HOCHSCHULE
HANNOVER
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS
–
Fakultät IV
Wirtschaft und
Informatik

Vulnerability Detection in Solidity Smart Contracts
A Comparison of Tools and Services

Kurt Merbeth

Bachelor Thesis in "Applied Computer Science"

January 8, 2021

Author: Kurt Merbeth
1168947
Kurt@Merbeth.io

First Examiner: Jussi Salzwedel M.Sc.
Abteilung Informatik, Fakultät IV
Hochschule Hannover
Jussi.Salzwedel@hs-hannover.de

Second Examiner: Prof. Dr. Matthias Hovestadt
Abteilung Informatik, Fakultät IV
Hochschule Hannover
Matthias.Hovestadt@hs-hannover.de

Declaration of Independence

I hereby declare that I have written the submitted Bachelor thesis independently and without outside help, that I have not used any sources or aids other than those indicated by me, and that I have marked any passages taken literally or in terms of content from the materials used as such.

Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die eingereichte Bachelor-Arbeit selbständig und ohne fremde Hilfe verfasst, andere als die von mir angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Werken wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Hanover, January 8, 2021 Signature

Contents

Abstract

This bachelor thesis focuses on different security aspects of smart contracts on the Ethereum blockchain.
The Ethereum blockchain posesses a native software layer, the Ethereum Virtual Machine, that allows for the execution of code in the form of smart contracts. These smart contracts are often written in the programming language Solidity and the deployment onto the blockchain renders them immutable, yet publicly accessible. However, the primary use cases of smart contracts are financial applications.

Thus, it is crucially important to develop secure software. Several tools and services exist that allow for the analysis of Solidity code in order to prevent vulnerabilities and common vectors of attack in smart contracts.

This thesis compares these analyzers to determine their general usage and overall efficiency in finding these common vulnerabilities.
For this purpose, the most common vulnerabilities are discussed, a smart contract based on these vulnerabilities is created and afterward inspected by the analyzers. The main focus is on the completeness of the vulnerabilities found, additional found problems, further information according to these results, and usability.

The comparison reveals that the use of tools and services is overall recommended. They are easy to use and mostly up to date. The analysis duration is adequate, and the results are usually well presented. However, it is noteworthy that none of the single applications was able to find all of the implemented weaknesses. Therefore, it is highly recommended to use the applications complementary to each other in order to cover a broad spectrum of security vulnerabilities and problems.

1. Introduction

Blockchain technology gained a high level of recognition over the past few years, especially through the cryptocurrency Bitcoin. In 2008, Satoshi Nakamoto published a paper that describes a digital currency that can be accessed via a public peer-to-peer network, which stores cryptographically encrypted data records in concatenated blocks and validates their authenticity using a consensus algorithm. [Nak08]

Based on this technology, Vitalik Buterin had the idea for Ethereum, a blockchain with an additional software layer. Buterin published the Ethereum whitepaper in 2013. In contrast to Bitcoin, Ethereum is not just a currency: Its software layer enables the execution of decentralized applications. This layer, termed the Ethereum Virtual Machine, allows for the execution of code called a smart contract. One of the main use cases of smart contracts are financial applications. [Vit20]

Because smart contracts are stored immutably and publicly on the blockchain, it is easy for attackers to find and exploit possible vulnerabilities. This, in addition to their usage as financial applications, makes these programs attractive to attackers.

In the past there have already been several attacks on smart contracts. For example the DAO hack[1], in which an attacker stole 3,641,694 ETH[3], or the Parity Wallet hack[2], where about 150,000 ETH[3] were stolen. These examples show the importance of writing secure smart contracts.

Currently, there are several applications to check Solidity code for security vulnerabilities. This bachelor thesis tests and compares recommend Solidity software analyzer.

To prepare for this topic, the following second chapter covers blockchain basics in general, as well as basics of the Ethereum blockchain in specific. Additionally, smart contracts will be explained, as well as their native programming language Solidity.

A description of the most important security vulnerabilities and an explanation using short examples follow in chapter three.

Chapter four introduces the tools and services for analyzing smart contracts. In order to test these analyzers, designing a smart contract with the previously mentioned vulnerabilities is necessary.

[1] The DAO Heist FAQ: https://media.consensys.net/the-dao-heist-faq-ef95a7f310f0

[2] The Parity Wallet Hack Explained: https://blog.openzeppelin.com/on-the-parity-wallet-multisig-hack-405a8c12e8f7/

[3] Today on January, 08 2021, 1 ETH is worth $1260.

Chapter five describes this smart contract in detail, as well as explains the implemented functions in specific. Additionally, the intentionally included vulnerabilities are pointed out.

Afterward, chapter six discusses the results of the smart contract analysis: The discussion includes usability, found vulnerabilities, additional identified problems, and the duration of the analysis.

The different analyzer and their results are shown in the seventh chapter. This includes usage, general characteristics, and all results discovered.

Finally, chapter eight summarizes the results of the previous analysis. The importance of vulnerability detection is emphasized, as well as the experience with testes analyzers. In the end, a recommendation is made based on the finding of this thesis on which tools and services should be used to develop secure Solidity code.

2. Basics

This chapter covers the essential basics of blockchain technology in general and Ethereum in particular. After a solid understanding has been established, this section discusses the basics of smart contracts and the Solidity programming language.

The following chapters and the core of this bachelor thesis build on these basics.

2.1. What is a Blockchain?

The blockchain is a type of decentralized database. Unlike traditional databases, they are organized in distributed peer-to-peer networks and store their data in blocks. Each block has a header that stores metadata. This metadata includes, among other data, the hash of the previous block. The hash is calculated cryptographically from the entire block, including the transactions contained in the block header. It is needed to verify the authenticity of the previous block. Transactions are the main content of blocks. They are data signed by a sender and addressed to a specific recipient. The blockchain network transmits these transactions and records them in blocks. [AW18]If a transaction is manipulated afterward, the hash would change and expose the manipulation. [Vos20] Each newly attached block points to the previous block. The resulting chain of blocks is the eponym for blockchain technology. [Mal18]

Figure 2.1 shows as an example of the chaining and the structure of blocks.

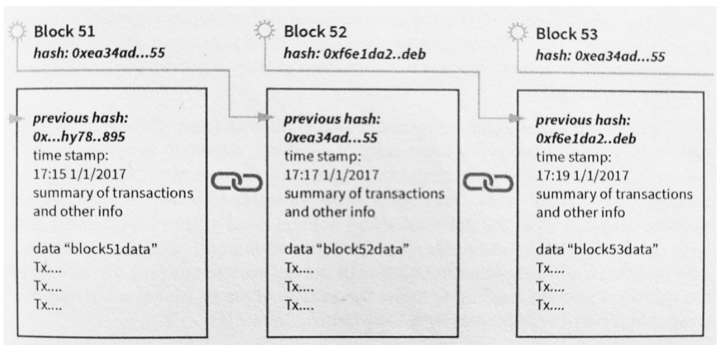


Figure 2.1.: How Blocks of Transactions are Chained [Vos20]

To execute transactions, an account is needed. *Note: Accounts are often called wallets.* Accounts consist of a key pair consisting of a public and a private key. Compared to conventional accounts of internet services, accounts on the blockchain do not store any personal data. The public key represents the account address that is used to receive transactions. The private key is used to access the related account address and to sign the data of its transactions. As knowing the private key allows spending the funds correlated to the public addresses account, this key should always be kept secret. It is possible to create as many accounts as wanted. [Dan17]

Blockchains have the possibility to implement a native currency, termed cryptocurrencies. These currencies are not implemented in every blockchain, but are implemented in Bitcoin and Ethereum. Bitcoin's currency is called "Bitcoin", and Ethereum's currency is called "Ether". They are hard-coded into the blockchain and needed to execute transactions. [Die16]
For every transaction executed on a blockchain network, it is necessary to pay a certain amount of cryptocurrency to include the transaction into a block. [Dan17]

Blockchains have a consensus mechanism implemented to confirm the authenticity of transactions. For example, Bitcoin and Ethereum both have implemented the Proof-of-Work (PoW) algorithm.

Servers that participate in the network are called nodes.
All nodes involved in the consensus mechanism receive the transactions. These are first validated and then written into a block, together with a pseudo-random number.
Nodes generate a hash from this block, which must match a from the network previously defined scheme (called difficulty). If the block corresponds to this difficulty, the node broadcasts this block to the other nodes in the network. If they successfully validate that the hash of the block's transactions and the pseudo-random number match the difficulty, the new block is appended onto the chain. The node that found the block receives a reward for its work in the form of the blockchain's native cryptocurrency.

If the new block does not conform to the specifications, it is discarded and the process starts again. [Vos20] The process of finding a new block is called 'mining' and all participants who take part in this are called 'miners'.

2.2. The Ethereum Blockchain

As already mentioned in the previous chapter, there are different kinds of blockchains. Nakamoto described the Bitcoin blockchain in his whitepaper as "A Peer-to-Peer Electronic Cash System." [Nak08] In other words, as a pure digital currency.

"A Next-Generation Smart Contract and Decentralized Application Platform"

— Vitalik Buterin

In 2013, Buterin published a whitepaper on Ethereum, a blockchain intended to be more than just a digital currency. Ethereum is a blockchain specifically designed for decentralized applications. While it also has a cryptocurrency called Ether, it provides the ability to execute software on the Ethereum Virtual Machine (EVM). The EVM is turing-complete, "this means that EVM code can encode any computation that can be conceivably carried out, including infinite loops." [Vit20]

For interaction with the Ethereum blockchain, namely the execution of transactions and the interaction with the deployed smart contracts, 'gas' is needed. 'Gas' is often described as 'fuel' that powers the blockchain. It is paid in Wei, the smallest Ether unit.

In his whitepaper, Buterin describes the following three categories for decentralized applications on the Ethereum blockchain.

- *Financial Applications*
  Users are offered more advanced options for managing their finances by concluding contracts without the need for a central instance.
  "This includes sub-currencies, financial derivatives, hedging contracts, savings wallets, wills, and ultimately even some classes of full-scale employment contracts." [Vit20]
- *Semi-Financial Applications*
  Values are inherently part of the application, yet the main focus is on non-financial program logic.
- *Non-Financial Applications*
  Applications without a financial background. The whitepaper mentions online voting and decentralized governance as examples.

In order to support the Ethereum ecosystem, the Ethereum Foundation was founded. The Foundation is a non-profit organization that focuses on the growth of the Ethereum community. For this purpose, they sponsor projects and organize conferences related to Ethereum. [Eth20b]

2.2.1. Smart Contracts

As mentioned, software deployed on the Ethereum blockchain and executed on the EVM is called smart contract. Ethereum provides two types of account: Externally owned accounts (EOA) created by a user and contract accounts. [AW18]
Thus, smart contracts also refer to an address and can receive Ether.

Two features should be highlighted: Smart contracts are immutable and deterministic.

Immutable means that the code of a deployed smart contact cannot be edited after it was deployed to the blockchain. It is necessary to deploy a smart contract again to change its code. After deployment, the smart contract receives a new address to interact with.

Deterministic describes that the execution of a smart contract with the same input will always get identical results.

Every interaction with a smart contract requires a transaction and only transactions are able to modify the current state of the blockchain.
As mentioned in chapter 2.2, every miner has to validate transactions. It is essential that program executions are deterministic and always change the state in the same way. [AW18]

2.2.2. Solidity

There are several programming languages to write EVM compatible software.
According to the Ethereum developer documentation, Solidity and Vyper are the most active programming languages for smart contracts. [Eth20d]

This paper discusses the object-oriented, high-level programming language Solidity.
It was created by Dr. Gavin Woods and inspired by the programming languages C++, Python, and JavaScript. [AW18] [Eth20d]
Solidity is a programming language specifically for smart contracts and finds no other implementation outside of Ethereum. It supports inheritance, complex types, and the functionality of libraries. [Fou20b]

The first version of Solidity was released in 2015 and is constantly being developed further. Thus, Solidity is still a very young and constantly evolving programming language. [Fou20a]

3. Vulnerabilities

There are a few things to keep in mind when programming smart contracts with Solidity. This chapter presents the most common security vulnerabilities. These have been selected based on the information for developer and the wiki of the Ethereum Foundation. As the Ethereum Foundation refers to publications by ConsenSys Diligence and the Smart Contract Weakness Classification (SWC) regarding security concerns, these were also included in the selection process of the most common and most important vulnerabilities. Further information about ConsenSys and the SWC can be found in chapter 4.1.1.

The following vulnerabilities are described and explained with short examples. They are the basis for the smart contract presented in chapter 5 and the later conducted analysis.

3.1. External Calls

An external call is a method call that exits a contract in order to call another function in an external contract. For this purpose, Solidity offers various low-level call methods that can be directly addressed. [Eth20e]

Solidity provides the following methods to execute an external call:

- **address.call()**
  Calls a function of another contract.
- **address.delegatecall()**
  "...a contract can dynamically load code from a different address at runtime. Storage, current address and balance still refer to the calling contract, only the code is taken from the called address." [Fou20b]
- **address.callcode() (deprecated)**
  Since Solidity 0.5.0: "Function callcode is now disallowed (in favor of delegatecall). It is still possible to use it via inline assembly." [Fou20d]
- **address.send()**
  `'address.send()'` is used to send an amount of Ether (in Wei) to an address. [Fou20b]

The following sections describe the problems of the various external call functions in more detail. The `'callcode()'` function is not mentioned since it is deprecated and replaced by `'delegatecall()'`.

3.1.1. Unchecked Call Return Value

As mentioned before, the functions of external smart contracts can be executed with `'call()'`. However, it will not be checked if the call was successful. A wrong return value or an exception has no impact on the further program flow but can lead to misbehavior in the program. [Dil20c]

Line 10 of the example code 3.1 demonstrates a `'call()'` where the return value is not checked. It is recommended to ensure the correctness of the call by checking its return value to avoid errors. For this purpose the `'require()'` function can be used(line 6), which throws an exception if an error occurs.

```
 1  pragma solidity 0.4.25;
 2
 3  contract ReturnValue {
 4
 5    function callchecked(address callee) public {
 6      require(callee.call());
 7    }
 8
 9    function callnotchecked(address callee) public {
10      callee.call();
11    }
12  }
```

Code 3.1: Unchecked Return Value [Dil20c]

3.1.2. Delegatecall to Untrusted Callee

As described, the `'delegatecall()'` function is used to load code from an external smart contract dynamically. It is important to note that storage, current address, and balance of the calling contract are used for method calls of the loaded contract. This may cause a malicious smart contract to modify the storage values of the calling contract. [Dil20c]

The following code 3.2 shows an example attack using a `'delegatecall()'`.

The vulnerable `'Caller'` contract has two functionalities. When the constructor gets executed, the `'owner'` variable in line 6 initializes with the smart contract deployers address (msg.sender). Furthermore, the function `'call'` calls in line 10 the `'functionA()'` of an external smart contract by using `'delegatecall()'`.

An attacker is now able to modify the `’owner’` variable of the `’Caller’` contract.
For this purpose the `’Callee’` contract is required. It implements the `’owner’` variable and the `’functionA()’`, which reinitialize the value of the `’owner’` variable with the `’msg.sender’` (line 23).

If the attacker invokes the `’call()’` function of the `’Caller’` contract and passes the `’Callee’` contract address, the `’owner’` variable of the `’Caller’` contract will be overwritten with the attacker’s address.
This could enable an attacker to gain unauthorized access to restricted functions of the `’Caller’` contract.

```
 1  // vulnerable contract
 2  contract Caller {
 3      address public owner;
 4
 5      constructor() public {
 6          owner = msg.sender;
 7      }
 8
 9      function call(address _contract) public {
10          _contract.delegatecall(abi.encodeWithSignature("functionA()"));
11      }
12  }
13
14  // attacker contract
15  contract Callee {
16      address public owner;
17
18      constructor() public {
19          owner = msg.sender;
20      }
21
22      function functionA() public {
23          owner = msg.sender;
24      }
25  }
```

Code 3.2: Example Delegatecall Attack

3.1.3. Failed Call using Send

The possibility that an external call will fail is always present.
When calling the `’send()’` function, an error in execution can lead to a Denial of Service condition.
This problem is skipped and explained in more detail in chapter 3.2.1.

3.2. Denial of Service

A Denial of Service (DoS) is a condition in which a service becomes unavailable due to a failure. Unlike conventional software, smart contracts cannot simply be resetted. If a DoS condition occurs, it can lead to a permanent malfunction of the smart contract. [NCC18]

A Denial of Service can be caused accidentally or on purpose. This chapter explains two possible causes of DoS in more detail.

3.2.1. DoS with (Unexpected) revert

This problem occurs with the low-level call method `'send()'`. Unlike the `'call()'` method (described in chapter 3.1.1), the `'send()'` method can return an incorrect value which can affect the whole application behavior.

Code 3.3 implements the function `'refundAll()'`. It iterates through an address array (line 6) and sends a payment to each address in that array using `'send()'` (line 7).
All payments are combined into one transaction after the for-loop is finished.
To guarantee that no payout fails, the return value of `'send()'` is checked by using `'require()'`.
If an address reverts the payment, the entire loop will be interrupted, and the whole transaction aborts.

