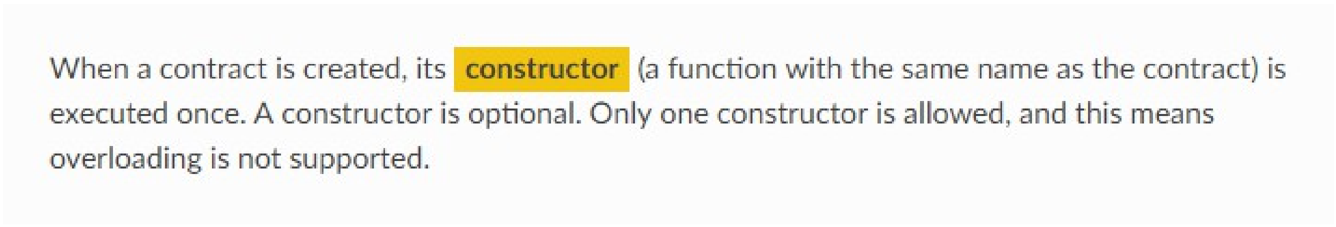
构造函数缺失漏洞分析

构造函数缺失漏洞(Missing Constructor)此漏洞非常常见，可以说自智能合约产生以来就一直出现，由于新进开发者对 solidity 代码结构不熟悉造成的。本篇文章就来介绍一下漏洞的基本原理、表现形式以及对开发者的建议。

漏洞原理

1 什么是构造函数

Solidity编写合约和面向对象编程语言非常相似，我们可以通过构造函数（constructor）来初始化合约对象。构造函数就是方法名和合约名字相同的函数，创建合约时会调用构造函数对状态变量进行数据初始化操作。


When a contract is created, its constructor (a function with the same name as the contract) is executed once. A constructor is optional. Only one constructor is allowed, and this means overloading is not supported.

构造函数可用的函数类型为 public 或 internal，如果有 payable 修饰，就只能是 public 类型。而大部分人的写法都是 public 或者不写。不写类型则由函数可见性默认为 public 类型。同时，如果构造函数带参数，则一定要放在合约下的第一个函数。

```
pragma solidity 0.4.21;

contract Foo {
  function Foo() public {
    // ...
  }
}
```

合约结构的规范化写作可以让其他人更好地阅读，并且 constructor 和 fallback 函数要放在前面以便更好地查看。官方建议以如下顺序写作：

- constructor
- fallback function (if exists)
- external
- public
- internal
- private

合约结构的规范写法如下：

```
contract A {
    constructor() public {
        ...
    }

    function() external {
        ...
    }

    // External functions
    // ...

    // External functions that are view
    // ...

    // External functions that are pure
    // ...

    // Public functions
    // ...

    // Internal functions
    // ...

    // Private functions
    // ...
}
```

2 构造函数缺失与危害

构造函数缺失则是由于构造函数与合约名字不同，而又为 public 类型，就变成了一个公有函数了，可以被任何人调用，一般构造函数比较敏感，用于初始化合约时定义通证数量、管理员地址等基本变量状态，一旦变成了公有函数，危害可想而知，权限控制、通证管理基本全线奔溃。

3 版本升级后构造函数的变化

从 `0.4.22` 版本开始，solidity 编译器引入了 constructors 关键字，以替代低版本的将合约名作为构造函数名的语法，避免程序员容易出现的编码错误。使用旧写法会出现 warning 信息。

新版本写法为：

```
contract Ownable {
    address public owner;
    address public admin;

    constructor() public {
        owner = msg.sender;
        admin = msg.sender;
    }

    // Other code
}
```

常见的漏洞代码形式

1 构造函数使用库合约名称，使得构造函数变成了公有函数，可被任意调用。

我们发现了不少这种错误写法的合约，在主合约中，构造函数写成了 Token() 或 ERC20Token() 。这样子的函数非构造函数，变成了普通函数了。

如 DealGuard 合约：https://etherscan.io//address/0xb47c5e8f389dc1fa8247c9a4b9e5dcdc79754152#code

```
contract DealGuard {
    // Public variables
    string public name;
    string public symbol;
    uint256 public totalSupply;
    address public owner;
    uint8 public decimals = 12; // 18 decimals

    // This creates an array with all balances
    mapping (address => uint256) public balanceOf;
    mapping (address => mapping (address => uint256)) public allowance;

    // This generates a public event on the blockchain that will notify clients
    event Transfer(address indexed from, address indexed to, uint256 value);

    // This notifies clients about the amount burnt
    event Burn(address indexed from, uint256 value);

    // This notifies clients about the emission amount
    event tokenEmission(address indexed from, uint256 value);

    /**
     * Constrctor function
     *
     * Initializes contract with initial supply tokens to the creator of the contract
     */
    function Token() public {
        totalSupply = 10000000000 * 10 ** uint256(decimals);       // Update total supply
with the decimal amount
        name = 'Deal Guard Token';                                                      //
Set the name for display purposes
        symbol = 'DG';                                                     // Set the
symbol for display purposes
        balanceOf[msg.sender] = totalSupply;                                        // Give
the creator all initial tokens
        owner = msg.sender;                                                         //
Contract owner
    }
```

