192.127 Seminar in Software Engineering (Smart Contracts) SWC-124: Write to Arbitrary Storage Location Report Exploits

Ivanov, Ivaylo (11777707) & Millauer, Peter (01350868)

WT 2023/24

1 Introduction

1.1 Precautions

1.2 Preprocessing

1.3 Setup

We attempted to implement an automatic weakness detection pipeline by using a multitude of tools. The software used includes:

- **solc**: Using **solc-select** we select the correct solc version and compile the associated *.sol file. This way we gather compiler hints and warnings.
- **Mythril**:
- **teEther**: A tool we found in the last research step of this seminar, teether is a dynamic analysis tool for smart contracts. It is unfortunately hardly documented and has been unmaintained for several years. We were unable to generate usable results with it.
- **Slither**: A highly useful tool that offers a large static analysis toolkit for solidity, it not only allows the extraction of contract data like storage layouts but also automatic scanning for common weaknesses. Although it did not seem to be able to detect SWC-124, the storage layout functionality was used extensively by our team.

2 Exploit Creation

2.1 extra short recap of weakness definitions

SWC-124 attempts to use the underlying mechanisms that govern dynamic array allocation to deterministically overwrite arbitrary data in smart contracts. An analogous weakness can be created by employing unchecked assembly instructions, although this is a less common attack vector due to its unusual structure.

2.2 exploit idea(s)

The workflow of determining the vulnerability of a given contract is straight forward and follows the general approach of automatic detection mechanisms from our last paper. Since SWC-124 requires the presence of very specific code, it is relatively easy to exclude non-vulnerable contracts. Namely any contract without dynamic arrays or raw assembly including a SSTORE instruction can immediately be considered non-vulnerable. The presence of dynamic arrays can be determined using **slither –print variable-order**. A sample output looks as follows:

```
$slither cans.sol --print variable-order

Cans:
+----------------------------+----------------------------------------------+------+--------+
| Name                       |                                         Type | Slot | Offset |
+----------------------------+----------------------------------------------+------+--------+
| ERC1155._balances          |              mapping(address => uint32[7]) |    0 |      0 |
| ERC1155.tokens             |                                    uint256[] |    1 |      0 |
| ERC1155._operatorApprovals | mapping(address => mapping(address => bool)) |    2 |      0 |
| ERC1155._uri               |                                       string |    3 |      0 |
| Ownable._owner             |                                      address |    4 |      0 |
| Functional._reentryKey     |                                         bool |    4 |     20 |
| Cans.START_TIME            |                                      uint256 |    5 |      0 |
| Cans.END_TIME              |                                      uint256 |    6 |      0 |
| Cans.amountClaimed         |                                 uint8[9998] |    7 |      0 |
| Cans.CLAIM_ENABLED         |                                         bool |  320 |      0 |
| Cans.soda                  |                                 SODAContract |  320 |      1 |
| Cans.SODA_CONTRACT         |                                      address |  321 |      0 |
| Cans.baseURI               |                                       string |  322 |      0 |
+----------------------------+----------------------------------------------+------+--------+
```

In this example the **ERC1155.tokens** array is the only potential weakess present. We would then look for the presence of what-where writes to this array in order to confirm the potential presence of SWC-124. What-where writes are of the form

```
someArray[where] = what;
```

and are a necessary code snippet for SWC-124. If such an instruction is present, we then attempt to reverse engineer a sequence of inputs to trigger the exploit, or formulate a reason why we believe this not to be possible.

3 Results

3.1 vulnerable contracts

None.

3.2 non-exploitable contracts

Solidity files that contained no contracts:

- AuctionLib.sol
- LibRegion.sol
- LToken.sol

Contracts that were discarded due to the non-presence of dynamic arrays or assembly using SSTORE:

- DCU.sol

- ERC20_Asset_Pool.sol
- FacelessNFT.sol
- GElasticTokenManager.sol
- GoldToken.sol
- GovernmentAlpha.sol
- HedgeSwap.sol
- HermesImplementation.sol
- IMETACoin223Token_13.sol
- UniswapV3PoolAdapter.sol
- UserDeposit.sol
- WPCMainnetBridge.sol

3.3 undecidable contracts

3.4 optionally fixes

4 Discussion

4.1 conclusions

4.2 lessons learned: what works, what doesn't

4.3 open challenges

References

[1] Jaeseung Choi, Doyeon Kim, Soomin Kim, Gustavo Grieco, Alex Groce, and Sang Kil Cha. Smartian: Enhancing smart contract fuzzing with static and dynamic data-flow analyses. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 227–239, 2021.

[2] doughoyte. Merdetoken: It's some hot shit. `https://github.com/Arachnid/uscc/tree/master/submissions-2017/doughoyte` [Accessed: Oct. 27th 2023].

[3] Li Duan, Yangyang Sun, Ke-Jia Zhang, and Yong Ding. Multiple-layer security threats on the ethereum blockchain and their countermeasures. *Security and Communication Networks*, 2022, 02 2022.

[4] Sukrit Kalra, Seep Goel, Mohan Dhawan, and Subodh Sharma. Zeus: Analyzing safety of smart contracts. In *Network and Distributed System Security Symposium*, 2018.

[5] Johannes Krupp and Christian Rossow. teEther: Gnawing at ethereum to automatically exploit smart contracts. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1317–1333, Baltimore, MD, August 2018. USENIX Association.

[6] Siddhasagar Pani, Harshita Vani Nallagonda, Vigneswaran, Raveendra Kumar Medicherla, and Rajan M. Smartfuzzdrivergen: Smart contract fuzzing automation for golang. In *Proceedings of the 16th Innovations in Software Engineering Conference*, ISEC '23, New York, NY, USA, 2023. Association for Computing Machinery.

[7] Petar Tsankov, Andrei Dan, Dana Drachsler-Cohen, Arthur Gervais, Florian Bünzli, and Martin Vechev. Securify: Practical security analysis of smart contracts. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 67–82, New York, NY, USA, 2018. Association for Computing Machinery.