As long as the corrupt address is contained in the array, `'refundAll()'` cannot be executed successfully. This causes no address from the array receives a payment.

```
1 address[] private refundAddresses;
2 mapping (address => uint) public refunds;
3
4 // bad
5 function refundAll() public {
6   for(uint x; x < refundAddresses.length; x++) { // arbitrary length
       iteration based on how many addresses participated
7     require(refundAddresses[x].send(refunds[refundAddresses[x]])) //
         doubly bad, now a single failure on send will hold up all funds
8   }
9 }
```

Code 3.3: Simple Auction [Con20a]

3.2.2. DoS with Block Gas Limit

In Ethereum, each block has a gas limit. All miners in the network determine the block gas limit. The more data is included in a single transaction, the more expensive it becomes. If a transaction exceeds this gas limit, it cannot be included in a block and will fail. [Con20a] [AW18]

The following example 3.4 iterates through an array to pay out the containing addresses (lines 10-11). Because every payout causes gas fees and the array can become arbitrarily large, it is possible that the block gas limit can exceed and the transaction fails. This can happen due to an accident or due to an attack where the attacker adds a large number of addresses to the array.

```
 1  struct Payee {
 2    address addr;
 3    uint256 value;
 4  }
 5
 6  Payee[] payees;
 7
 8  function payOut() {
 9    uint256 i = 0;
10    while (i < payees.length && msg.gas > 200000) {
11      payees[i].addr.send(payees[i].value);
12      i++;
13    }
14  }
```

Code 3.4: Loop over an array [Con20a]

3.3. Reentrancy

The chapter 3.1 already described external calls and their usage.
A special danger is the reentrancy attack. This is a recursive call back of a malicious contract into its caller. This attack is made possible by invoking the `'fallback()'` function on an external call, which in turn can execute malicious code and make a call back to its caller.

This attack requires two smart contracts to explain the problem.
First, the victim of the attack `'simpleBank'` (Code 3.5) is shown. The mapping `'balance'` in line 2 stores the balance for each wallet address.
In lines 4-6 a simple `'deposit()'` function is implemented. It receives funds and stores the amount into the `'balance'` mapping.
And finally a `'withdraw()'` function in lines 8-13. It verifies that enough funds are deposited before they can be withdrawn. Afterward, an external call is executed, and the balance will be sent to the address of the `'deposit()'` function caller. If the function call is successful, the `'amount'` will be reduced from the `'balance'` mapping.

```
 1 contract simpleBank {
 2   mapping (address => uint) private balance;
 3
 4   function deposit() public payable {
 5     balance[msg.sender] += msg.value;
 6   }
 7
 8   function withdraw(uint amount) public {
 9     if (balance[msg.sender]>= amount) {
10       (bool sent, ) = msg.sender.call{value: amount}("");
11       require(sent, "Failed to send");
12       balance[msg.sender]-=amount;
13     }
14   }
15 }
```

Code 3.5: simpleBank reentrancy

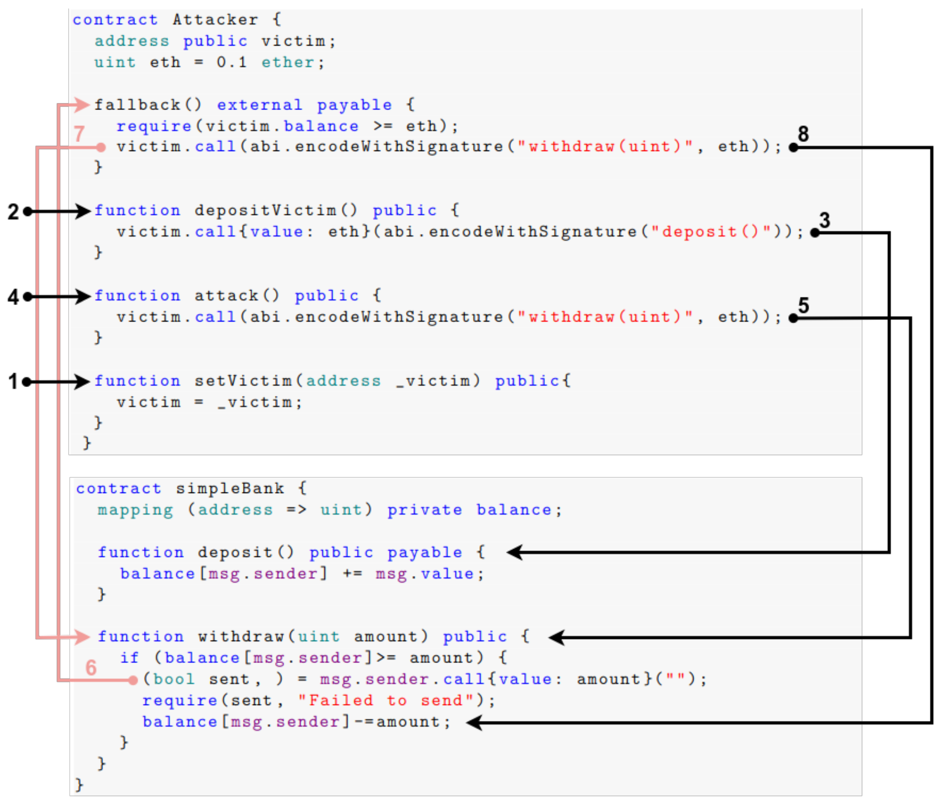
On the other side is the 'Attacker' contract (Code 3.6).

In line 2 the variable for the address of the 'victim' contract is declared, which will be set using the 'setVictim()' function in line 18.
The 'fallback()' function in line 5 will later perform the reentrancy attack. The variable 'eth' defines how much of the 'simpleBank' balance should be withdrawn. Afterward, the 'fallback()' function checks if the 'victim' has enough balance and then it calls the 'withddraw()' function of the 'vitim'. In order to perform the attack, first some Ether must sent to the 'simpleBank' contract. This can be done with the 'depositVictim()' function in line 10.
With the 'attack()' function the attack can be started. This can be simply done by calling the 'withdraw()' function of the 'victim' with an Ether amount as parameter.

```
 1 contract Attacker {
 2   address public victim;
 3   uint eth = 0.1 ether;
 4 
 5   fallback() external payable {
 6     require(victim.balance >= eth);
 7     victim.call(abi.encodeWithSignature("withdraw(uint)", eth));
 8   }
 9 
10   function depositVictim() public {
11     victim.call{value: eth}(abi.encodeWithSignature("deposit()"));
12   }
13 
14   function attack() public {
15     victim.call(abi.encodeWithSignature("withdraw(uint)", eth));
16   }
17 
18   function setVictim(address _victim) public{
19     victim = _victim;
20   }
21  }
```

Code 3.6: simpleBank Attacker

For a better understanding of the reentrancy attack in figure 3.1 shows the step-by-step process of the attack in detail. At the arrows marked in red, the reentrancy attack can be performed.



```
contract Attacker {
  address public victim;
  uint eth = 0.1 ether;

  fallback() external payable {
    require(victim.balance >= eth);
    victim.call(abi.encodeWithSignature("withdraw(uint)", eth));
  }

  function depositVictim() public {
    victim.call{value: eth}(abi.encodeWithSignature("deposit()"));
  }

  function attack() public {
    victim.call(abi.encodeWithSignature("withdraw(uint)", eth));
  }

  function setVictim(address _victim) public{
    victim = _victim;
  }
}
```

```
contract simpleBank {
  mapping (address => uint) private balance;

  function deposit() public payable {
    balance[msg.sender] += msg.value;
  }

  function withdraw(uint amount) public {
    if (balance[msg.sender]>= amount) {
      (bool sent, ) = msg.sender.call{value: amount}("");
      require(sent, "Failed to send");
      balance[msg.sender]-=amount;
    }
  }
}
```

Figure 3.1.: Procedure of a Reentrancy Attack

- **Step 1:** Set the address of the victim contract (`'simpleBank'`).
- **Step 2:** Execute `'depositVictim()'`.
- **Step 3:** Deposit Ether in the `'simpleBank'` contract.
- **Step 4:** Execute `'attack()'` function.
- **Step 5:** Call the `'withdraw()'`-function of `'simpleBank'` and refund the deposited Ether.
- **Step 6:** Checks if enough Ether has been deposited and sends the Ether to the attacker.
  This automatically executes the `'fallback()'` function.
- **Step 7:** Re-enter the `'withdraw()'` function of `'simpleBank'` after checking the `'simpleBank'` balance.
  Steps 6 and 7 loop until the balance of the `'simpleBank'` contract runs out of Ether.
  Note: Until step 8 is reached, the if-statement is true and the reentrancy attack is possible.
- **Step 8:** After the `'simpleBank'` contract runs out of Ether, the recursive call is interrupted and the `'balance'` mapping reduced.

3.4. Access Control

Since Ethereum is designed with a special emphasis for financial applications and smart contracts are stored publicly and executable by everyone, it is crucial to pay attention toward access restrictions.
Wrong permissions can not only cause malfunctions in the application, they can also enable attackers to steal large amounts of money.

In this section, three typical scenarios are explained that can have significant consequences due to a lack of access restrictions.

3.4.1. Unprotected Owner Setter

When smart contracts are deployed on the blockchain, it is common practice to set the address of the contract deployer to a variable. This is important to protect access to sensitive functions.
An example would be a 'Circuit Breaker', which can pause the functionality of a smart contract in case of a security gap or general malfunction. [Eth20e]

The code example 3.7 a simplified version of an initialization function is shown.

Since the `'initContract()'` is not protected, it can be called by anyone.
An attacker can become the smart contract owner simply by calling `'initContract()'` and thereby gain access to sensitive functions. [NCC18]

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

Code 3.7: Init function [NCC18]

3.4.2. Unprotected Ether Withdrawl

One of smart contracts' basic function is to store Ether, and it is important that nobody gets unauthorized access to the stored balance.
An attacker with access to an unprotected withdraw function can cause significant financial damage.

Code 3.8 shows in line 5 of the `'SimpleEtherDrain'` contract an unprotected withdraw function named `'withdrawAllAnyone()'`. The `'fallback()'` function implemented in line 9 enables the contract to receive Ether.
The attacker can easily withdraw these Ethers by calling the `'withdrawAllAnyone()'` function and cause losses.

```
 1 pragma solidity ^0.4.22;
 2
 3 contract SimpleEtherDrain {
 4
 5   function withdrawAllAnyone() {
 6     msg.sender.transfer(this.balance);
 7   }
 8
 9   function () public payable {
10   }
11 }
```

Code 3.8: Simple Ether drain [Dil20c]

3.4.3. Unprotected Selfdestruct

The only way to disable the functionality of a smart contract is the `'selfdestruct()'` function. This may be necessary if a new version of a smart contract is deployed, and interaction with the old one needs to be disabled. After executing `'selfdestruct()'` it is no longer possible to interact with the contract. It is only stored in the previous blocks. The function `'selfdestruct()'` can be passed an address as parameter. The complete contract balance will be paid to this address. [Fou20b]

The contract in code 3.9 has an unprotected `'suicideAnyone()'` function, which executes `'selfdestruct()'`. By calling this function, an attacker can destroy the contract and get the entire balance.

```
pragma solidity ^0.4.22;
contract SimpleSuicide {
  function sudicideAnyone() public {
    selfdestruct(msg.sender);
  }
}
```

Code 3.9: SimpleSuicide [Dil20c] SWC-106

3.5. Integer Overflow and Underflow

Integer overflows and underflows occur when the value range of an integer exceeds or underruns. This can lead to a malfunction in the application.

Solidity offers the possibility to use unsigned integers in smart contracts. For example, an unsigned integer with a 2^8 (uint8) size has 256 values. This results in a range of values from 0 to 255.

3.5.1. Integer Overflow

The following example explains an integer overflow with a variable from type uint8. The variable '`number`' is declared as uint8 and initialized with the maximum value of 255. By incrementing the variable '`number`' by 1, the value range is exceeded, and the variable starts counting again at 0.

```
// integer overflow
uint8 number = 255;
number +=1;      // value of number: 0
```

Code 3.10: Simple Integer Overflow

3.5.2. Integer Underflow

An integer underflow is similar to the previously mentioned overflow. In the following example, '`number`' is initialized with 0 and decremented by 1. In this case, the arithmetic operations result is 255, since the minimum value falls below 0 and the counter is set to the maximum value.

```
// integer underflow
uint8 number = 0;
number -=1;      // value of number: 255
```

Code 3.11: Simple Integer Underflow

Code 3.12 shows an integer underflow in the '`withdraw()`' function.
Line 2 checks if the user has enough '`balances`'. If the user tries to withdraw a larger '`_amount`', the result is less than 0 and becomes positive again due to the underflow. This allows the user to withdraw the entire contract balance.

```
1 function withdraw(uint _amount) {
2   require(balances[msg.sender] - _amount > 0);
3   msg.sender.transfer(_amount);
4   balances[msg.sender] -= _amount;
5 }
```

Code 3.12: Infinite Token Withdraw [NCC18]

3.6. Bad Randomness

The consensus mechanism of Ethereum requires every transaction in a new block to be validated by more than 50% of its miners. If the miners do not reach this consensus, the new block will not be appended.

To achieve identical results when validating transactions, the result of each code execution must be equal. Therefore, it is necessary to generate pseudo-random numbers that are identical for all miners for each execution.

It is common practice for smart contracts to use the block hash, the timestamp of a block, or a seed phrase to generate pseudo-random numbers. This approach works because these values are identical for all miners, and therefore, the result of the execution of a transaction is always the same. [ABC17]

This chapter describes two approaches used to implement randomness and explains how an attacker may exploit both of them.

3.6.1. With Private Seed

The idea behind a random number generated by a private seed is that the value of a private variable is not accessible from an external source.

The code example 3.13 shows a smart contract snippet that has implemented this approach. In line 6, a private `'seed'` and the current `'iteration'` are used to create a hash that is used as `'randomNumber'`.

Overall, there are two problems with this approach.
First, the last `'iteration'` can be read publicly, and therefore the next value is predictable. Second, the private `'seed'` can also be seen publicly on the blockchain. As a transaction is needed to initialize its value, this value is stored on the blockchain and accessible without restriction.

Because these two values are not a true secret, an attacker can pre-compute the `'randomNumber'` and predict the result of the if-statement in line 7.

```
 1 uint256 private seed;
 2 
 3 function play() public payable {
 4     require(msg.value >= 1 ether);
 5     iteration++;
 6     uint randomNumber = uint(keccak256(seed + iteration));
 7     if (randomNumber % 2 == 0) {
 8         msg.sender.transfer(this.balance);
 9     }
10 }
```

Code 3.13: Randomness with Private Seed [NCC18]

3.6.2. With Block Hash

Another approach is to generate a random number by a block hash.
This only works with the hash of past blocks.

The current block hash is always 0 because the current transaction is included in the calculation of the current block hash. Even if the block is more than 256 blocks in the past, the `'block.blockhash()'` function generates the value 0.

To execute the `'block.blockhash()'` function in line 3 (Code 3.14), a `'blockNumber'` of a past block is needed.

An attacker can execute a smart contract during the same transaction and read the `'blockNumber'`, calculate the `'block.blockhash()'` and predict the result of the if-statement.

```
1 function play() public payable {
2   require(msg.value >= 1 ether);
3   if (block.blockhash(blockNumber) % 2 == 0) {
4     msg.sender.transfer(this.balance);
5   }
6 }
```

Code 3.14: Randomness with Block Hash [NCC18]

4. Vulnerability Detection Tools and Services

The Ethereum Foundation recommends seven tools and services for developers to inspect their smart contracts for security vulnerabilities. These are Mythril, Slither, Securify, Manticore, MythX, SmartContract.Codes, and the ERC20 Verifier. [Eth20c]

SmartContract.Codes is a search engine for previously verified source codes, and the ERC20 Verifier is a tool to check if a smart contract complies with the ERC20 standard. Because these two tools are not suitable to search for the vulnerabilities mentioned in chapter 3, they will not be discussed in this thesis.

However, since the Remix IDE was used to create, deploy and interact with the smart contract described in chapter 5, its static code analyzer will be examined in this chapter as well, in addition to the recommendations made by the Ethereum Foundation.

4.1. Tools

The following section introduces four command-line tools for the analysis of Solidity smart contracts. This section gives a brief overview of the analyzers and their developers.

4.1.1. Mythril

Mythril[1] is a command-line interface for analyzing EVM compatible smart contracts. It is a part of MythX[2](Chapter 4.2.2) and thus is part of ConsenSys[3] Diligence.

