

以太坊 Solidity 未初始化存储指针安全风险浅析

By Thinking@慢雾安全团队

0x00 引子

看到安比实验室有篇文章在说《警惕！Solidity缺陷易使合约状态失控》的问题，原文链接可以在参考链接中获取。

这个问题实际上之前在慢雾区中，爱上平顶山(山哥)和 keywolf 就有对一篇外文进行了翻译，可以在 SlowMist 的 GitHub 中找到(地址见参考链接)，这篇译文《Solidity 安全：已知攻击方法和常见防御模式综合列表》里面就有讲到。

其实就是 **Unintialised Storage Pointers (未初始化的存储指针)** 的安全问题，EVM中会将数据存储为 storage 或 memory，在函数中局部变量的默认类型取决于它们本身的类型，未进行初始化的 storage 变量，会指向合约中的其他变量，从而改变其他变量的值，常见的场景就是指向状态变量，改变状态变量的值，导致漏洞的产生。

0x01 分析过程

依据 Solidity 官方手册上的介绍，以及经过实验得到了一些总结分析。

这里要注意结构体，数组和映射的局部变量，在官方手册中有提到这些类型的局部变量默认是放在 storage 中的，因此这些局部变量可能都存在相同的问题。（本文分析了结构体和数组的 Unintialised Storage Pointers 问题，而 mapping 暂未找到存在问题的案例）

Introduction to Smart Contracts

Installing the Solidity Compiler

Solidity by Example

Solidity in Depth

Security Considerations

Using the compiler

Contract Metadata

Application Binary Interface Specification

Joyfully Universal Language for (Inline) Assembly

Style Guide

Common Patterns


List of Known Bugs

Contributing

Frequently Asked Questions

Basic Questions

Advanced Questions



让我们帮您刊登广告

每个月会有一百万位以上
中文程式设计师看到您

What is the **memory** keyword? What does it do?

The Ethereum Virtual Machine has three areas where it can store items.

The first is "storage", where all the contract state variables reside. Every contract has its own storage and it is persistent between function calls and quite expensive to use.

The second is "memory", this is used to hold temporary values. It is erased between (external) function calls and is cheaper to use.

The third one is the stack, which is used to hold small local variables. It is almost free to use, but can only hold a limited amount of values.

For almost all types, you cannot specify where they should be stored, because they are copied everytime they are used.

The types where the so-called storage location is important are structs and arrays. If you e.g. pass such variables in function calls, their data is not copied if it can stay in memory or stay in storage. This means that you can modify their content in the called function and these modifications will still be visible in the caller.

There are defaults for the storage location depending on which type of variable it concerns:

- state variables are always in storage
- function arguments are in memory by default
- local variables of struct, array or mapping type reference storage by default
- local variables of value type (i.e. neither array, nor struct nor mapping) are stored in the stack

而 struct 中在和局部变量进行赋值操作的时候，是保存成一个引用

← → ↻ ① 不安全 | solidity.readthedocs.io/en/latest/types.html#structs

Structure of a Contract

Types

- Value Types
- Reference Types
 - Data location
 - Arrays
 - Structs**

Mappings

- Operators Involving LValues
- Conversions between Elementary Types

Units and Globally Available Variables

Expressions and Control Structures

Contracts

Solidity Assembly

Miscellaneous

Security Considerations

- Using the compiler
- Contract Metadata
- Contract ABI Specification
- Yul
- Style Guide
- Common Patterns

```
function contribute(uint campaignID) public payable {
    Campaign storage c = campaigns[campaignID];
    // Creates a new temporary memory struct, initialised with the given values
    // and copies it over to storage.
    // Note that you can also use Funder(msg.sender, msg.value) to initialise.
    c.funders[c.numFunders++] = Funder({addr: msg.sender, amount: msg.value});
    c.amount += msg.value;
}

function checkGoalReached(uint campaignID) public returns (bool reached) {
    Campaign storage c = campaigns[campaignID];
    if (c.amount < c.fundingGoal)
        return false;
    uint amount = c.amount;
    c.amount = 0;
    c.beneficiary.transfer(amount);
    return true;
}
```

The contract does not provide the full functionality of a crowdfunding contract, but it contains the basic concepts necessary to understand structs. Struct types can be used inside mappings and arrays and they can itself contain mappings and arrays.