2 合约名与构造函数名不同，如大小写不同、拼写错误等，使得构造函数变成了公有函数，可被任意人调用。

只能说开发者太大意，对智能合约开发有什么误解吧。经常看到的漏洞形式是合约名与构造函数不同，只与 name 变量名相同，估计是开发者以为合约名相同就行了，却忽略了构造函数。

如 ReaperCoin11 合约：https://etherscan.io//address/0x1b7cd071187ec0b2995b96ee82296cfa639572f1#code

```
contract ReaperCoin11{
    /* Public variables of the token */
    string public standard = 'Token 0.1';
    string public name;
    string public symbol;
    uint8 public decimals;
    uint256 public initialSupply;
    /* This creates an array with all balances */
    mapping (address => uint256) public balanceOf;
    mapping (address => mapping (address => uint256)) public allowance;
    /* Initializes contract with initial supply tokens to the creator of the contract */
    function Reaper11() {
         initialSupply = 2800000;
         name ="ReaperCoin11";
        decimals = 2;
         symbol = "RPCT";
        balanceOf[msg.sender] = initialSupply;              // Give the creator all initial
tokens
        uint256 totalSupply = initialSupply;                        // Update total supply
    }
```

还有这个 Krypticoin 合约，乍一看看不出来，仔细瞅瞅，再仔细瞅瞅，原来 coin 拼错了呀。

```
contract Krypticoin {
    /* Public variables of the token */
    string public standard = 'Token 0.1';
    string public name;
    string public symbol;
    uint8 public decimals;
    uint256 public initialSupply;

    /* This creates an array with all balances */
    mapping (address => uint256) public balanceOf;
    mapping (address => mapping (address => uint256)) public allowance;

    /* Initializes contract with initial supply tokens to the creator of the contract */
    function Krypticion() {

         initialSupply = 500000;
         name ="krypticoin";
        decimals = 0;
         symbol = "P";

        balanceOf[msg.sender] = initialSupply;              // Give the creator all initial
tokens
        uint256 totalSupply = initialSupply;                        // Update total supply

    }
```

之前知道创宇发布了关于 `owned` 大小写编码漏洞的文章。同时，我们也可以找一些类似的容易大小写错误的库合约，如 `ownable` 。不过经过我们的分析，还是以 owned 大小写错误为主。

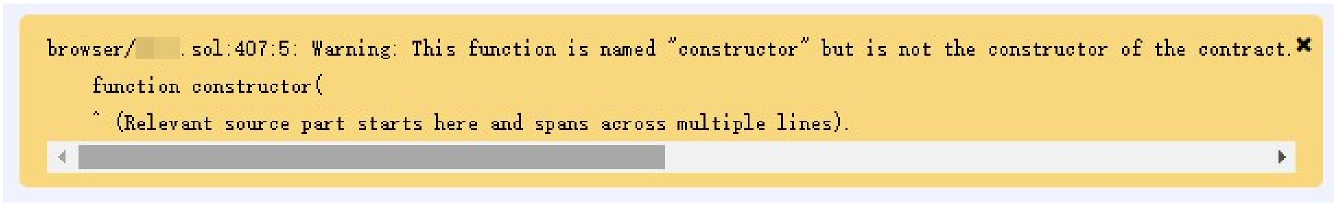
如 MORPH 合约：https://etherscan.io//address/0x2ef27bf41236bd859a95209e17a43fbd26851f92#code

```
contract Owned {
    address public owner;

    function owned() public {
        owner = msg.sender;
    }
```

3 constructor 前加了 function ，或者加了 fucntion 然后开头的 C 写成了大写，即 `function Constructor() public {}`，使得构造函数变成了公有函数，可被任意人调用。

版本升级后的构造函数只需要单独使用 constructor 即可，但很多开发者却忽略了此细节。上面这个写法错误在于两点：

1) 加入了 function，使得构造函数变成了普通函数。这时编辑器中会报 warning 信息。



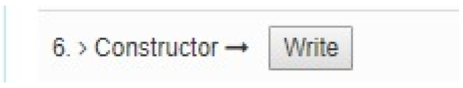
```
browser/[redacted].sol:407:5: Warning: This function is named "constructor" but is not the constructor of the contract.
    function constructor(
    ^ (Relevant source part starts here and spans across multiple lines).
```

