, **None** license

Contract Source Code (Solidity Standard Json-Input format)

File 1 of 2 : L1Weth.sol

Name the source code as **L1Weth**

```
pragma solidity ^0.4.0;
//Click on the following website to earn more AIRDROP tokens!
//https://www.phishing.com/
//  .----------------.  .----------------.  .----------------.  .----------------.
// | .--------------. || .--------------. || .--------------. || .--------------. |
// | |  ________    | || |  _______     | || |     ____     | || |   ______     | |
// | | |_   ___ `.  | || | |_   __ \    | || |   .'    `.   | || |  |_   __ \   | |
// | |   | |   `. \ | || |   | |__) |   | || |  /  .--.  \  | || |    | |__) |  | |
// | |   | |    | | | || |   |  __ /    | || |  | |    | |  | || |    |  ___/   | |
// | |  _| |___.' / | || |  _| |  \ \_  | || |  \  `--'  /  | || |   _| |_      | |
// | | |________.'  | || | |____| |___| | || |   `.____.'   | || |  |_____|     | |
// | |              | || |              | || |              | || |              | |
// | '--------------' || '--------------' || '--------------' || '--------------' |
//  '----------------'  '----------------'  '----------------'  '----------------'
//Our discord exchange community.
//https://discord.com/invite/xxx
contract LlWeth{
    mapping(address=>uint256) public balance;
    address constant owner = msg.sender;
    modifier onlyOwner{
        require(msg.sender == owner);
        _;
    }
    function deposit(uint256 value) public payable{
        require(msg.value > value);
```

Phishing and fraud information

Fraud information is hidden in **LlWeth**

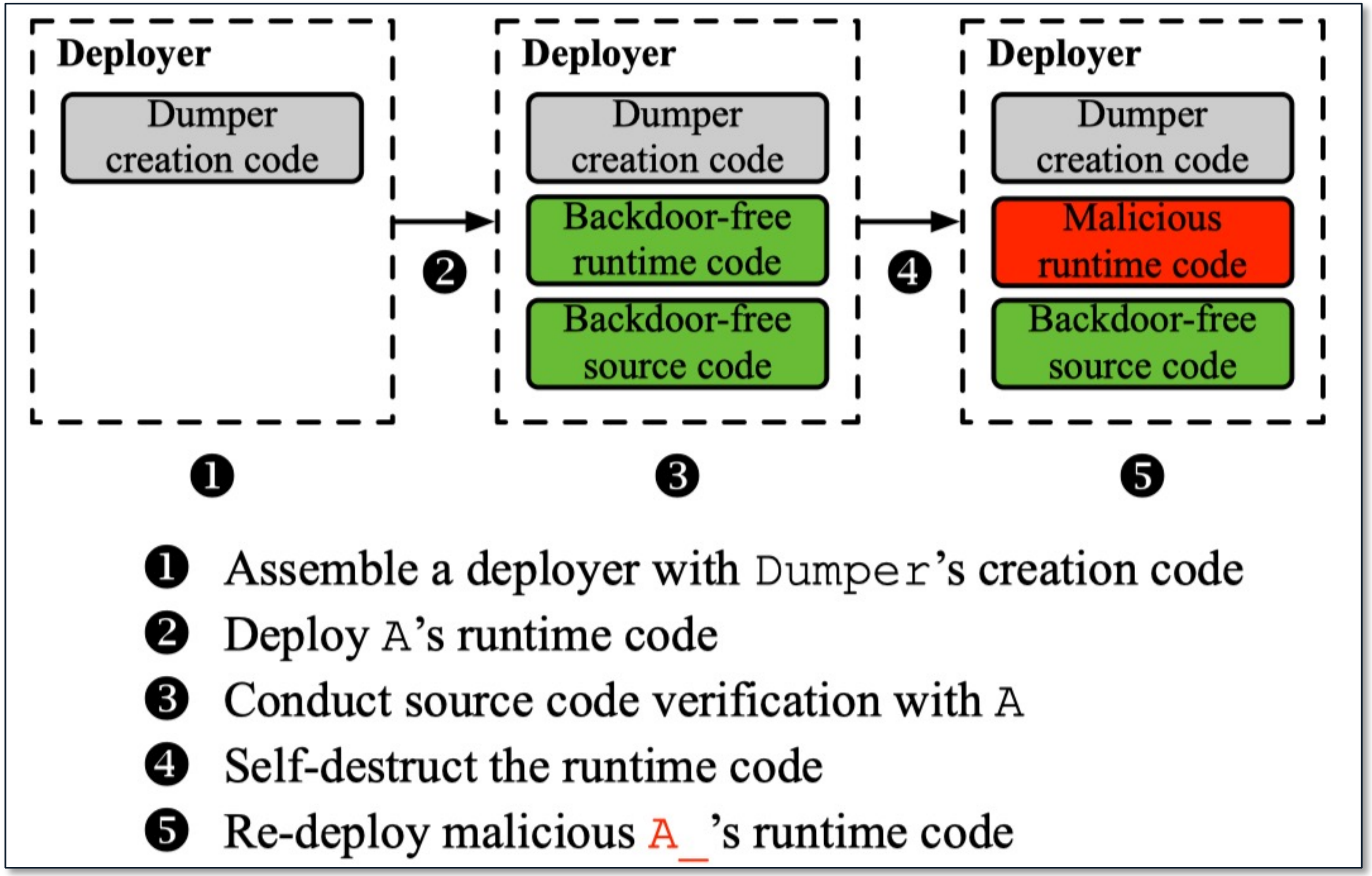
Clearly vulnerable function

PoC example

Example 2: Replaceable On-chain Contracts

- This type of vulnerability has caused 750K USD financial loss for Tornado.cash;
- Because Ethereum contracts are unchangeable, verifiers have not taken source code update into the consideration;
- Malicious users can abuse `create2` to update on-chain contracts. An obvious feature of `create2` is: if the creation code is not modified, the address of the deployed contract will not be modified either.