ConsenSys describes itself as a "leading Ethereum software company" and offers solutions from areas like decentralization, decentralized finance, security, development, and much more.
Specifically the ConsenSys Diligence focuses on blockchain security. [Con20b]

[1]Mythril Github Repository: https://github.com/ConsenSys/mythril
[2]MythX Website: https://mythx.io
[3]ConsenSys Website: https://consensys.net

Even the Ethereum Foundation and the Solidity documentation refers to ConsenSys regarding security. [Eth20c] [Fou20c]

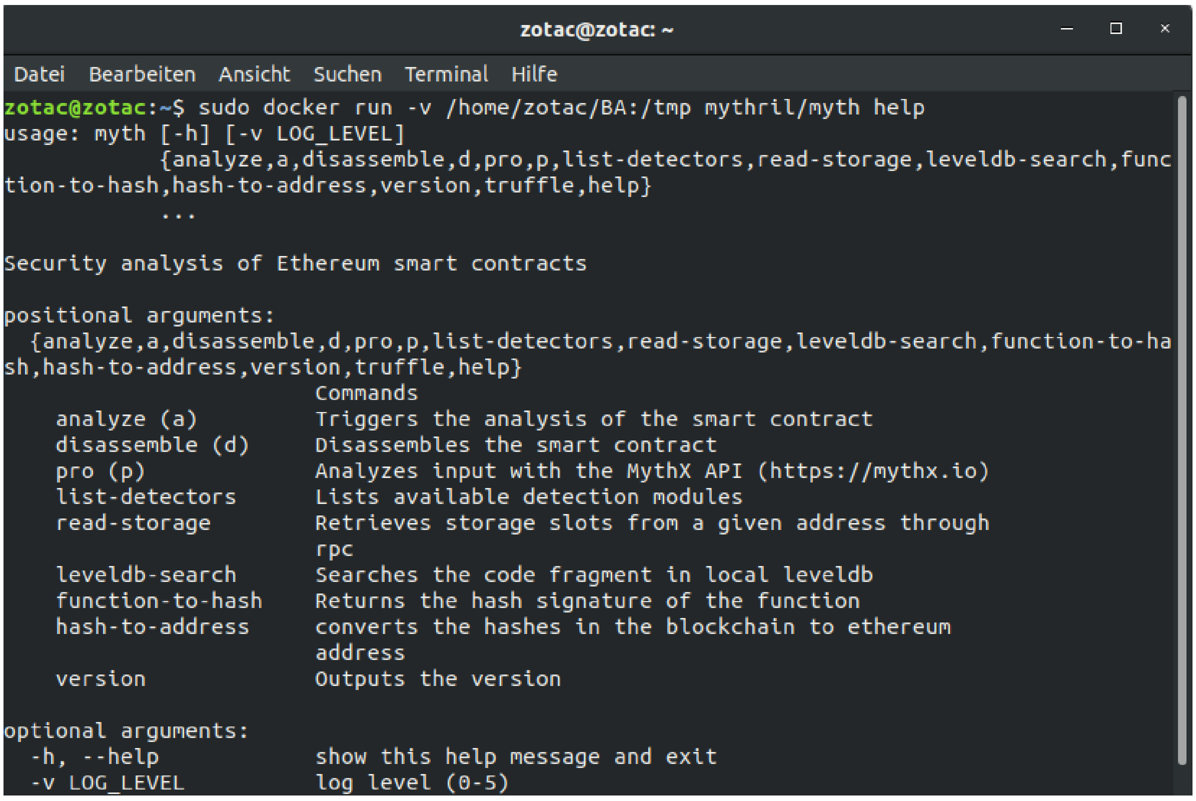
The MythX team also maintains the SWC registry, a registry for classifying Ethereum based vulnerabilities. [Dil20c]

Mythril is provided as a Docker container or for installation using the python package installer pip3.

Mythril can analyze smart contracts and EVM bytecode. It is also used in MythX, a service discussed in chapter 4.2.2. [Dil20a]

Furthermore, Mythril is compatible with the latest Solidity version and the Truffle Suite[4].

The command '`myth analyze <solidity-file>`' starts the smart contract analysis.



```
zotac@zotac:~$ sudo docker run -v /home/zotac/BA:/tmp mythril/myth help
usage: myth [-h] [-v LOG_LEVEL]
            {analyze,a,disassemble,d,pro,p,list-detectors,read-storage,leveldb-search,func
tion-to-hash,hash-to-address,version,truffle,help}
            ...

Security analysis of Ethereum smart contracts

positional arguments:
  {analyze,a,disassemble,d,pro,p,list-detectors,read-storage,leveldb-search,function-to-ha
sh,hash-to-address,version,truffle,help}
                        Commands
    analyze (a)         Triggers the analysis of the smart contract
    disassemble (d)     Disassembles the smart contract
    pro (p)             Analyzes input with the MythX API (https://mythx.io)
    list-detectors      Lists available detection modules
    read-storage        Retrieves storage slots from a given address through
                        rpc
    leveldb-search      Searches the code fragment in local leveldb
    function-to-hash    Returns the hash signature of the function
    hash-to-address     converts the hashes in the blockchain to ethereum
                        address
    version             Outputs the version

optional arguments:
  -h, --help            show this help message and exit
  -v LOG_LEVEL          log level (0-5)
```

Figure 4.1.: Screenshot: Mythril

[4]Truffle is a popular developer suite for building applications based on Ethereum
Truffle Website: https://trufflesuite.com

4.1.2. Slither

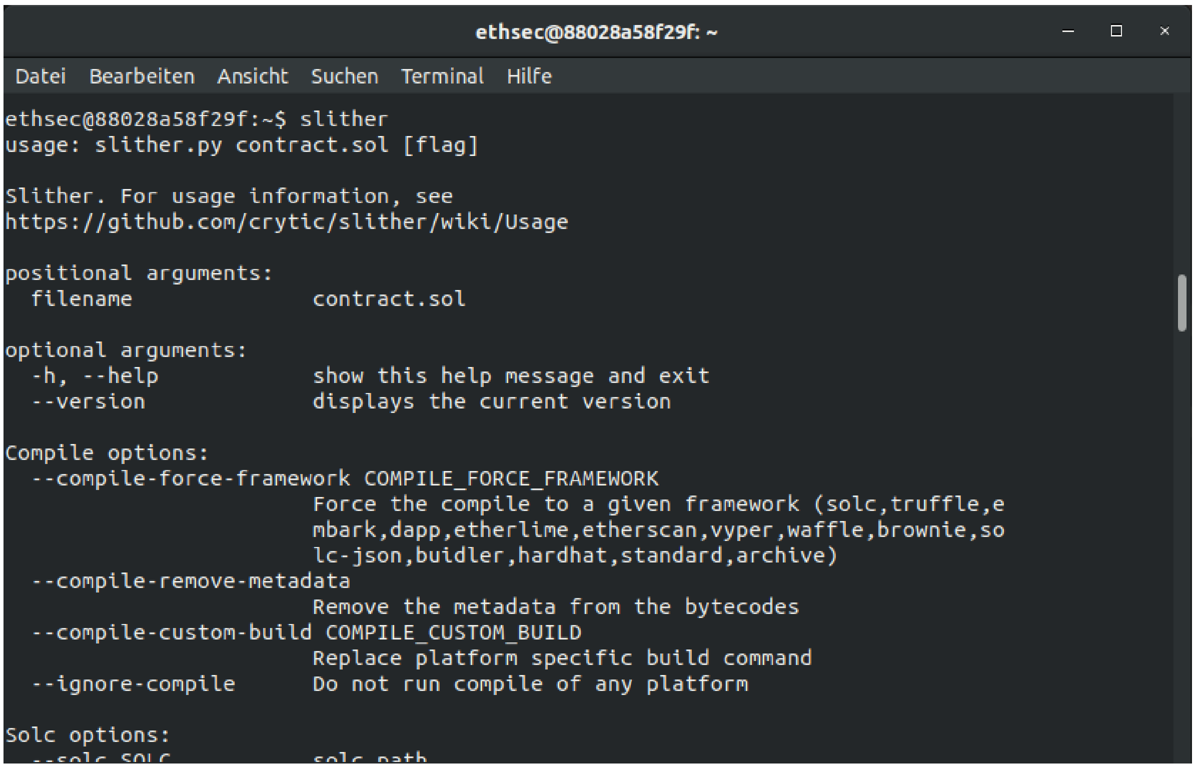
Slither[5] is a static code analyzer for Solidity and is maintained by Trail of Bits[6].

Trail of Bits is a security company with 54 employees that provides security services to companies such as Facebook and the Defense Advanced Research Projects Agency[7] (DARPA). Their areas of focus include reverse engineering, cryptography, virtualization, malware, and software exploits. [oBI20a]

Slither is delivered as a command line interface (CLI). It analyzes smart contracts since Solidity version 0.4.26 using 72 different detectors[8]. Slither can be used as a Docker container, or it can also be installed using the python package installer pip3. Also, Slither is compatible with different Ethereum developer frameworks.

Besides source code files, Slither is able to also analyze already deployed contracts for vulnerabilities by specifying its address. [Cry20]

The command **'slither <solidity-file>'** starts the smart contract analysis.



```
ethsec@88028a58f29f:~$ slither
usage: slither.py contract.sol [flag]

Slither. For usage information, see
https://github.com/crytic/slither/wiki/Usage

positional arguments:
  filename              contract.sol

optional arguments:
  -h, --help            show this help message and exit
  --version             displays the current version

Compile options:
  --compile-force-framework COMPILE_FORCE_FRAMEWORK
                        Force the compile to a given framework (solc,truffle,e
                        mbark,dapp,etherlime,etherscan,vyper,waffle,brownie,so
                        lc-json,buidler,hardhat,standard,archive)
  --compile-remove-metadata
                        Remove the metadata from the bytecodes
  --compile-custom-build COMPILE_CUSTOM_BUILD
                        Replace platform specific build command
  --ignore-compile      Do not run compile of any platform

Solc options:
```

Figure 4.2.: Screenshot: Slither

[5]Slither Github Repository: https://github.com/crytic/slither
[6]Trail of Bits website: https://trailofbits.com
[7]Defense Advanced Research Projects Agency: Belongs to the United States Department of Defense Website: https://darpa.mil
[8]List of 71 detectors: https://github.com/crytic/slither

4.1.3. Securify2

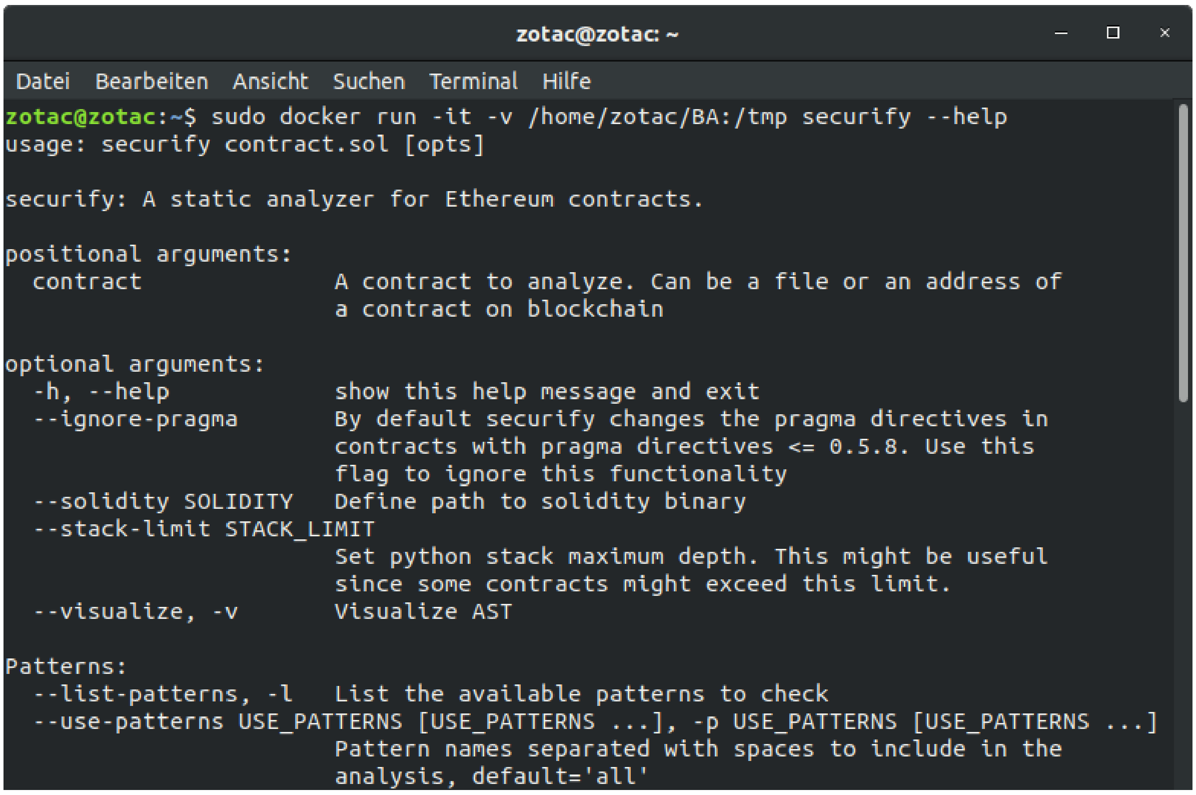
Securify2[9] (called Securify in the further paper) is also a CLI application for vulnerability detection in Solidity source code.

Responsible for Securify is the "Secure, Reliable, and Intelligent Systems Lab" (SRI) at ETH Zurich[10]. The SRI is part of the Department of Computer Science at the Zurich University of Applied Sciences.

They "work at the broad intersection of machine learning and formal reasoning with applications to safe artificial intelligence, probabilistic and quantum programming, AI-driven coding, security, and others." [ETH20a]

Securify is available via Github as a Dockerfile or for manual installation.

It uses 37 different patterns for vulnerability detection, most of which originate from the SWC registry. However, Securify only supports the Solidity versions 0.5.8 to 0.5.12. [Sec20]



```
zotac@zotac:~$ sudo docker run -it -v /home/zotac/BA:/tmp securify --help
usage: securify contract.sol [opts]

securify: A static analyzer for Ethereum contracts.

positional arguments:
  contract              A contract to analyze. Can be a file or an address of
                        a contract on blockchain

optional arguments:
  -h, --help            show this help message and exit
  --ignore-pragma       By default securify changes the pragma directives in
                        contracts with pragma directives <= 0.5.8. Use this
                        flag to ignore this functionality
  --solidity SOLIDITY   Define path to solidity binary
  --stack-limit STACK_LIMIT
                        Set python stack maximum depth. This might be useful
                        since some contracts might exceed this limit.
  --visualize, -v       Visualize AST

Patterns:
  --list-patterns, -l   List the available patterns to check
  --use-patterns USE_PATTERNS [USE_PATTERNS ...], -p USE_PATTERNS [USE_PATTERNS ...]
                        Pattern names separated with spaces to include in the
                        analysis, default='all'
```

Figure 4.3.: Screenshot: Securify

[9]Securify2 Github Repository: https://github.com/eth-sri/securify2
[10]SRI website: https://www.sri.inf.ethz.ch/

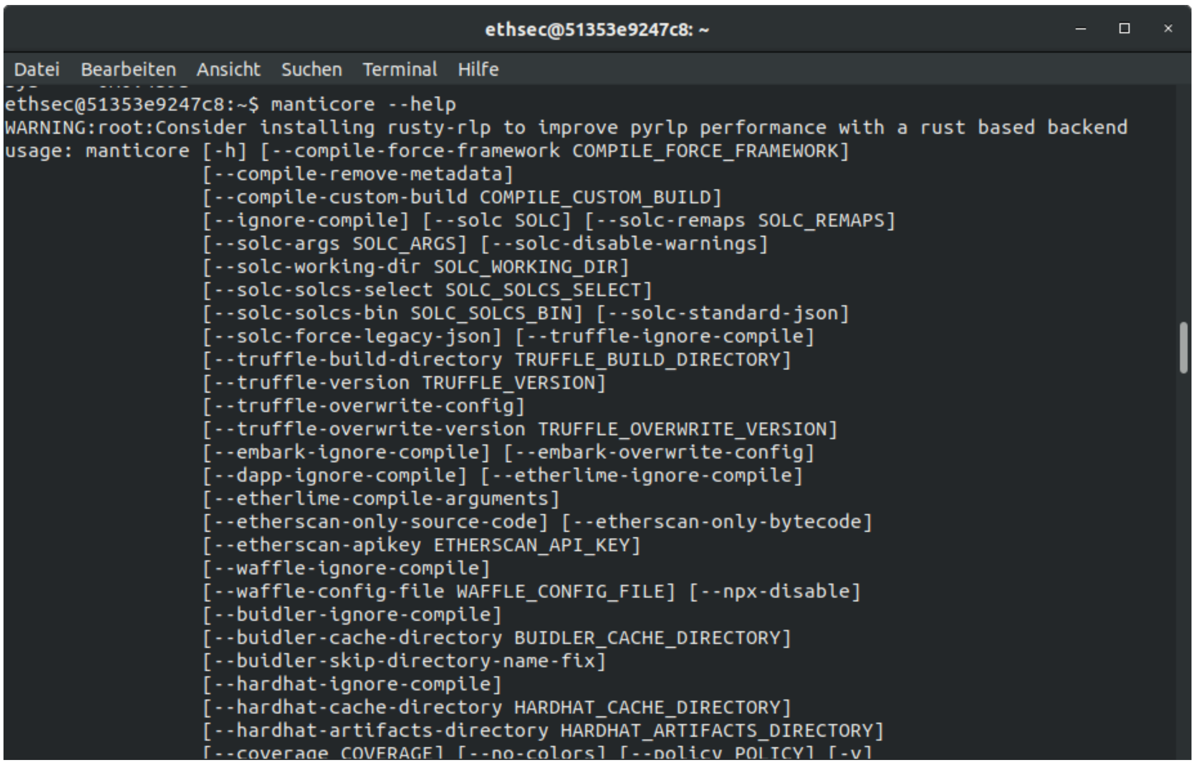
4.1.4. Manticore

Manticore[11] belongs to Trail of Bits[12], as does Slither (Chapter 4.1.2).

It is a CLI application to analyze smart contracts and binaries.

It is also compatible with the latest Solidity versions.