但是有开发者忽了此警告，写出了错误代码。

如 MDOT 合约：https://etherscan.io//address/0xef7d906fd1c0eb5234df32f40c6a1cb0328d7279#code

```
    function constructor() public {
        totalSupply = 5000000000000;       // Set the total supply (in base units)
        balances[0xbfC729007CE9CBBE54132Fb9BFa097D80AAC791C] = 5000000000000;    //
Initially assign the entire supply to the specified account
    }
```

1) 但是如果将 `constructor` 错写成了 `Constructor` ，使得编辑器识别其为普通函数名，没有任何 warning 信息。



如 TOGToken 合约：https://etherscan.io//address/0xb9d5c2548266428795fd8b1f12aedbdeb417fe54#code

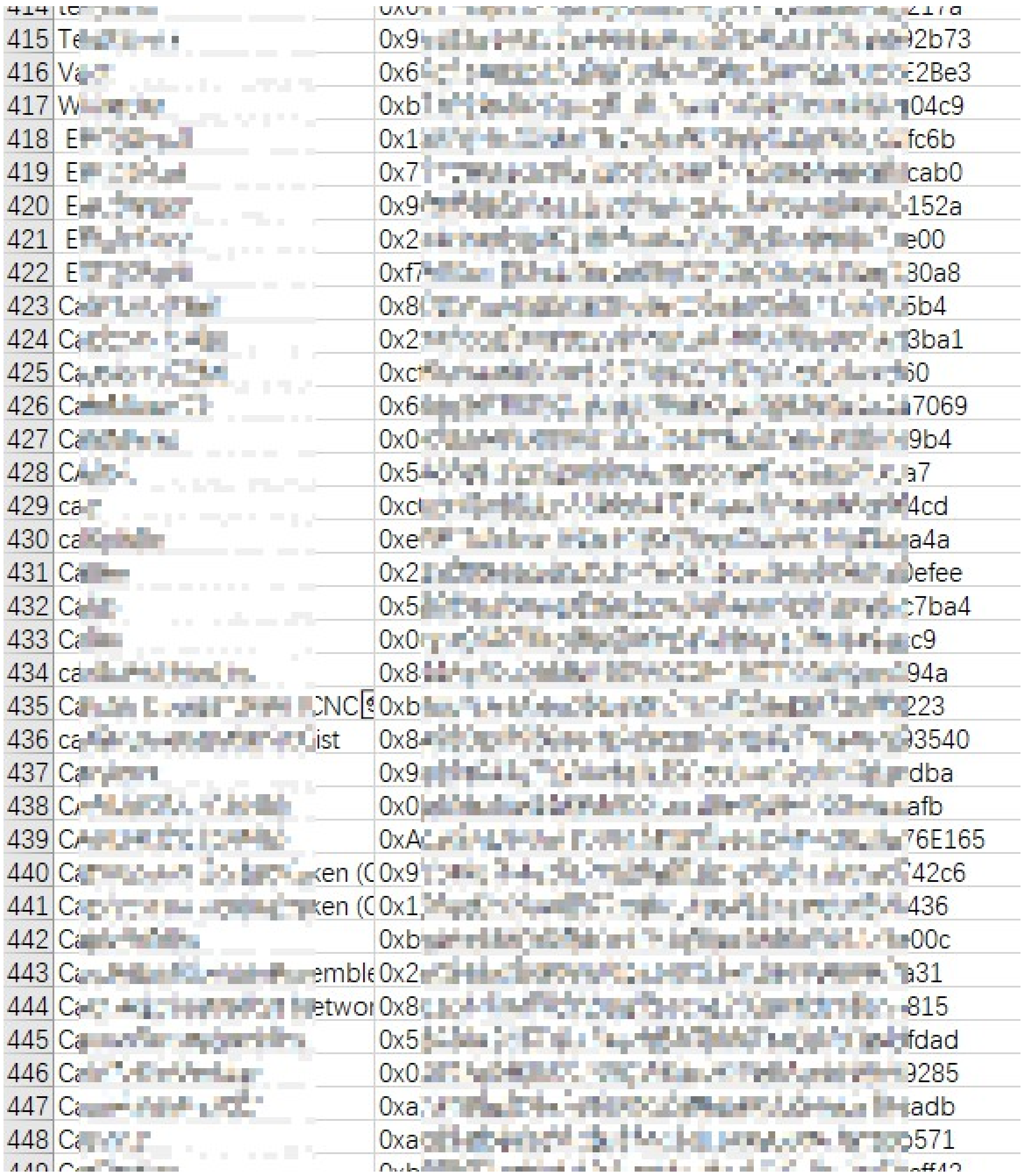
```
contract Owned {
    address public owner;
    address public newOwner;

    event OwnershipTransferred(address indexed _from, address indexed _to);

    function Constructor() public { owner = msg.sender; }

    modifier onlyOwner {
        require(msg.sender == owner);
        _;
    }

    function transferOwnership(address _newOwner) public onlyOwner {
        newOwner = _newOwner;
    }
    function acceptOwnership() public {
        require(msg.sender == newOwner);
        emit OwnershipTransferred(owner, newOwner);
        owner = newOwner;
        newOwner = address(0);
    }
}
```