It is not possible for a struct to contain a member of its own type, although the struct itself can be the value type of a mapping member. This restriction is necessary, as the size of the struct has to be finite.

Note how in all the functions, a struct type is assigned to a local variable (of the default storage data location). This does not copy the struct but only stores a reference so that assignments to members of the local variable actually write to the state.

Of course, you can also directly access the members of the struct without assigning it to a local variable, as in `campaigns[campaignID].amount = 0`.

如下是问题代码，struct 在函数中被声明但是没有初始化，根据官方文档中可以知道，struct 在局部变量中默认是存放在 storage 中的，因此可以利用 Unintialised Storage Pointers 的问题，p 会被当成一个指针，并默认指向 slot[0] 和 slot[1]，因此在进行 p.name 和 p.mappedAddress 赋值的时候，实际上会修改变量 testA，test B 的值。

```
pragma solidity ^0.4.0;
contract testContract{

    bytes32 public testA;
    address public testB;

    struct Person {
        bytes32 name;
        address mappedAddress;
    }

    function test(bytes32 _name, address _mappedAddress) public{
        Person p;
        p.name = _name;
        p.mappedAddress = _mappedAddress;
    }
}
```

同理数组也有同样的问题，如下是问题代码

```
pragma solidity ^0.4.0;

contract C {
    uint public someVariable;
```

```

uint[] data;

function f() public {
    uint[] x;
    x.push(2);
    data = x;
}
}

```

0x02 解决方案

结构体 Unintialised Storage Pointers 问题的正确的解决方法是将声明的 struct 进行赋值初始化，通过创建一个新的临时 memory 结构体，然后将它拷贝到 storage 中。

```

pragma solidity ^0.4.0;
contract testContract{

    bytes32 public testA;
    address public testB;

    struct Person {
        bytes32 name;
        address mappedAddress;
    }

    mapping (uint => Person) persons;

    function test(uint _id, bytes32 _name, address _mappedAddress) public{
        Person storage p = persons[_id];
        p.name = _name;
        p.mappedAddress = _mappedAddress;
    }
}

```

数组 Unintialised Storage Pointers 问题的正确解决方法是在声明局部变量 x 的时候，同时对 x 进行初始化操作。

```

pragma solidity ^0.4.0;

contract C {
    uint public someVariable;
    uint[] data;

    function f() public {
        uint[] x = data;
        x.push(2);
    }
}

```

Solidity 编译器开发团队不出意外将在下一个版本（Solidity 0.4.25）中对存在 Unintialised Storage Pointers 问题的代码进行修复，否则将无法通过编译。

开发人员需要关注 Solidity 0.4.25 版本的发布，并且使用 Solidity 0.4.25 编写代码。

最后 本篇未涉及的 mapping 未初始化存储指针的安全问题和案例，期待能够和师傅们一起研究讨论。

0x03 参考链接

《警惕！Solidity 缺陷易使合约状态失控》

<https://mp.weixin.qq.com/s/xex9Eef6Hz5o24sX5vE1Yg>

《Solidity 安全：已知攻击方法和常见防御模式综合列表》

<https://github.com/slowmist/Knowledge-Base/blob/master/solidity-security-comprehensive-list-of-known-attack-vectors-and-common-anti-patterns-chinese.md#%E8%99%9A%E6%8B%9F%E5%8C%96%E5%AD%98%E5%82%A8%E6%8C%87%E9%92%88>

《Solidity 官方文档》

<http://solidity.readthedocs.io/en/v0.4.24/frequently-asked-questions.html>

<http://solidity.readthedocs.io/en/latest/types.html#structs>