Manticore can be used as a Docker container or installed with the python package installer pip. [oBI20b]



```
ethsec@51353e9247c8:~$ manticore --help
WARNING:root:Consider installing rusty-rlp to improve pyrlp performance with a rust based backend
usage: manticore [-h] [--compile-force-framework COMPILE_FORCE_FRAMEWORK]
                 [--compile-remove-metadata]
                 [--compile-custom-build COMPILE_CUSTOM_BUILD]
                 [--ignore-compile] [--solc SOLC] [--solc-remaps SOLC_REMAPS]
                 [--solc-args SOLC_ARGS] [--solc-disable-warnings]
                 [--solc-working-dir SOLC_WORKING_DIR]
                 [--solc-solcs-select SOLC_SOLCS_SELECT]
                 [--solc-solcs-bin SOLC_SOLCS_BIN] [--solc-standard-json]
                 [--solc-force-legacy-json] [--truffle-ignore-compile]
                 [--truffle-build-directory TRUFFLE_BUILD_DIRECTORY]
                 [--truffle-version TRUFFLE_VERSION]
                 [--truffle-overwrite-config]
                 [--truffle-overwrite-version TRUFFLE_OVERWRITE_VERSION]
                 [--embark-ignore-compile] [--embark-overwrite-config]
                 [--dapp-ignore-compile] [--etherlime-ignore-compile]
                 [--etherlime-compile-arguments]
                 [--etherscan-only-source-code] [--etherscan-only-bytecode]
                 [--etherscan-apikey ETHERSCAN_API_KEY]
                 [--waffle-ignore-compile]
                 [--waffle-config-file WAFFLE_CONFIG_FILE] [--npx-disable]
                 [--buidler-ignore-compile]
                 [--buidler-cache-directory BUIDLER_CACHE_DIRECTORY]
                 [--buidler-skip-directory-name-fix]
                 [--hardhat-ignore-compile]
                 [--hardhat-cache-directory HARDHAT_CACHE_DIRECTORY]
                 [--hardhat-artifacts-directory HARDHAT_ARTIFACTS_DIRECTORY]
                 [--coverage COVERAGE] [--no-colors] [--policy POLICY] [-v]
```

Figure 4.4.: Screenshot: Manticore

[11]Manticore Hithub Repository: https://github.com/trailofbits/manticore
[12]Trail of Bits Website: https://www.trailofbits.com/

4.2. Services

In addition to the previously mentioned tools, this chapter also presents two vulnerability detection services for Solidity smart contracts.
In contrast to the tools, these services do not require any software to be installed and can used through a web browser.

4.2.1. Remix IDE

Remix[13] is an open source web based and desktop IDE.

The Remix Project consists of a seven member team working globally remotely on the Remix IDE and are funded by the Ethereum Foundation. [Pro20]

The Remix IDE offers a text editor with syntax-highlighting, a debugger, and a solidity compiler in its main features. It has several plugins like one that allows deploying a smart contract on a virtual or public blockchain, or even a static code analyzer, which analyzes the code during compilation. In addition, Remix has many other plugins that can be used to access services such as sourcify[14] or MythX[15] and much more.

Remix covers all compiler versions from 0.1.1 to the current version 0.7.5, and it does not require an installation or registration.

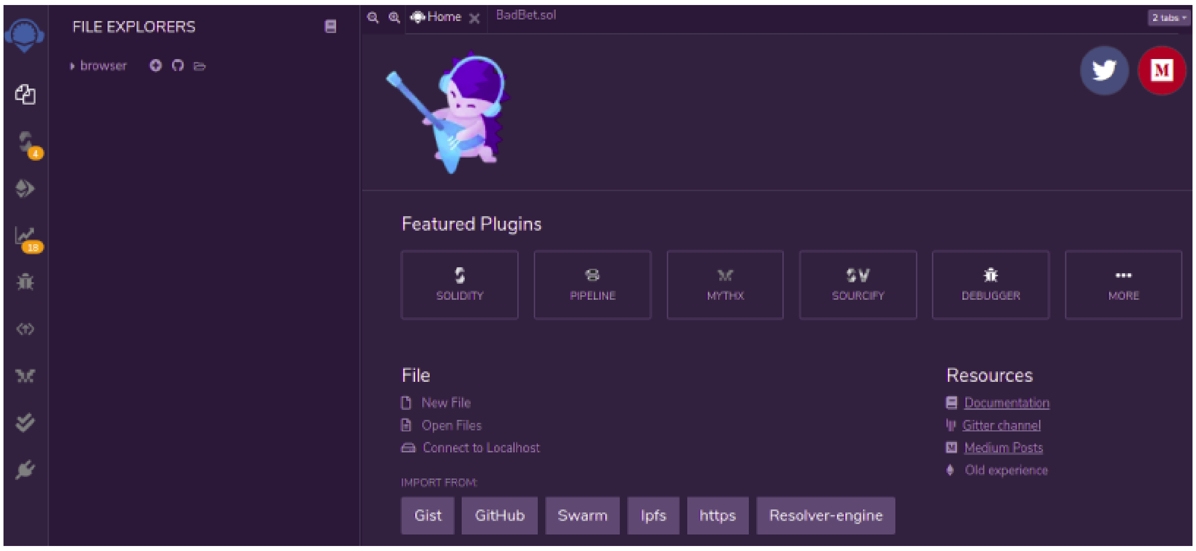


Figure 4.5.: Screenshot: Remix IDE

[13]Remix Website: https://remix.ethereum.org/
[14]Sourcify Website: http://sourcify.dev/
[15]MythX Website: https://mythx.io/

4.2.2. MythX

MythX[16] belongs the same way as Mythril (Chapter 4.1.1), to ConsenSys Diligence.

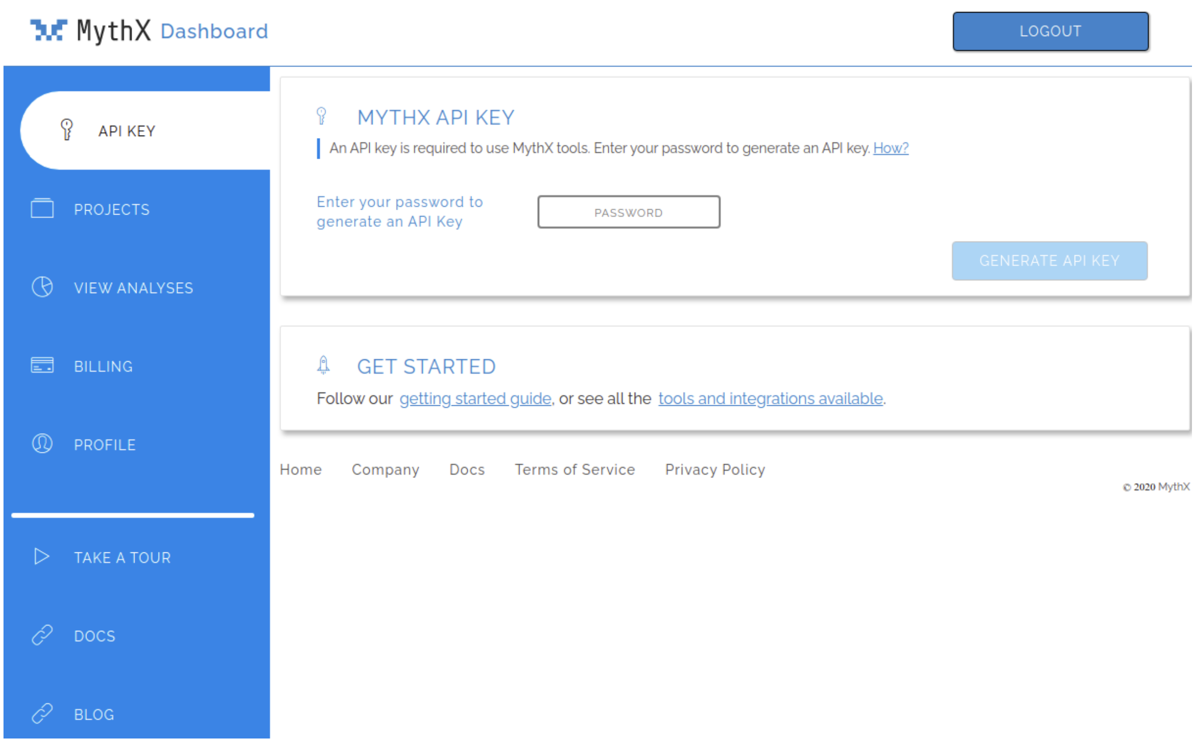
The project started in early 2018 with two developers and grew to 18 developers within the next two years.

The idea behind MythX was to create an API that takes a smart contract and returns a security report. Access to this report is provided by the MythX Dashboard.

Unlike the other analyzers, MythX is a paid vulnerability detection service that uses, among others, Mythril for vulnerability scanning. [Mue19]

In addition to a CLI, it is available as an extension for Visual Studio Code, Remix (Chapter 4.2.1) and other services. [Dil20b]

For this thesis, MythX provided three free scans to try their service.



MYTHX API KEY

An API key is required to use MythX tools. Enter your password to generate an API key. How?

Enter your password to generate an API Key

GET STARTED

Follow our getting started guide, or see all the tools and integrations available.

Figure 4.6.: Screenshot: MythX

[16]MythX website: https://mythx.io

5. Vulnerable Smart Contract: BadBet

In order to test the vulnerabilities mentioned in chapter 3, a smart contract was developed which contains all these gaps. The entire smart contract can also be found in the appendix A.1.

The smart contract was designed as a game. It is comparable to a coin flip and named 'BadBet'.

It is noteworthy that the program logic is not fully thought out and is prone to errors overall. However, the creation of a playable game was not the goal of this thesis. The design of the smart contract as a game simply demonstrates the processes of how an user would possibly interact with the contract.

Therefore, this chapter will provide an overview of the smart contract gameplay and logic, as well as the built-in security vulnerabilities.

5.1. Gameplay

The goal of the BadBet game is to win funds via a coinflip-like event.
In order to win, the player has several options to interact with the smart contract.

- To participate in the game, the player has to deposit funds first by using the `'deposit()'` function.
- To start the game, the `'bet()'` function is called with an amount of the deposited funds.
- The `'bet()'` function calls two random functions alternately.
- If the called random function returns true, the player has won and gets back 1.5 times the amount he bet.
- If false is returned, the player loses the funds to the smart contract.
- The bet is over and can be started again.

5.2. BadBet Smart Contract

The following contract `'BadBet'` was written in Solidity version 0.7.4.

Firstly, the needed variables are initialized. The variable `'boolean'` is used as a logical value to switch the random functions. The address `'owner'` is for the deployer of the contract. In the `'userlist'` the addresses of the players are stored.

To switch between the random functions, an external smart contract is used. Its address is stored in `'toggleContract'`. The `'balance'` mapping stores the stakes of all players.

The logical value `'winner'` is true if the player won or false if the player lost.

For the random function from chapter 3.6.1 the variables `'seed'` and `'iteration'` are used.

In the constructor, the variables are initialized to a default value or are assigned a value passed as an argument.

```
pragma solidity 0.7.4;

contract BadBet {
   bool public boolean;
   address payable public owner;
   address payable[] public userlist;
   address public toggleContract;
   mapping(address => uint) public balance;
   bool public winner;
   uint256 private seed;
   uint256 public iteration;

   constructor(uint256 _seed, address _toggleContract) {
       owner = msg.sender;
       toggleContract = _toggleContract;
       boolean = true;
       winner = false;
       seed = _seed;
       iteration = 0;
   }
```

Code 5.1: Contract initialization

Modifiers are used to modify the behavior of a function. They are used in the function header and are executed before entering the actual function. In this case, the two modifiers in code 5.2 are used for logical checks. Only if they are positive, the actual functions can be called.

The modifier `'isWinner()'` checks if a player has won the game and `'isOwner()'` checks if a user is the deployer and therefore the owner of the smart contract.

```
    modifier isWinner() {
        require(winner);
        _;
    }

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

Code 5.2: Modifiers

In order to participate in the game, the user has to call the `'deposit()'` function (Code 5.3) and deposit funds. The desired funds will be attached to the transaction and are accessible as `'msg.value'`.

By calling `'addUserToList()'`, the address of the user (`'msg.sender'`) will be added to the list of all players.

Afterward, the deposited funds will be assigned to the user address in the `'balance'` mapping.

```
    // deposit funds for bets
    function deposit() public payable {
        addUserToUserlist(msg.sender);
        balance[msg.sender] += msg.value;
    }
```

Code 5.3: Deposit Funds

The `'bet()'` function (Code 5.4) is called to start the game. To do this, the desired amount of funds to bet is passed as a parameter.

In the marked lines, an integer overflow or underflow (Chapter 3.5) is possible.

The underflow allows the player to gamble a larger `'_amount'` than its available `'balance'` and the overflow could make a bet smaller than intended. If the player has deposited 1 Ether and tries to bet 2 Ether, the sum will be less than 0. Thus the number will be positive, and the condition will be true.

If the condition in line 7 is true, the `'badRandomness()'` function is called to check randomly if the player has won. If false is returned, the player has lost.

Otherwise, the player has won, is marked as a `'winner'`, and `'withdraw()'` is called to pay out the winnings. Afterward, the betted `'amount'` is subtracted from his deposited `'balance'`.

```
// if user has enough balance for bet
// withdraw bet amount +50\% and substract bet amount from user
   balance
// Vulnerability:
// 3.5 Integer Over- and Underflow
function bet(uint _amount) public {
    uint256 amount = 1000000000000000000*_amount;  // wei to eth,
                                                     Integer Overflow
    if(balance[msg.sender]-amount > 0){   // Integer Underflow
        winner = badRandomness();
    }

    // require(winner, "no winner");
    if(winner) {
        withdraw(amount+(amount/2));
        winner = false;
    }
    balance[msg.sender]-=amount;
}
```

Code 5.4: bet function

The `'withdraw()'` function in code 5.5 can only be called if the modifier `'isWinner()'` is true. It checks if the contract has enough `'balance'` and then withdraws the winnings directly to the receiver using an unchecked call.

At this point, a reentrancy attack can occur if the recipient is a smart contract:

If the payout calls the malicious `'fallback()'` function of this smart contract, it is possible to re-enter the `'withdraw()'` function while `'isWinner()'` is true and the contract has enough `'balance'`.

Only after the transaction is finished are these two values changed. But the transaction is not complete until the code in the `fallback()` function has been fully executed.

```
    // if user is winner, withdraw winner amount
    // Vulnerabilities:
    // 3.1 External Calls - 3.1.1 Unchecked Call Return Value
    // 3.3 Reentrancy
    function withdraw(uint amount) public isWinner payable {
                  // isWinner: Reentrancy
        require(address(this).balance > amount);
        msg.sender.call{value: amount}("");                  //
            Unchecked Call Return Value, Reentrancy
    }
```

Code 5.5: Withdraw the winning

The `'badRandomness()'` function in code 5.6 is implemented to switch between the two random functions from chapter 3.6. If the return value `'b'` of a random function is true, the player has won, and if it is false, the player has lost.

At the end, `'callToggleBoolean()'` is called, which calls an external contract to toggle `'boolean'`. The external contract is only called to enable a delegatecall (Chapter 3.1.2).

```
    // uses two different random algortihmns
    // returns random boolean
    // true if win
    // false if lose
    function badRandomness() private returns(bool) {
        bool b = false;
        if(boolean) {
            b =  randomWithSeed();
        } else {
            b = randomWithBlockHash();
        }
        callToggleBoolean(toggleContract);
        return b;
    }
```

Code 5.6: Calling a random function

Code snippet 5.7 shows the implementation of the random function from chapter 3.6.1.

A `'randomNumber'` is generated using a private `'seed'` and an `'iteration'` number. The result of the modulo operation in line 7 returns the logical value.

```
1       // generates randomly true or false
2       // Vulnerability:
3       //  3.6 Bad Randomness -  3.6.1 With Private Seed
4       function randomWithSeed() private returns(bool) {
5           iteration++;
6           uint randomNumber = uint(keccak256(abi.encodePacked(seed+
                iteration)));
7           return ((randomNumber % 2) == 0);
8       }
```

Code 5.7: Randomness with Private Seed

The random function from chapter 3.6.2 is implemented in code 5.8.

Based on the previous `'block.number'`, a random number is generated using `'blockhash'`. According to the arithmetic operation, a logical value is returned.

```
    // generates randomly true or false
    // Vulnerability:
    //  3.6 Bad Randomness -  3.6.2 With Block Hash
    function randomWithBlockHash() private returns(bool) {
        return (uint(blockhash(block.number-1)) % 2 == 0);
    }
```

Code 5.8: Randomness with Block Hash

The '`addUserToUserlist()`' function ic code 5.9 is only used to add the player to a list, which will allow the contract to pay out the funds of all players at once.

It is the basis for the Denial of Service attacks from chapter 3.2.

```
    // if first deposit, add user to userlist
    function addUserToUserlist(address payable _user) private {
        if(balance[_user] == 0) {
            userlist.push(_user);
        }
    }
```

Code 5.9: Add User to userlist

In the code section 5.10 the '`boolean`' value for the random functions is toggled.

For this purpose, a '`delegatecall()`' to an external contract is executed in line 5. (Chapter 3.1.2) The address of the contract will be passed as a parameter.

Since the function is declared public, it can be executed by an attacker.

```
1       // toggles the toggle boolean. For Delegate Call Vulnerability
           only.
2       // Vulnerability:
3       // 3.1 External Calls - 3.1.2 Delegatecall to Untrusted Callee
4       function callToggleBoolean(address _toggleContract) public {
5           _toggleContract.delegatecall(abi.encodeWithSignature("
               toggleBoolean(bool)",boolean));
6       }
```

Code 5.10: Delegatecall to external smart contract

The following are administrative functions that are only used to implement some of previously described security vulnerabilites.

First the '`payOutAll()`' function in code 5.11. It is protected by the modifier '`isOwner`' and can only be executed by the owner.

At this point, the DoS attacks described in chapter 3.2 are possible. The '`while`'-loop will iterate through the '`userlist`' to pay out funds to each user. If a user from the list is a smart contract that does not accept payments, the transaction will be canceled. (3.2.1DoS with (Unexpected) revert)

The second possibility for a DoS is the block gas limit (mentioned in chapter 3.2.2). If the '`userlist`' becomes too long, the gas for this transaction will exceed the block gas limit, thus causing the transaction to fail.

In both cases no user will get paid and the entire function becomes unusable.

```
    /*
     *
     * administrative functions
     *
     */

    // emergency payout for all depositors (only owner)
    // if depositor is in userlist, send deposited balance to user
       address
    // Vulnerabilities:
    // 3.2 Denial of Service - 3.2.1 DoS With Unexpected Revert
    //                       - 3.2.2 DoS With Block Gas Limit
    function payOutAll() public isOwner {
        uint256 i = 0;
        while (i < userlist.length  && gasleft() > 200000) {
            require(userlist[i].send(balance[userlist[i]]-20000));
        }
    }
```

Code 5.11: Pay out all users

The following three functions cover the access control vulnerabilities mentioned in chapter 3.4.

With the first function `'setOwner()'` in code 5.12, which is declared as public, any user can become the `'owner'` of the smart contract.