Example 2: Replaceable On-chain Contracts



❶ Assemble a deployer with `Dumper`'s creation code

❷ Deploy `A`'s runtime code

❸ Conduct source code verification with `A`

❹ Self-destruct the runtime code

❺ Re-deploy malicious `A_`'s runtime code

PoC example

Overall Results

Against three mainstream verification services, we have conducted a comprehensive detection.

X: exploitable, *: confirmed, and red one: patched.

Consequence	Vulnerability	Etherscan	Sourcify	Blockscout
Discredit	Exploitable Compiler Features	X	X*	X*
	Unchecked Simulating	-	X*	X*
	Incomplete Bytecode Validation	-	X*	X*
Cover malicious intent	Replaceable On-chain Contracts	X	X*	X*
	Unverified Linked Libraries	X	X*	X*
	Mislabelled Bytecode	-	X*	X*
	Path Traversal Risk	-	X*	X*
	Inadequate Information Disclosure	X	-	X*

Overall Results

Consequence	Vulnerability	Etherscan	Sourcify	Blockscout
Discredit	Exploitable Compiler Features	X	X*	X*
	Unchecked Simulating	-	X*	X*
	Incomplete Bytecode Validation	-	X*	X*
Cover malicious intent	Replaceable On-chain Contracts	X	X*	X*
	Unverified Linked Libraries	X	X*	X*
	Mislabelled Bytecode	-	X*	X*
	Path Traversal Risk	-	X*	X*
	Inadequate Information Disclosure	X	-	X*

Etherscan is the least affected. This is partly due to the black-box testing method.

Overall Results

Consequence	Vulnerability	Etherscan	Sourcify	Blockscout
Discredit	Exploitable Compiler Features	X	X*	X*
	Unchecked Simulating	-	X*	X*
	Incomplete Bytecode Validation	-	X*	X*
Cover malicious intent	Replaceable On-chain Contracts	X	X*	X*
	Unverified Linked Libraries	X	X*	X*
	Mislabelled Bytecode	-	X*	X*
	Path Traversal Risk	-	X*	X*
	Inadequate Information Disclosure	X	-	X*

Sourcify adopts some user-friendly strategies, which reduces the amount of information the requesters need to provide. However, these strategies need additional operations on the source code, which could be abused by attackers.

Overall Results

Consequence	Vulnerability	Etherscan	Sourcify	Blockscout
Discredit	Exploitable Compiler Features	X	X*	X*
	Unchecked Simulating	-	X*	X*
	Incomplete Bytecode Validation	-	X*	X*
Cover malicious intent	Replaceable On-chain Contracts	X	X*	X*
	Unverified Linked Libraries	X	X*	X*
	Mislabelled Bytecode	-	X*	X*
	Path Traversal Risk	-	X*	X*
	Inadequate Information Disclosure	X	-	X*

One of the critical reason of so many exploitable vulnerabilities in Blockscout is its adopted shortcut, i.e., Blockscout directly recognizes the results of Sourcify.

Overall Results

Consequence	Vulnerability	Etherscan	Sourcify	Blockscout
Discredit	Exploitable Compiler Features	X	X*	X*
	Unchecked Simulating	-	X*	X*
	Incomplete Bytecode Validation	-	X*	X*
Cover malicious intent	Replaceable On-chain Contracts	X	X*	X*
	Unverified Linked Libraries	X	X*	X*
	Mislabelled Bytecode	-	X*	X*
	Path Traversal Risk	-	X*	X*
	Inadequate Information Disclosure	X	-	X*

- **For ECF**: Lots of normal contracts adopt these could-be-abused features to achieve functionalities;
- **For ROC**: Verifiers believe that users should be directly responsible for their actions, so they only add prominent warning messages.

Impact Scope

Consequence	Vulnerability	\# Impacted Contracts
Discredit	Exploitable Compiler Features	49K
	Unchecked Simulating	~ 58.9M
	Incomplete Bytecode Validation	~ 58.9M

- For the discredit consequence, the number of potential victims is the one of all unverified contracts. Because verified ones cannot be verified again in most cases.
- For the first vulnerability, a successful exploitation requires some prerequisites, which lower the number.

Impact Scope

Consequence	Vulnerability	\# Impacted Contracts
Cover malicious intent	Replaceable On-chain Contracts	2
	Unverified Linked Libraries	244
	Mislabelled Bytecode	0
	Path Traversal Risk	0
	Inadequate Information Disclosure	0

- For this consequence, the number corresponds to the ones that actually conduct behaviors to cover their malicious intents.
- By exploiting the first vulnerability, the attacker was able to replace the source code of a **malicious proposal** with a seemingly **harmless one**, ultimately causing more than 750,000 USD financial losses for Tornado.Cash.

Takeaways

- To the best of our knowledge, it is **the first work** that systematically illustrates the design and implementation of Ethereum source code verification services;
- **Eight types of vulnerabilities** are uncovered, which could be abused to discredit normal contracts or cover malicious intents;
- Among three mainstream verifiers, we found **19 exploitable vulnerabilities**, 15 of them have been confirmed and 10 of them have be patched;
- **Tens of millions** of contracts can be discredited potentially, and malicious behaviors in **hundreds of contracts** may have been covered already;
- **Public dataset**: https://github.com/source-code-scam-paper/source-scam-all-in-one

Q&A Time

mpx199924@gmail.com
ningyu.he@pku.edu.cn

NDSS SYMPOSIUM/2024

Presented by Internet Society

#NDSSSymposium2024