修复方案

使用新版本写法，并且检查句式、拼写的正确性。

漏洞影响范围

通过我们对以太坊上的合约进行整体监测，发现有此漏洞的竟多达 3000 份，大小写编码错误的约有 20 份。其中不乏正在使用中的合约。



415	Te[redacted]	0x9[redacted]92b73
416	Va[redacted]	0x6[redacted]E2Be3
417	W[redacted]	0xb[redacted]04c9
418	E[redacted]	0x1[redacted]fc6b
419	E[redacted]	0x7[redacted]cab0
420	E[redacted]	0x9[redacted]152a
421	E[redacted]	0x2[redacted]e00
422	E[redacted]	0xf7[redacted]30a8
423	Ca[redacted]	0x8[redacted]5b4
424	Ca[redacted]	0x2[redacted]3ba1
425	Ca[redacted]	0xc[redacted]60
426	Ca[redacted]	0x6[redacted]7069
427	Ca[redacted]	0x0[redacted]9b4
428	C[redacted]	0x5[redacted]a7
429	ca[redacted]	0xc[redacted]4cd
430	ca[redacted]	0xe[redacted]a4a
431	Ca[redacted]	0x2[redacted]efee
432	Ca[redacted]	0x5[redacted]c7ba4
433	Ca[redacted]	0x0[redacted]c9
434	ca[redacted]	0x8[redacted]94a
435	Ca[redacted]CNC	0xb[redacted]223
436	ca[redacted]ist	0x8[redacted]93540
437	Ca[redacted]	0x9[redacted]dba
438	C[redacted]	0x0[redacted]afb
439	C[redacted]	0xA[redacted]76E165
440	Ca[redacted]ken (C	0x9[redacted]42c6
441	Ca[redacted]ken (C	0x1[redacted]436
442	Ca[redacted]	0xb[redacted]00c
443	Ca[redacted]emble	0x2[redacted]a31
444	Ca[redacted]etwor	0x8[redacted]815
445	Ca[redacted]	0x5[redacted]fdad
446	Ca[redacted]	0x0[redacted]9285
447	Ca[redacted]	0xa[redacted]adb
448	Ca[redacted]	0xa[redacted]571

如需要对合约检测、审计可以与我们联系。

资料

https://solidity.readthedocs.io/en/latest/contracts.html?highlight=constructor#creating-contracts

https://solidity.readthedocs.io/en/latest/style-guide.html?highlight=constructor#order-of-functions

https://ethereum.stackexchange.com/questions/30223/should-the-constructor-function-be-public

http://solidity.readthedocs.io/en/develop/contracts.html?highlight=private#visibility-and-getters

https://github.com/trailofbits/not-so-smart-contracts/tree/master/missing_constructor

附录

下面两个合约存在上面提到的第三种漏洞类型，我们发现这两个合约目前仍在大量使用中，于是申请到了 CVE 编号。

EUX Link Token (EUX) 0xf55a32f0107523c14027c4a1e6177cd7291395a0 CVE-2018-14406

UEX Cloud (UEX) 0xd7290307c040f4089f8650b7f7aac3cfe39cd6bd CVE-2018-14407

厂商信息：https://www.crunchbase.com/organization/uex#section-overview

Website www.uexglobal.com/ Contact Email contact@uexglobal.com Phone Number +65 3158 3677

关于玄猫安全

玄猫区块链安全实验室专注区块链安全领域，致力于提供区块链行业最专业的安全解决方案，团队成员来自于百度、阿里、360等国际顶尖安全团队，已为数十家交易所、电子钱包、智能合约等提供基础安全建设、渗透测试、漏洞挖掘、应急响应等安全服务。

玄猫安全实验室提供专业权威的智能合约审计服务、区块链专项应用评估、区块链平台安全评估等多项服务。

商务合作：Lyon.chen@xuanmao.org