```
    // set new owner
    // Vulnerability:
    // 3.4 Access Control - 3.4.1 Unprotected Owner Setter
    function setNewOwner(address payable _newOwner) public {
        owner = _newOwner;
    }
```

Code 5.12: Unprotected Owner Setter

The second function `'getContractBalance()'` in code snippet 5.13 allows an attacker to send the entire balance of the smart contract to the owner.
If this function is called, all the players' funds are gone and they can not be paid out anymore. This would at least significantly damage the gameplay and in the worst rob the player of their funds.

```
    // transfer the contract balance to msg.sender
    // Vulnerability:
    // 3.4 Access Control - 3.4.2 Unprotected Ether Withdrawl
    function getContractBalance() public {
        owner.transfer(address(this).balance);
    }
```

Code 5.13: Unprotected Ether Withdrawl

Using the function `'destroyContract()'` in code 5.14 makes it possible for anyone to destroy this smart contract. The balance of the contract will be sent to the `'owner'`. After calling this function, the smart contract is disabled forever.

```
    // destroys the smart contract
    // sends the balance to the msg.sender
    // Vulnerability:
    // 3.4 Access Control - 3.4.3 Unprotected Selfdestruct
    function destroyContract() public {
        selfdestruct(owner);
    }
}
```

Code 5.14: Unprotected Selfdestruct

The following smart contract `'Toggle'` in code 5.15 has been created to enable the `'delegatecall()'` and is used only to enable the corresponding vulnerability from chapter 3.1.2.
Its only function is to toggle the `'boolean'` variable.

```
    // Toggle contract, only to use delegatecall
    // toggles boolean input
    contract Toggle {
        bool public boolean;

        constructor() {
            boolean = false;
        }

        function toggleBoolean(bool _b) public {
            boolean = !_b;
        }
    }
```

Code 5.15: Toggle Smart Contract

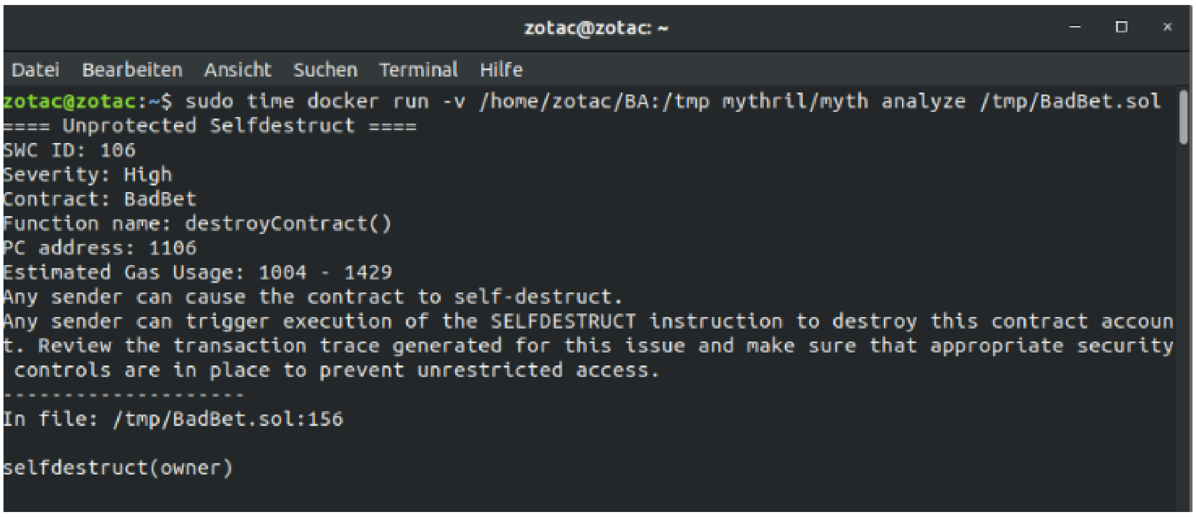
6. Results of the BadBet Smart Contract Analysis

Each of the tools and services listed in chapter 4 has analyzed the vulnerable BadBet contract. This chapter provides an overview of all the test results of the analysis.
Some names of the results have been aligned to make it easier to compare the results afterward. When it was possible, the duration was measured with the command `time`.

By using the Docker container, the BadBet smart contract was located in the `/home/zotac/BA` folder and mounted into the `/tmp` folder of the container.

6.1. Mythril

Mythril was obtained as an image from the Docker hub. Version v0.22.14 of Mythril was used. It is compatible with the current Solidity version.
According to the instructions, the analysis was started by using the following command:
`docker run -v /home/zotac/BA:/tmp mythril/myth analyze /tmp/BadBet.sol`
The analysis finished in about 25 minutes and returned a total of 4 results, classified as High or Medium vulnerable.



```
zotac@zotac:~$ sudo time docker run -v /home/zotac/BA:/tmp mythril/myth analyze /tmp/BadBet.sol
==== Unprotected Selfdestruct ====
SWC ID: 106
Severity: High
Contract: BadBet
Function name: destroyContract()
PC address: 1106
Estimated Gas Usage: 1004 - 1429
Any sender can cause the contract to self-destruct.
Any sender can trigger execution of the SELFDESTRUCT instruction to destroy this contract accoun
t. Review the transaction trace generated for this issue and make sure that appropriate security
 controls are in place to prevent unrestricted access.
--------------------
In file: /tmp/BadBet.sol:156

selfdestruct(owner)
```

Figure 6.1.: Screenshot: Mythril Output

6.1.1. Analysis Results

As seen in table 6.1, the analysis found 3 of the 12 intentionally implemented vulnerabilities of the BadBet contract. Mythril detected the 'Delegate Call to Untrusted Callee' (3.1.2) in line 116, the 'Unprotected Selfdestruct' (3.4.3) in line 116 and the 'Integer Overflow' (3.5.1) in line 47.

Vulnerability	**Line(s)**	**Found**
External Calls		
Unchecked Call Return Value	68	
Delegate Call to Untrusted Callee	116	x
Failed Call Using Send (See: DoS with Unexpected Revert)		
DoS		
With Unexpected Revert	133	
With Block Gas Limit	130-133	
Reentrancy		
Reentrancy	66-68	
Access Control		
Unprotected Owner Setter	140, 141	
Unprotected Ether Withdrawl	147, 148	
Unprotected Selfdestruct	155, 156	x
Integer Over- and Underflow		
Integer Overflow	48	x
Integer Underflow	49	
Bad Randomness		
With Private Seed	92	
With Block Hash	100	

Table 6.1.: Overview of Intentionally Implemented Vulnerabilities: Mythril

Additionally, an 'Unchecked Call Return Value' (3.1.1) was found. (Table 6.2)

Line	**Result**
116:	Medium: Unchecked Call Return Value

Table 6.2.: Further Results: Mythril

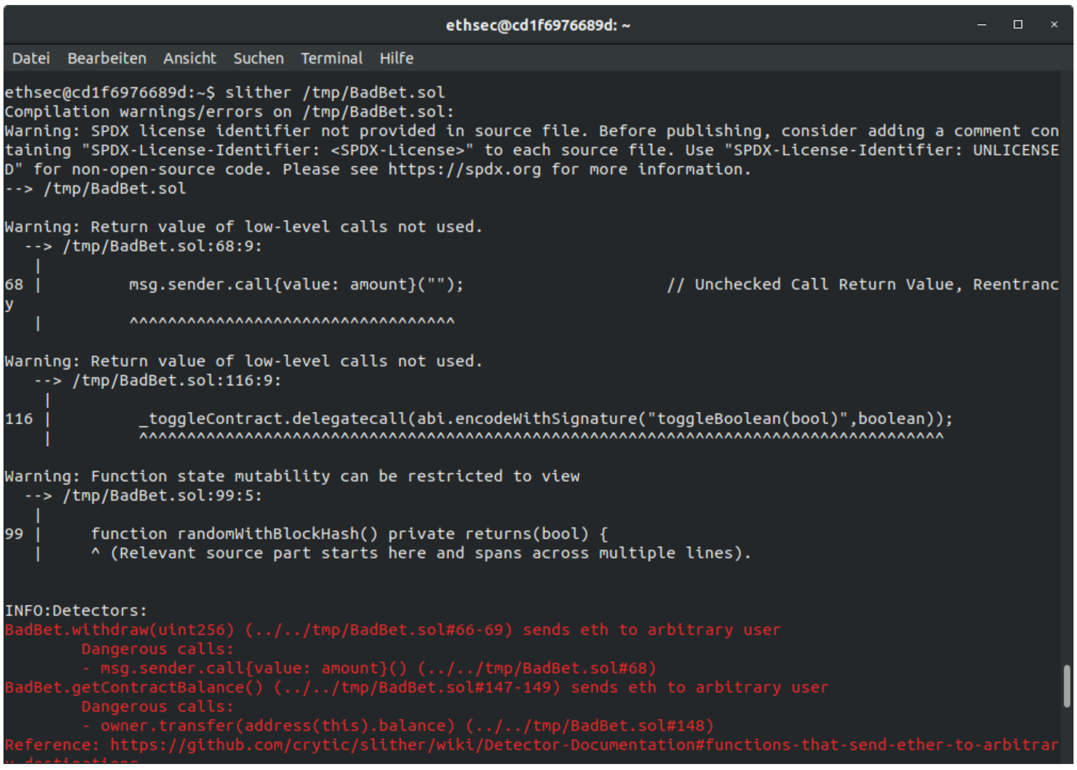
The complete analysis result can be seen in appendix A.3.

6.1.2. Conclusion

The analysis took a comparatively long time from start to finish. Mythril found only 3 of the 12 intentionally implemented vulnerabilities. As can be seen in figure 6.1, the output is very organized. It names and describes the vulnerability.
The severity of the vulnerabilities is shown through a classification in "High", "Medium" and "Low". For further information about the found vulnerability, the SWC ID is provided. Information about the state, the call data, gas usage, and PC address are also provided.

6.2. Slither

Slither was also run using the given Docker container. Version 0.6.14 of Slither was used for analysis. It is compatible with the current Solidity version. Slither uses by default 72 detectors for the vulnerability analysis, which can also be used separately as an option.
The analysis was started by the command: `slither /tmp/BadBet.sol`
The code scan was finished within 0.254 seconds and returned 27 results. The results were classified by the colors white, red, yellow, and green.



```
ethsec@cd1f6976689d:~$ slither /tmp/BadBet.sol
Compilation warnings/errors on /tmp/BadBet.sol:
Warning: SPDX license identifier not provided in source file. Before publishing, consider adding a comment con
taining "SPDX-License-Identifier: <SPDX-License>" to each source file. Use "SPDX-License-Identifier: UNLICENSE
D" for non-open-source code. Please see https://spdx.org for more information.
--> /tmp/BadBet.sol

Warning: Return value of low-level calls not used.
  --> /tmp/BadBet.sol:68:9:
   |
68 |        msg.sender.call{value: amount}("");                   // Unchecked Call Return Value, Reentranc
y
   |        ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Warning: Return value of low-level calls not used.
   --> /tmp/BadBet.sol:116:9:
    |
116 |        _toggleContract.delegatecall(abi.encodeWithSignature("toggleBoolean(bool)",boolean));
    |        ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Warning: Function state mutability can be restricted to view
  --> /tmp/BadBet.sol:99:5:
   |
99 |    function randomWithBlockHash() private returns(bool) {
   |    ^ (Relevant source part starts here and spans across multiple lines).


INFO:Detectors:
BadBet.withdraw(uint256) (../../tmp/BadBet.sol#66-69) sends eth to arbitrary user
        Dangerous calls:
        - msg.sender.call{value: amount}() (../../tmp/BadBet.sol#68)
BadBet.getContractBalance() (../../tmp/BadBet.sol#147-149) sends eth to arbitrary user
        Dangerous calls:
        - owner.transfer(address(this).balance) (../../tmp/BadBet.sol#148)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrar
```

Figure 6.2.: Screenshot: Slither Output

6.2.1. Analysis Results

As can be seen in table 6.3, Slither found the 'Unchecked Call Return Value' (3.1.1) in row 68, the 'Delegate Call to Untrusted Callee' (3.1.2) in row 116, the 'DoS with unexpected Revert' (3.2.1) in row 133, the 'Unprotected Ether Withdrawl' (3.4.2) in row 148 and the 'Unprotected Selfdestruct' (3.4.3) in row 156.

Vulnerability	**Line(s)**	**Found**
External Calls		
Unchecked Call Return Value	68	x
Delegate Call to Untrusted Callee	116	x
Failed Call Using Send (See: DoS with Unexpected Revert)		
DoS		
With Unexpected Revert	133	x
With Block Gas Limit	130-133	
Reentrancy		
Reentrancy	66-68	
Access Control		
Unprotected Owner Setter	140, 141	
Unprotected Ether Withdrawl	147, 148	x
Unprotected Selfdestruct	155, 156	x
Integer Over- and Underflow		
Integer Overflow	48	
Integer Underflow	49	
Bad Randomness		
With Private Seed	92	
With Block Hash	100	

Table 6.3.: Overview of Intentionally Implemented Vulnerabilities: Slither

In addition to the results from table 6.3, Slither found 22 other results. As seen in table 6.4, these include a 'SPDX license identifier not provided' warning in line 0, 'Unchecked Call Return Values' in lines 68 and 116, 'Function visibility' reports in lines 37, 47, 99, 130, 140, 147, 155 and 170, a 'Possible to send eth to arbitrary user' warning in line 68, 'Possible Reentrancy Issue' in lines 47-59, 'Dangerous strict equality' in line 100, a 'Pragma version too recent' warning in line 5, 'Violation of solidity naming conventions' in lines 47, 104, 115, 140 and 170, and finally 'Literals with too many digits' in lines 48 and 132.

Line(s)	**Result**
(0:	*SPDX license identifier is not provided)*
116:	Warning: Unchecked Call Return Value
99:	Warning: Function visibility
68:	Possible to send eth to arbitrary user
47-59:	Possible Reentrancy Issue
100:	Dangerous strict equality
116:	Unchecked Call Return Value
5:	Pragma version is too recent
68:	Unchecked Call Return Value
47:	Violation of solidity naming conventions
104:	Violation of solidity naming conventions
115:	Violation of solidity naming conventions
140:	Violation of solidity naming conventions
170:	Violation of solidity naming conventions
48:	Literals with too many digits
132:	Literals with too many digits
37:	Function visibility
47:	Function visibility
130:	Function visibility
140:	Function visibility
147:	Function visibility
155:	Function visibility
170:	Function visibility

Table 6.4.: Further Reults: Slither

Note: Slither grouped the 'Possible Reentrancy Issues' in lines 47-59 to a single result. They refer in detail to lines 50, 55, 56, and 58. In order to compare the results later in this thesis, they are treated as individual results. This increases the total number of results to 30.

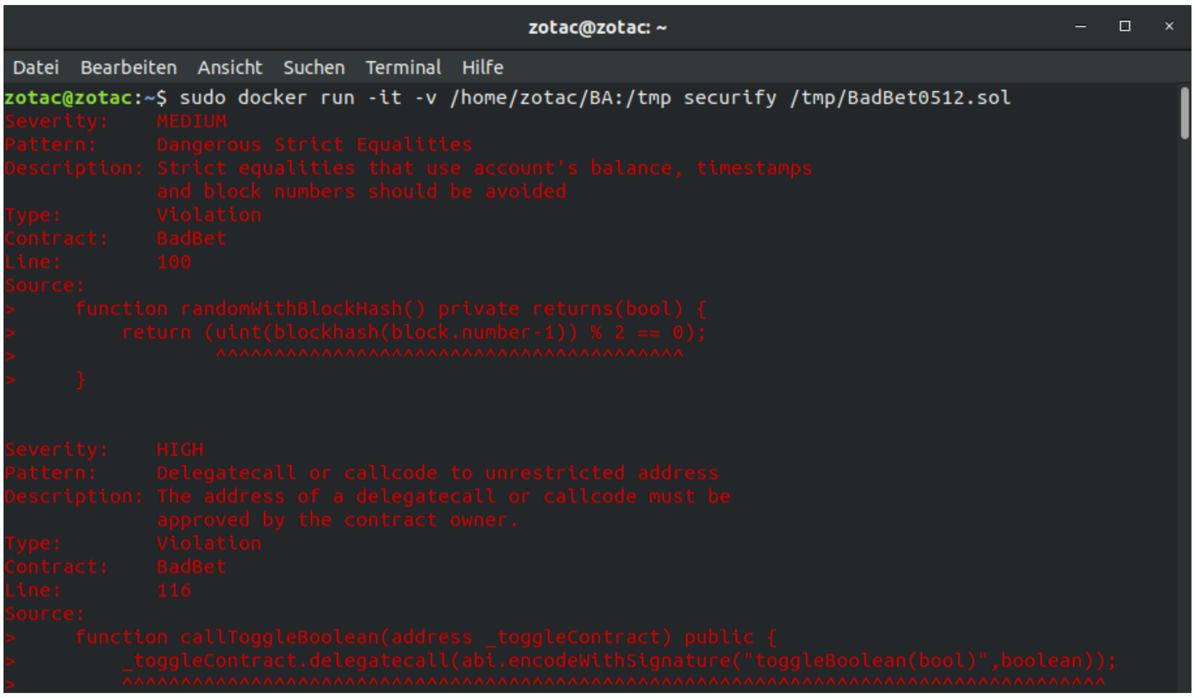
The complete analysis result can be seen in appendix A.4.

6.2.2. Conclusion

Compared to Myhtrill, the analysis of the smart contract through Slither is incredibly fast. The vulnerability detection is also more sophisticated, with 30 vulnerabilities found in total. As shown in figure 6.2, the output is less organized. Slither names the gap, points out the line in the source code, and provides a link for more information. In addition to critical security vulnerabilities, Slither also provides information on naming conventions and good practices.

6.3. Securify

Securify v2.0 was executed as a Docker container for the analysis. Securify does not support the current Solidity version. It uses the compiler version 0.5.12. Thus, a smart contract with Solidity version 0.5.12 was used. This contract does not differ in scope, function, type, and amount of vulnerabilities from the previously used smart contract and can therefore be used as a reference without any concerns. The entire smart contract can also be found in the appendix A.2. The analysis was started by the command:
`docker run -it -v /home/zotac/BA:/tmp securify /tmp/BadBet0512.sol`
The analysis ran for 22 seconds and resulted in 27 results, categorized in CRITICAL, HIGH, MEDIUM and INFO.



```
zotac@zotac:~$ sudo docker run -it -v /home/zotac/BA:/tmp securify /tmp/BadBet0512.sol
Severity:    MEDIUM
Pattern:     Dangerous Strict Equalities
Description: Strict equalities that use account's balance, timestamps
             and block numbers should be avoided
Type:        Violation
Contract:    BadBet
Line:        100
Source:
>     function randomWithBlockHash() private returns(bool) {
>         return (uint(blockhash(block.number-1)) % 2 == 0);
>                 ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
>     }


Severity:    HIGH
Pattern:     Delegatecall or callcode to unrestricted address
Description: The address of a delegatecall or callcode must be
             approved by the contract owner.
Type:        Violation
Contract:    BadBet
Line:        116
Source:
>     function callToggleBoolean(address _toggleContract) public {
>         _toggleContract.delegatecall(abi.encodeWithSignature("toggleBoolean(bool)",boolean));
>         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

Figure 6.3.: Screenshot: Securify Output

6.3.1. Analysis Results

Securify has found 5 of the intentionally implemented vulnerabilities. As can be seen in table 6.5, these are the 'Delegate Call to Untrusted Callee' (3.1.2) in line 116, the 'DoS with unexpected Revert' (3.2.1) in line 133, the 'Unprotected Owner Setter' (3.4.1) in line 141, the 'Unprotected Ether Withdrawl' (3.4.2) in line 148, and the 'Unprotected Selfdestruct' (3.4.3) in line 156.

Vulnerability	**Line(s)**	**Found**
External Calls		
Unchecked Call Return Value	68	
Delegate Call to Untrusted Callee	116	x
Failed Call Using Send (See: DoS with Unexpected Revert)		
DoS		
With Unexpected Revert	133	x
With Block Gas Limit	130-133	
Reentrancy		
Reentrancy	66-68	
Access Control		
Unprotected Owner Setter	140, 141	x
Unprotected Ether Withdrawl	147, 148	x
Unprotected Selfdestruct	155, 156	x
Integer Over- and Underflow		
Integer Overflow	48	
Integer Underflow	49	
Bad Randomness		
With Private Seed	92	
With Block Hash	100	

Table 6.5.: Overview of Intentionally Implemented Vulnerabilities: Securify

As Table 6.6 shows, Securify detected 22 further problems.
The Information about an 'Unused State Variable' found in line 164 relates to the Toggle contract. The 21 others concern the BadBet contract.

The analysis found a 'Dangerous strict equality' in line 100, hints about 'Function visibility' in lines 37, 47, 130, 140, 147, 155 and 170, information that 'address.call should be avoided' in line 116, 'Missing Input Validations' in lines 47, 66, 115 and 170, information about 'Literals with too many digits' in lines 48 and 132, the critical messages 'Transaction Order Affects Ether Amount' in lines 133 and 148, an 'Unrestricted Ether Flow' in line 116, 'Unrestricted write to storage' in lines 106 and 171 and finally, an 'Unchecked return value from external call' in line 116.

Line	**Result**
100:	Medium: Dangerous strict equality
37:	Low: Function visibility
47:	Low: Function visibility
130:	Low: Function visibility
140:	Low: Function visibility
147:	Low: Function visibility
155:	Low: Function visibility
170:	Low: Function visibility
116:	Info: address.call should be avoided
47:	Medium: Missing Input Validation
66:	Medium: Missing Input Validation
115:	Medium: Missing Input Validation
170:	Medium: Missing Input Validation
48:	Info: Literals with too many digits
132:	Info: Literals with too many digits
133:	Critical: Transaction Order Affects Ether Amount
148:	Critical: Transaction Order Affects Ether Amount
116:	Critical: Unrestricted Ether Flow
106:	Critical: Unrestricted write to storage
171:	Critical: Unrestricted write to storage
116:	Medium: Unchecked Call Return Value
(164:	*Info: Unused State Variable in Toggle contract)*

Table 6.6.: Further Results: Securify

Note: Since the last result refers to the Toggle Contract and not to the BadBet Contract, it will not be considered any further in this paper. This reduces the total number of results to 26.

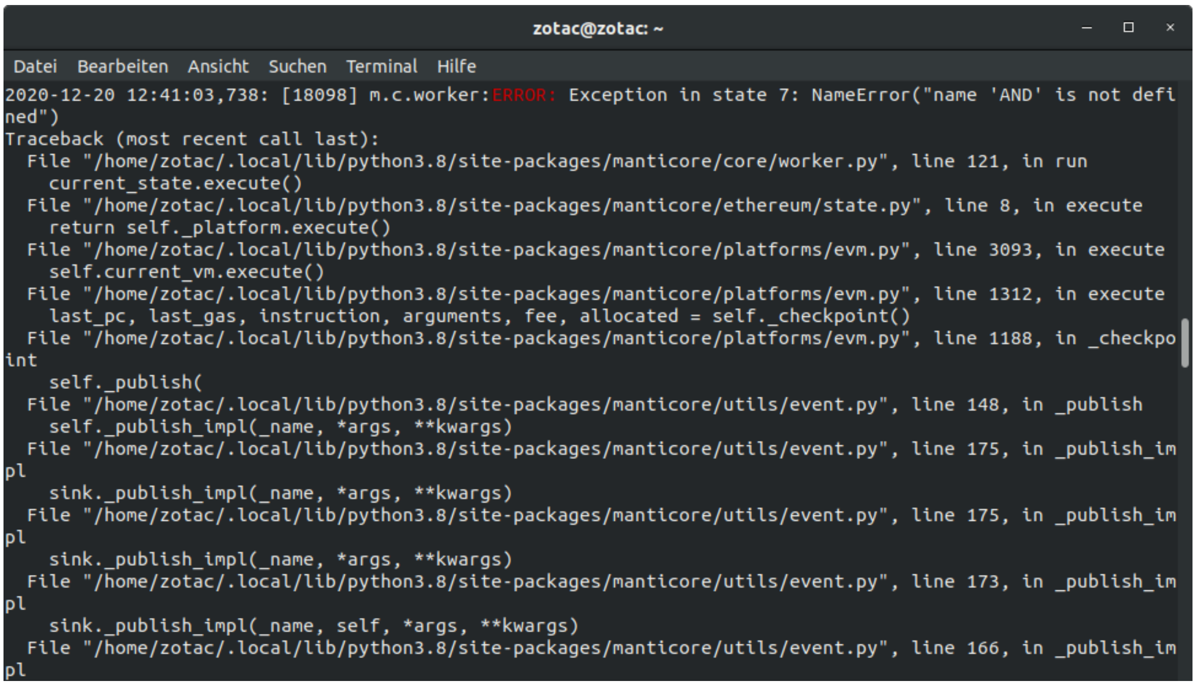
The complete analysis result can be seen in appendix A.5.

6.3.2. Conclusion

The analysis of Securify runs also very fast. Overall, the detection of 26 problems is rather good. However, the outdated compiler version is a significant disadvantage. Newer Solidity versions already close some security gaps by default and should also be supported by developers' tools. As seen in figure 6.3, the results are good structured. The problem is named, shown, briefly described, and a severity classification was made. Unfortunately, Securify does not provide any further information about the vulnerability.

6.4. Manticore

The usage of Manticore was problematic. Because of this, it was used as a Docker container, as well as manually installed. However, in both cases, a program crash occurred at the beginning of the analysis. As can be seen in figure 6.4, the `'worker.py'` script throws an exception.
Because Manticore could not be executed correctly, it is not included in this thesis's further discussion.

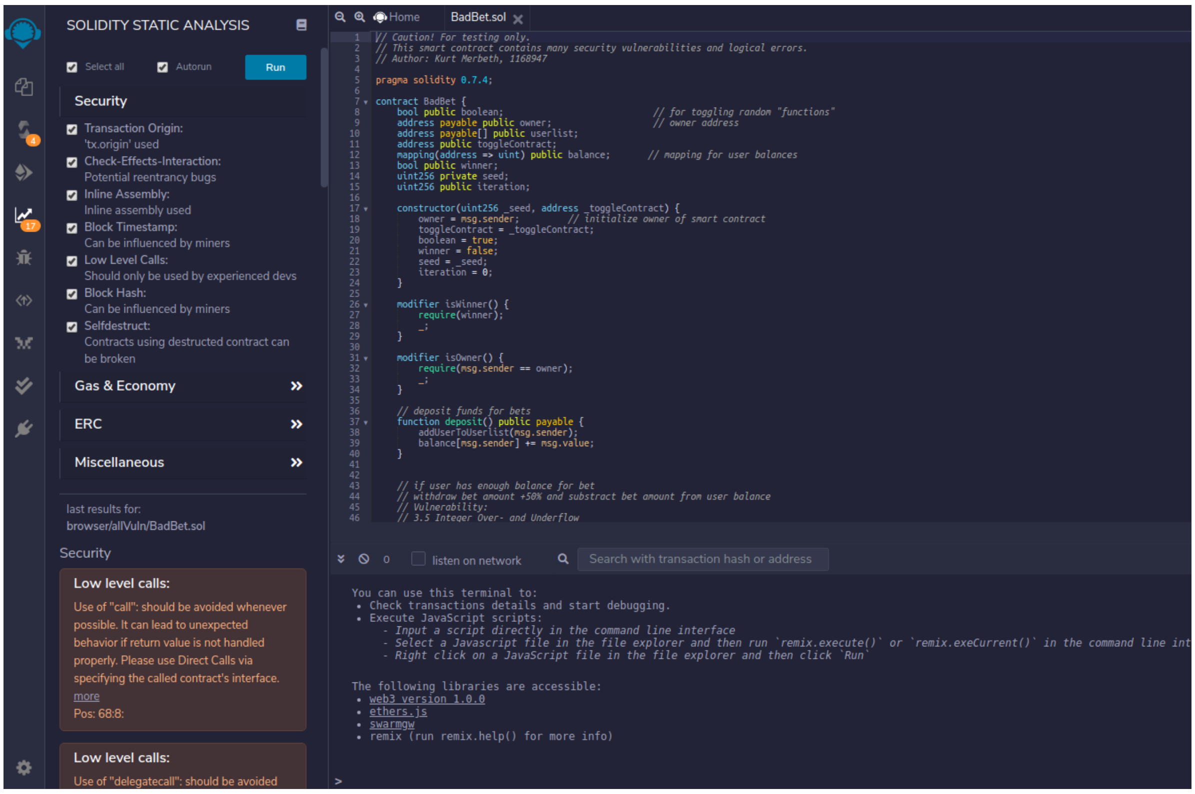


```
2020-12-20 12:41:03,738: [18098] m.c.worker:ERROR: Exception in state 7: NameError("name 'AND' is not defi
ned")
Traceback (most recent call last):
  File "/home/zotac/.local/lib/python3.8/site-packages/manticore/core/worker.py", line 121, in run
    current_state.execute()
  File "/home/zotac/.local/lib/python3.8/site-packages/manticore/ethereum/state.py", line 8, in execute
    return self._platform.execute()
  File "/home/zotac/.local/lib/python3.8/site-packages/manticore/platforms/evm.py", line 3093, in execute
    self.current_vm.execute()
  File "/home/zotac/.local/lib/python3.8/site-packages/manticore/platforms/evm.py", line 1312, in execute
    last_pc, last_gas, instruction, arguments, fee, allocated = self._checkpoint()
  File "/home/zotac/.local/lib/python3.8/site-packages/manticore/platforms/evm.py", line 1188, in _checkpo
int
    self._publish(
  File "/home/zotac/.local/lib/python3.8/site-packages/manticore/utils/event.py", line 148, in _publish
    self._publish_impl(_name, *args, **kwargs)
  File "/home/zotac/.local/lib/python3.8/site-packages/manticore/utils/event.py", line 175, in _publish_im
pl
    sink._publish_impl(_name, *args, **kwargs)
  File "/home/zotac/.local/lib/python3.8/site-packages/manticore/utils/event.py", line 175, in _publish_im
pl
    sink._publish_impl(_name, *args, **kwargs)
  File "/home/zotac/.local/lib/python3.8/site-packages/manticore/utils/event.py", line 173, in _publish_im
pl
    sink._publish_impl(_name, self, *args, **kwargs)
  File "/home/zotac/.local/lib/python3.8/site-packages/manticore/utils/event.py", line 166, in _publish_im
pl
```

Figure 6.4.: Screenshot: Manticore Exception

6.5. Remix

As mentioned in chapter 3, Remix was used as the IDE to develop the BadBet smart contract of chapter 5. The static code analysis is executed automatically during compilation and takes only 0.107 seconds. In this case, the duration was measured using the Chrome browser's built-in developer tools. All Solidity versions starting at version 0.1.1 are supported. Remix found a total of 17 problems, categorized as "Security", "Gas & Economy", "ERC", and "Miscellaneous".



Security

- ✓ Transaction Origin: 'tx.origin' used
- ✓ Check-Effects-Interaction: Potential reentrancy bugs
- ✓ Inline Assembly: Inline assembly used
- ✓ Block Timestamp: Can be influenced by miners
- ✓ Low Level Calls: Should only be used by experienced devs
- ✓ Block Hash: Can be influenced by miners
- ✓ Selfdestruct: Contracts using destructed contract can be broken

Gas & Economy »

ERC »

Miscellaneous »

last results for: browser/allVuln/BadBet.sol

Security

Low level calls:
Use of "call": should be avoided whenever possible. It can lead to unexpected behavior if return value is not handled properly. Please use Direct Calls via specifying the called contract's interface. more
Pos: 68:8:

Low level calls:
Use of "delegatecall": should be avoided

```
// Caution! For testing only.
// This smart contract contains many security vulnerabilities and logical errors.
// Author: Kurt Merbeth, 1168947

pragma solidity 0.7.4;

contract BadBet {
    bool public boolean;                              // for toggling random "functions"
    address payable public owner;                     // owner address
    address payable[] public userlist;
    address public toggleContract;
    mapping(address => uint) public balance;        // mapping for user balances
    bool public winner;
    uint256 private seed;
    uint256 public iteration;

    constructor(uint256 _seed, address _toggleContract) {
        owner = msg.sender;         // initialize owner of smart contract
        toggleContract = _toggleContract;
        boolean = true;
        winner = false;
        seed = _seed;
        iteration = 0;
    }

    modifier isWinner() {
        require(winner);
        _;
    }

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

    // deposit funds for bets
    function deposit() public payable {
        addUserToUserlist(msg.sender);
        balance[msg.sender] += msg.value;
    }


    // if user has enough balance for bet
    // withdraw bet amount +50% and substract bet amount from user balance
    // Vulnerability:
    // 3.5 Integer Over- and Underflow
```

```
You can use this terminal to:
 • Check transactions details and start debugging.
 • Execute JavaScript scripts:
    - Input a script directly in the command line interface
    - Select a Javascript file in the file explorer and then run `remix.execute()` or `remix.exeCurrent()` in the command line int
    - Right click on a JavaScript file in the file explorer and then click `Run`

The following libraries are accessible:
 • web3 version 1.0.0
 • ethers.js
 • swarmgw
 • remix (run remix.help() for more info)

>
```

Figure 6.5.: Screenshot: Remix Overview

6.5.1. Analysis Results

Remix found 6 of the 12 intended vulnerabilities.
As seen in table 6.7, the 'Unchecked Call Return Value' (3.1.1) in line 68, 'Delegate Call to Untrusted Callee' (3.1.2) in line 116, the 'DoS with unexpected Revert' (3.2.1) and 'DoS with Block Gas Limit' (3.2.2) in line 133, the 'Unprotected Selfdestruct' (3.4.3) in line 156 and the 'Bad Randomness with Block Hash' (3.6.2) in line 100 were detected.

Vulnerability	**Line(s)**	**Found**
External Calls		
Unchecked Call Return Value	68	x
Delegate Call to Untrusted Callee	116	x
Failed Call Using Send (See: DoS with Unexpected Revert)		
DoS		
With Unexpected Revert	133	x
With Block Gas Limit	130-133	x
Reentrancy		
Reentrancy	66-68	
Access Control		
Unprotected Owner Setter	140, 141	
Unprotected Ether Withdrawl	147, 148	
Unprotected Selfdestruct	155, 156	x
Integer Over- and Underflow		
Integer Overflow	48	
Integer Underflow	49	
Bad Randomness		
With Private Seed	92	
With Block Hash	100	x

Table 6.7.: Overview of Intentionally Implemented Vulnerabilities: Remix IDE

In addition, Remix has found 11 other results.
Under the category 'Gas & Economy', in lines 47, 66, 115, and 147 'Gas requirement of function is infinite' was noted.
Listed as 'Miscellaneous' were 'Potentially missing function modifier Constant/View/Pure' in lines 99 and 104, 'Use assert() and/or require()' in lines 27, 32, 67, and 133, and that the 'Result of an integer division is an integer' in line 55.

Line	**Result**
47:	Gas & Economy: Gas requirement of function is infinite
66:	Gas & Economy: Gas requirement of function is infinite
115:	Gas & Economy: Gas requirement of function is infinite
147:	Gas & Economy: Gas requirement of function is infinite
99:	Miscellaneous: Potentially missing function modifier Constant/View/Pure
104:	Miscellaneous: Potentially missing function modifier Constant/View/Pure
27:	Miscellaneous: Use assert() and/or require()
32:	Miscellaneous: Use assert() and/or require()
67:	Miscellaneous: Use assert() and/or require()
133:	Miscellaneous: Use assert() and/or require()
55:	Miscellaneous: Result of an integer division is an integer

Table 6.8.: Further Results: Remix IDE

The complete analysis result can be seen in appendix A.6.

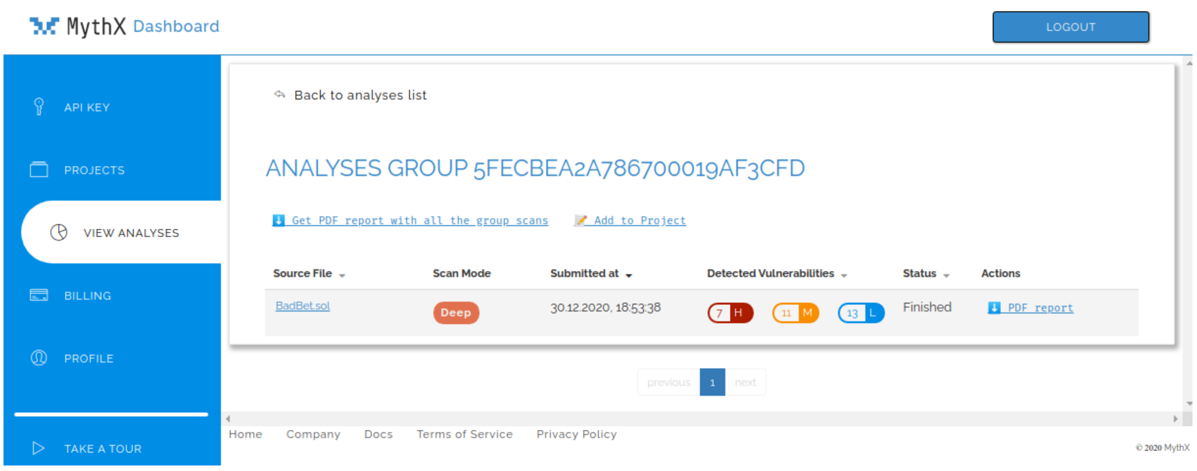
6.5.2. Conclusion

The static code analyzer from Remix is very easy to use, requires no installation, and can be used with all Solidity versions. The analysis was quick and yielded a total of 17 results. As can be seen in figure 4.5, the warnings are clearly presented in the left area of the window, and most of them provide further links. By clicking on the specific result, the related code section is highlighted in the editor window, which allows to view the erroneous part of the code quickly.

6.6. MythX

The Remix extension was used to access the MythX API. For this purpose, an API key has to been created in the MythX Dashboard and entered in the Remix IDE. Afterward, the analysis is started with one click. All results are available as PDF in the dashboard after finishing the analysis.

MythX's analysis took 45 minutes, which made it the slowest of all analyzers. It returned 31 vulnerabilities, categorized into "High", "Medium", and "Low".

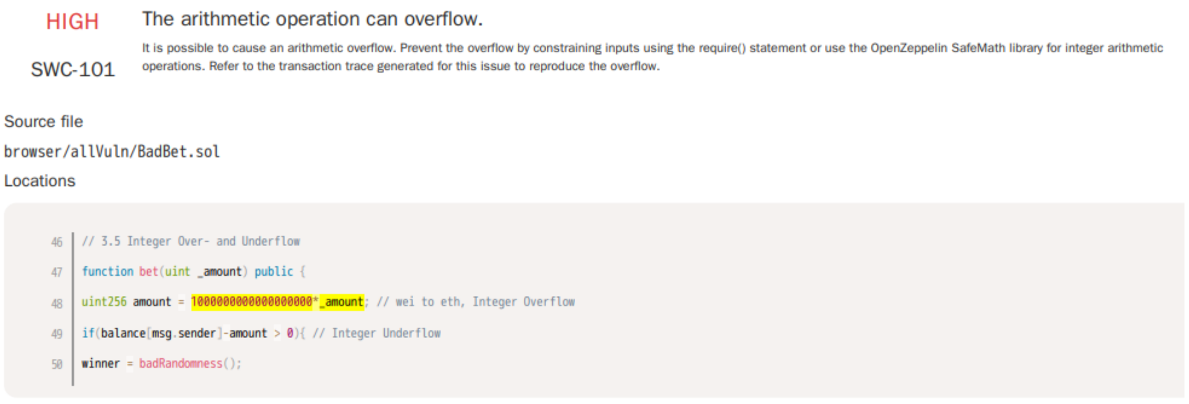


ANALYSES GROUP 5FECBEA2A786700019AF3CFD

Get PDF report with all the group scans   Add to Project

Source File	Scan Mode	Submitted at	Detected Vulnerabilities	Status	Actions
BadBet.sol	Deep	30.12.2020, 18:53:38	7 H 11 M 13 L	Finished	PDF report

Figure 6.6.: Screenshot: MythX Dashboard


HIGH

SWC-101

The arithmetic operation can overflow.

It is possible to cause an arithmetic overflow. Prevent the overflow by constraining inputs using the require() statement or use the OpenZeppelin SafeMath library for integer arithmetic operations. Refer to the transaction trace generated for this issue to reproduce the overflow.

Source file

browser/allVuln/BadBet.sol

Locations

```
// 3.5 Integer Over- and Underflow
function bet(uint _amount) public {
uint256 amount = 1000000000000000000*_amount; // wei to eth, Integer Overflow
if(balance[msg.sender]-amount > 0){ // Integer Underflow
winner = badRandomness();
```

Figure 6.7.: Screenshot: MythX Example Result

6.6.1. Analysis Results

As the table 6.9 shows, MythX found 8 of these entries. According to the table, the 'Unchecked Call Return Value' (3.1.1) in line 68, the 'Delegate Call to Untrusted' (3.1.2) in line 116, the 'DoS with unexpected Revert' (3.2.1) in line 133, the 'Reentrancy' (3.3) in line 67, the 'Unprotected Selfdestruct' (3.4.3) in line 156, the 'Integer Overflow' (3.5.1) in line 47, the 'Integer Underflow' (3.5.2) in line 48, and the 'Bad Randomness with Block Hash' (3.6.2) in line 100 were detected.

Vulnerability	**Line(s)**	**Found**
External Calls		
Unchecked Call Return Value	68	x
Delegate Call to Untrusted Callee	116	x
Failed Call Using Send (See: DoS with Unexpected Revert)		
DoS		
With Unexpected Revert	133	x
With Block Gas Limit	130-133	
Reentrancy		
Reentrancy	66-68	x
Access Control		
Unprotected Owner Setter	140, 141	
Unprotected Ether Withdrawl	147, 148	
Unprotected Selfdestruct	155, 156	x
Integer Over- and Underflow		
Integer Overflow	48	x
Integer Underflow	49	x
Bad Randomness		
With Private Seed	92	
With Block Hash	100	x

Table 6.9.: Overview of Intentionally Implemented Vulnerabilities: MythX

MythX has also found further results.
The 22 other results include two 'Arithmetic operator overflows' in lines 55 and 92, and two 'Arithmetic operator underflows' in lines 49 and 58, notes on 'Function visibility' in lines 37, 47, 130, 140, 147, 155, and 170, more 'Possible Reentrancy Issues' in lines 47, 50, 54, 56, and 58, two 'Requirement violations' in lines 148 and 176, two 'Calls with hardcoded gas amount' in lines 133 and 134, an 'Unchecked return value from external call' in line 116, and finally 'Multiple calls in the same transaction' in line 68.

Line	Result
92:	High: Arithmetic operator overflow
58:	High: Arithmetic operator underflow
55:	High: Arithmetic operator overflow
49:	High: Arithmetic operator underflow
37:	Medium: Function visibility
47:	Medium: Function visibility
130:	Medium: Function visibility
140:	Medium: Function visibility
147:	Medium: Function visibility
155:	Medium: Function visibility
170:	Medium: Function visibility
116:	Medium: Unchecked Call Return Value
68:	Medium: Multiple calls in the same transaction
50:	Low: Possible Reentrancy Issue
47:	Low: Possible Reentrancy Issue
58:	Low: Possible Reentrancy Issue
56:	Low: Possible Reentrancy Issue
54:	Low: Possible Reentrancy Issue
148:	Low: Requirement violation
176:	Low: Requirement violation
133:	Low: Call with hardcoded gas amount
134:	Low: Call with hardcoded gas amount

Table 6.10.: Further Results: MythX

Note: In total, only 30 results are listed in the 6.9 and 6.10 tables. According to the 'Bad Randomness with Block Hash' vulnerability from chapter 3.6.2 two results were listed in the report. One result for using the `'blockhash()'` *function and the other for using the* `'block.number'` *as a source for randomness.*
Since these two entries belong together, they were combined in the tables.

The complete analysis result can be seen in appendix A.7.

6.6.2. Conclusion

MythX is easy to use and can be used through an extension or through the MythX Cli. It supports the latest Solidity version and costs $9.99 for three scans in the smallest package. Thus, a code check at MythX also only costs $3.33, which is a manageable price for the security of a smart contract.

For the BadBet contract, the scan took about 45 minutes but provided a comprehensive result, which is accessible in the dashboard as a pdf. MythX found 8 of the 12 intentionally implemented vulnerabilities listed in table 6.9.

The report also provides much additional information on potential weaknesses that were not taken into account when the BadBet contract was created. In addition to the risk level classification, the report contains a reference to the SWC, a description of the problem and the exact location in the code.

7. Comparison of Results

In this chapter, the results of the different analyzers are compared.
The following aspects are taken into account for this comparison: Usage, Costs and Registration, the Solidity version used, the duration of the analysis, the content of the results, the quantity of the results, the intentionally implemented vulnerabilities found, and the further results.

7.1. Usage

Mythril, Slither, and Securify are CLI applications and can be used in the same way. On one hand, all three provide a Docker container. They are quick to downloaded, easy to use, and run on every machine that can run Docker container. On the other hand, all three tools can also be installed manually. Only a single command is needed to start the analysis.
In contrast, Remix IDE is a web application and can be accessed with any web browser. The smart contract must be written in the integrated editor or can be uploaded via the menu. After that, the analysis starts dynamically during compilation and the results can be viewed in the editor.
MythX uses a different approach: By providing an API that is accessible in different ways. It offers a CLI tool and various extensions such as Remix or Visual Studio Code. However, an API key is required to use each of those. The key is obtained after logging into the dashboard. With a few clicks, the analysis can be started in the extensions. The result is displayed on the MythX dashboard.

7.2. Costs and Registration

The tools Mythril, Slither, and Securify are offered for download free of charge. Also, the Remix IDE website is free to use. These four applications also do not require any registration.
Only MythX demands their users to register and pay in order to use their services. Single scans cost $9.99 for three analyses and additionally three different monthly subscriptions[1] are available.

[1]MythX Subscriptions: https://dashboard.mythx.io/#/console/subscription

7.3. Solidity Version

Mythril, Slither, Remix, and MythX are compatible with the latest Solidity version. Remix even offers compatibility with all versions since Solidity 0.1.1. Only Securify must be used with an older version. It is compatible with versions 0.5.8 - 0.5.12. Slither supports the current Solidity versions but advises to use an older, trusted version.

7.4. Duration of the Analysis

Remix analyzed the BadBet smart contract within the shortest timeframe. The scan required only 0.106 seconds. Slither took only slightly longer with 0.254 seconds. With 22 seconds, Securify took longer than Remix and Slither combined. Yet with under a minute runtime, the waiting period is manageable.
Mythril and MythX both took the longest for a complete analysis of the smart contract: Mythril took 25 minutes, MythX had the most extended runtime with 45 minutes.

7.5. Content of Results

According to table 7.1, all analyzers' results naming the security gap, contain a classification, the position in the source code, and the affected code location.
However, the given information differs between the tools in wording. In the case of Remix, it is necessary to click on the result to see the affected code location.

All analyzers, except for Slither, provide a short description of the detected problem.

Related sources are provided by Mythril, Slither, Remix, and MythX. Mythril and MythX provide the SWC ID of the weakness. Slither and Remix add a link for additional information to the search result.

Overall, Mythril provides the most comprehensive results. These include the PC Address, Gas Usage, Initial State, and a Transaction Sequence.

	Mythril	**Slither**	**Securify**	**Remix**	**MythX**
Naming of the Gap	x	x	x	x	x
Description	x		x	x	x
Classification	x	x	x	x	x
Position	x	x	x	x	x
Affected Code	x	x	x	x	x
SWC ID	x				x
Related Link		x		x	

Table 7.1.: Content of Results

7.6. Quantity

The chart 7.1 provides an overview of the number of analysis results.
In total, MythX was the most successful with 30 results.
MythX found 8 of the intentionally implemented vulnerabilities and 22 of the further results.
Slither has found a total of 30 problems. Of these, five are intentionally implemented vulnerabilities, and 25 belong to further results.
Similar to Slither, Securify found five of the intentionally implemented vulnerabilities, yet only 21 further results.
Remix has 17 results in total, which is significantly less than MythX, Slither, and Securify. Of these, six are from the intentionally implemented vulnerabilities and 11 further results.
It can be seen that Mythril performed the worst. With three intentionally implemented and one further result, Mythril found only four problems in total.

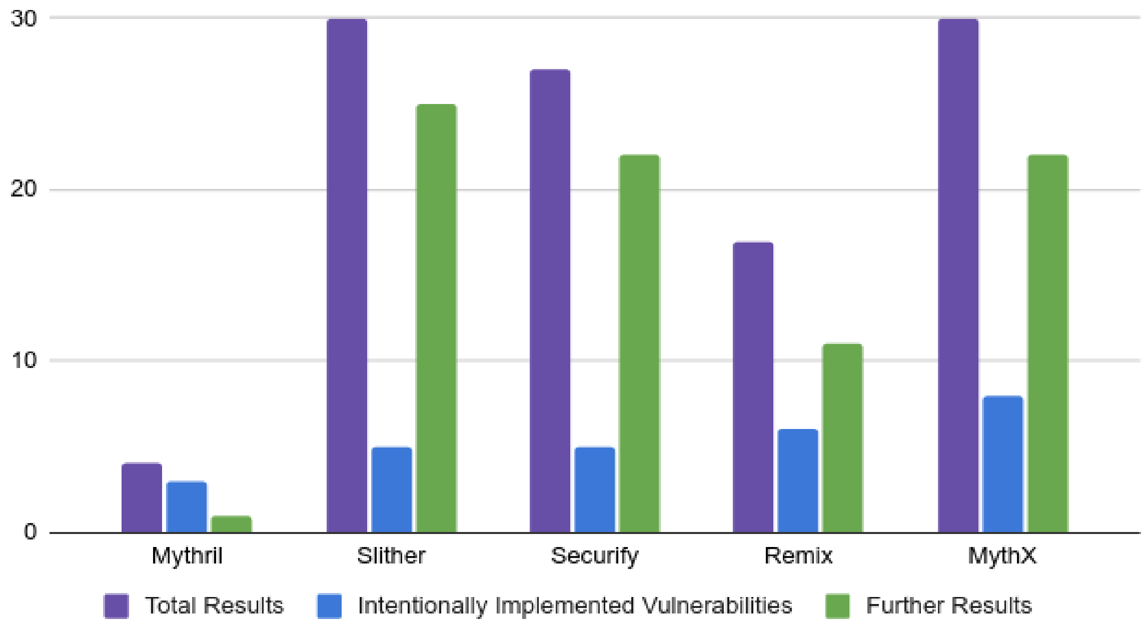


Figure 7.1.: Quantity of Results

The chart in figure 7.2 shows the ranking of the number of intentionally implemented vulnerabilities found by the analyzers in descending order. MythX found the most intentionally implemented vulnerabilities with eight results, followed by Remix with six results. Slither and Securify found each one vulnerability less than Remix with five results. With only three results, Mythril detected the least.

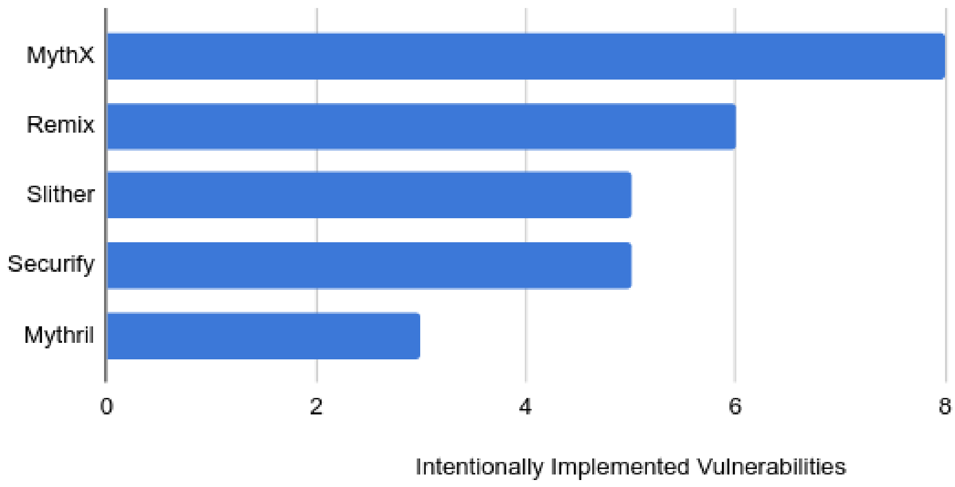


Figure 7.2.: Ranking: Intentionally Implemented Vulnerabilities

Figure 7.3 shows the ranking based on the further results in descending order. Slither found the most further results with 25 hits. MythX achieved 22 results, followed by Securify with 21 results. Remix found only 11 further problems. As before, Mythril found the lowest amount of results with only one hit.

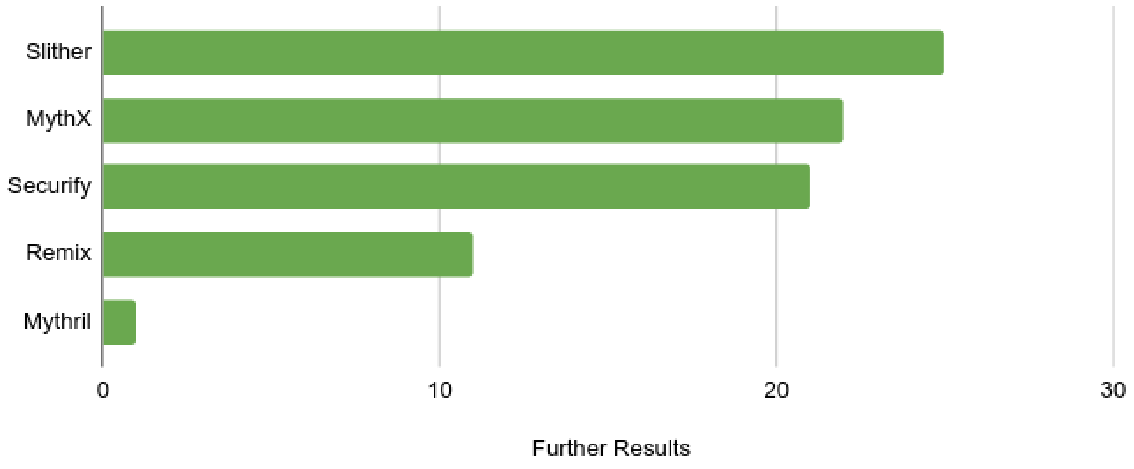


Figure 7.3.: Ranking: Further Results

A comparison of the total results is shown in the diagram in figure 7.4. As can be seen, MythX and Slither have found the most results with 30 each. Securify is in second place with 26 results. Remix is in third place with 17 results. With a significant gap, Mythril's analysis yielded a total of only four results.

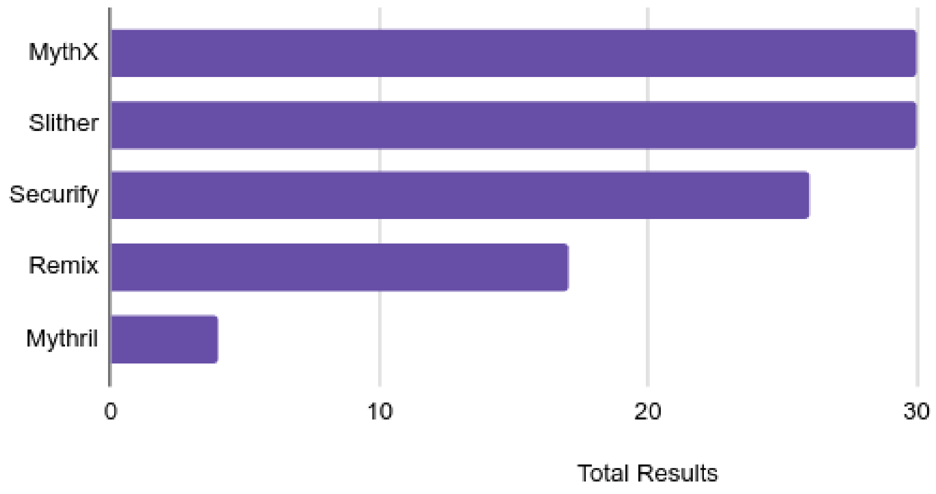


Figure 7.4.: Ranking: Total Results

7.7. Intentionally Implemented Vulnerabilities

The following table 7.2 shows a comparison of the analysis results of the intentionally implemented vulnerabilities, based on the BadBet smart contract. The previous chapters' results have been summarized and will not be discussed in detail.

The entry highlighted in red was not detected by any software.

The entries marked in gray were detected by exactly one analyzer.
Remix only detected the 'DoS With Block Gas Limit'. MythX only detected the 'Reentrancy' possibility and the 'Integer Overflow'. Furthermore, only Securify found the 'Unprotected Owner Setter'.

With the three applications just mentioned, all vulnerabilities found can be covered. Since Mythril is integrated with Mythx, its search results match those of MythX.
All results found by Slither were also discovered by one of the other analyzers.

Vulnerabilities	**Line(s)**	**Mythril**	**Slither**	**Securify**	**Remix**	**MythX**
External Calls						
Unchecked Call Return Value	68		x		x	x
Delegate Call to Untrusted Callee	116	x	x	x	x	x
Failed Call Using Send (see: DoS with Unexpected Revert)						
DoS						
With Unexpected Revert	133		x	x	x	x
With Block Gas Limit	130-133				x	
Reentrancy						
Reentrancy	66-68					x
Access Control						
Unprotected Owner Setter	140, 141			x		
Unprotected Ether Withdrawl	147, 148		x	x		
Unprotected Selfdestruct	155, 156	x	x	x	x	x
Integer Over- and Underflow						
Integer Overflow	48	x				x
Integer Underflow	49					x
Bad Randomness						
With Private Seed	92					
With Block Hash	100				x	x
Total:		**3**	**5**	**5**	**6**	**8**

Table 7.2.: Intentionally Implemented Vulnerabilities: All Results

7.8. Further Results

The tables 7.3 and 7.4 show all 54 further results sorted by the five analyzers and the results. Again, the gray highlighting indicates results that were only found by exactly one application.
As can be seen, this concerns most of the problems. Only 13 of the 54 entries were found more than once.

All analyzers found the ’Unchecked Call Return Value’ in line 116. Information about ’Function visibility’ was provided by Slither, Securify, and MythX. Messages about a ’Dangerous strict equality’ and ’Literals with too many digits’ were returned only by Slither and Securify. Slither and MythX found the ’Possible Reentrancy Issues’ in lines 50 and 58.

Slither only found the ’Possible Reentrancy Issues’ in lines 48 and 55. Warnings about that the ’SPDX license identifier is not Provided’, the ’Pragma version is too recent’, the ’Unchecked Call Return Value’ in line 68, and the ’Violation of solidity naming conventions’ were only provided by Slither.

Securify issued a general warning that ’Address.call should be avoided’. It also detected risks due to ’Missing Input Validations’ that the ’Transaction Order Affects Ether Amount’. It found an ’Unrestricted Ether Flow’ and detected an ’Unrestricted write to storage’ twice.

Remix detected ’Gas requirement of a function is infinite’ several times and ’Potentially missing function modifier’ twice. It also noticed that the ’Result of an integer division is an integer’ and pointed to ’Use assert() and/or require()’.

MythX, on the other hand, found ’Arithmetic operator overflows’ and ’Arithmetic operator underflows’. It also noted not using a ’Call with hardcoded gas amount’, found two other ’Possible Reentrancy Issues’ in lines 54 and 46, one ’Requirement violation’ and once pointed out ’Multiple calls in the same transaction’.

Line	Result	Mythril	Slither	Securify	Remix	MythX
116:	Unchecked Call Return Value	x	x	x		x
37:	Function visibility		x	x		x
47:	Function visibility		x	x		x
130:	Function visibility		x	x		x
140:	Function visibility		x	x		x
147:	Function visibility		x	x		x
155:	Function visibility		x	x		x
170:	Function visibility		x	x		x
100:	Dangerous strict equality		x	x		
48:	Literals with too many digits		x	x		
132:	Literals with too many digits		x	x		
50:	Possible Reentrancy Issue		x			x
58:	Possible Reentrancy Issue		x			x
48:	Possible Reentrancy Issue		x			
55:	Possible Reentrancy Issue		x			
5:	Pragma version is too recent		x			
0:	SPDX license identifier is not Provided		x			
68:	Unchecked Call Return Value		x			
47:	Violation of solidity naming conventions		x			
115:	Violation of solidity naming conventions		x			
140:	Violation of solidity naming conventions		x			
170:	Violation of solidity naming conventions		x			
116:	Address.call should be avoided			x		
47:	Missing Input Validation			x		
66:	Missing Input Validation			x		
115:	Missing Input Validation			x		
170:	Missing Input Validation			x		
133:	Transaction Order Affects Ether Amount			x		
148:	Transaction Order Affects Ether Amount			x		
116:	Unrestricted Ether Flow			x		
106:	Unrestricted write to storage			x		
171:	Unrestricted write to storage			x		

Table 7.3.: Further Results: All 54 Results, Part 1

Line	Result	Mythril	Slither	Securify	Remix	MythX
47:	Gas requirement of a function is infinite				x	
66:	Gas requirement of a function is infinite				x	
115:	Gas requirement of a function is infinite				x	
147:	Gas requirement of a function is infinite				x	
99:	Potentially missing function modifier Constant/View/Pure				x	
104:	Potentially missing function modifier Constant/View/Pure				x	
55:	Result of an integer division is an integer				x	
27:	Use assert() and/or require()				x	
32:	Use assert() and/or require()				x	
67:	Use assert() and/or require()				x	
133:	Use assert() and/or require()				x	
55:	Arithmetic operator overflow					x
58:	Arithmetic operator overflow					x
92:	Arithmetic operator overflow					x
49:	Arithmetic operator underflow					x
133:	Call with hardcoded gas amount					x
134:	Call with hardcoded gas amount					x
68:	Multiple calls in the same transaction					x
54:	Possible Reentrancy Issue					x
56:	Possible Reentrancy Issue					x
168:	Requirement violation					x
176:	Requirement violation					x

Table 7.4.: Further Results: All 54 Results, Part 2

8. Conclusion

In this bachelor thesis, six tools and services have been tested to detect security vulnerabilities in smart contracts on the Ethereum blockchain.

To maximize the readers understanding, first a solid foundation of knowledge has been established in chapter 2, regarding not only the basic mechanisms and ideas behind blockchains in general but also of the Ethereum blockchain in specific. This knowledge has been enriched by additional knowledge regarding smart contracts and their native programming language Solidity.

Following the general introduction of this thesis's topic, the most common security vulnerabilities were introduced and their developmental backgrounds illuminated in chapter 3: Each vulnerability has been explained in detail and in example. Grouped into the broad categories of External Calls, Denial of Service attacks, Reentrancy attacks, Access Control, Integer Overflow and Underflow vulnerabilities and problematic usage of Bad Randomness, the most common attack vectors have been covered.

Afterward, chapter 4 made a selection of vulnerability detection tools and services that were potentially able to detect the vulnerabilities covered in chapter 3. Each of these has been introduced briefly.

This chapter concluded the general coverage of foundational knowledge needed to understand the specific work done in this document.

Chapter 5 presents the smart contract that was designed and implemented specifically for the analysis conducted in this thesis. It explains the general gameplay, as well as the specific implementation of this contract and refers to the vulnerabilities from chapter 3: each vulnerability has been grafted into the smart contract code in order to allow for analyzation through the tools and services from chapter 4.

These tests and analyses were conducted in chapter 6. The results of each tool and service have been summarized in detail, as well as individual conclusions in comparison to each other tool have been drawn.

Most notably in this chapter is the vast difference in the runtime of each tools analysis. While Remix, Slither and Securify have analyzed the code within a few seconds, however clearly in under one minute, Mythril and MythX needed between 25 and 45 minutes to conduct their analysis.

Overall, all of the tools and services proved to be easy to use and delivered well-structured results. An exception was Manticore, which could not conduct an analysis because the software did not run properly.

Positive highlights, however, are Mythril, Slither, Remix and MythX, since they provided more information about the problems found.

The complete comparison of all gathered results was made in chapter 7. Here, specific aspects have been used to allow for a transparent and fair comparison of the conducted analyses.

The analysis was performed based on a smart contract that contains all the vulnerabilities presented in this thesis. The duration of the analysis was very different. Remix, Slither, and Securify were able to analyze the code within a few seconds. Only Mythril and Mythx needed several minutes.

Of the 12 vulnerabilities that were obviously hidden in the smart contract, a total of 11 were found combined by all analyzers. Securify, Remix, and MythX together were able to cover these 11 vulnerabilities.

Unfortunately, Securify can only be used with an outdated Solidity version. This issue was migrated in this thesis by creating a new contract in accordance with the outdated Soldidity version. However, this practice is not feasible in the real world and proves a massive hindrance regarding the general usage of Securify.

Additionally, Mythril, Slither, Securify, Remix, and MythX found 54 further problems. Overall, there were not many intersections between their results. The results included besides security vulnerabilities, general bugs in the code, information on good practices, and Solidity conventions.

Except for MythX, no costs were incurred for an analysis. Since Mythril is part of Mythx, there is no need to use Mythril separately when using MythX.

In conclusion, it is highly advisable to use multiple Solidity source code scanners: The individual applications point out relevant problems in the code, each with a different but essential focus.

However, the combination of all tools and services results in a comprehensive analysis that enables the development of a reasonably secure smart contract.

List of Figures

List of Tables

Bibliography

[ABC17] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. A survey of attacks on ethereum smart contracts sok, 2017.
https://doi.org/10.1007/978-3-662-54455-6_8.

[AW18] A.M. Antonopoulos and G. Wood. *Mastering Ethereum: Building Smart Contracts and DApps.* O'Reilly Media, Incorporated, 2018.

[Con20a] ConsenSys Diligence. Known Attacks, 2020.
https://consensys.github.io/smart-contract-best-practices/known_attacks/
(accessed: October 10, 2020).

[Con20b] ConsenSys Inc. About ConsenSys, 2020. https://consensys.net/about/
(accessed: December 16, 2020).

[Cry20] Crytic. Slither, the Solidity source analyzer, 2020.
https://github.com/crytic/slither
(accessed: December 19, 2020).

[Dan17] Chris Dannen. *Introducing Ethereum and Solidity: Foundations of Cryptocurrency and Blockchain Programming for Beginners.* Apress, USA, 1st edition, 2017.

[Die16] Diedrich, H. *Ethereum: Blockchains, Digital Assets, Smart Contracts, Decentralized Autonomous Organizations.* Wildfire Publishing., 2016.

[Dil20a] ConsenSys Diligence. Mythril, 2020.
https://github.com/ConsenSys/mythril
(accessed: November 06, 2020).

[Dil20b] Consensys Diligence. MythX Smart contract security service for Ethereum, 2020. https://mythx.io
(accessed: November 10, 2020).

[Dil20c] ConsenSys Diligence. Smart Contract Weakness Classification and Test Cases, 2020. https://swcregistry.io/
(accessed: November 02, 2020).

[ETH20a] ETH Zürich — Department of Computer Science — Secure, Reliable, and Intelligent Systems Lab. Secure, Reliable, and Intelligent Systems Lab — SRI Group Website, 2020. `https://www.sri.inf.ethz.ch` (accessed: December 19, 2020).

[Eth20b] Ethereum Foundation. About the Ethereum Foundation, 2020. `https://ethereum.org/en/foundation/` (accessed: December 24, 2020).

[Eth20c] Ethereum Foundation. Informationen für Entwickler, 2020. `https://ethereum.org/de/developers/` (accessed: October 02, 2020).

[Eth20d] Ethereum Foundation. Smart Contract Languages, 2020. `https://ethereum.org/en/developers/docs/smart-contracts/languages/` (accessed: December 22, 2020).

[Eth20e] Ethereum Foundation, Ethereum Wiki. Ethereum Contract Security Techniques and Tips, 2020. `https://eth.wiki/en/howto/smart-contract-safety` (accessed: October 16, 2020).

[Fou20a] Ethereum Foundation. Releases ethereum/solidity, 2020. `https://github.com/ethereum/solidity/releases` (accessed: December 22, 2020).

[Fou20b] Ethereum Foundation. Solidity Documentation, 2020. `https://solidity.readthedocs.io/en/latest/` (accessed: October 31, 2020).

[Fou20c] Ethereum Foundation. Solidity Documentation - Security Considerations, 2020. `https://docs.soliditylang.org/en/latest/security-considerations.html` (accessed: December 30, 2020).

[Fou20d] Ethereum Foundation. Solidity v0.5.0 Breaking Changes, 2020. `https://solidity.readthedocs.io/en/latest/050-breaking-changes.html` (accessed: October 31, 2020).

[Mal18] Maldonado, Fatima Castiglione. *Introduction to Blockchain and Ethereum: Use Distributed Ledgers to Validate Digital Transactions in a Decentralized and Trustless Manner.* Packt Publishing, 2018.

[Mue19] Bernhard Mueller. MythX Tech: Behind the Scenes of Smart Contract Security Anlaysis, December 17, 2019.
https://blog.mythx.io/features/mythx-tech-behind-the-scenes-of-smart-contract-analysis/
(accessed: December 17, 2020).

[Nak08] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System, 2008.
`https://bitcoin.org/bitcoin.pdf`
(accessed: November 12, 2020).

[NCC18] NCC Group GmbH, Decentralized Application Security Project. Top 10, 2018. `https://dasp.co/`
(accessed: October 12, 2020).

[oBI20a] Trail of Bits Inc. Company, 2020.
`https://www.trailofbits.com/about/company/`
(accessed: December 17, 2020).

[oBI20b] Trail of Bits Inc. Trail of Bits Inc., Manticore, 2020.
`https://github.com/trailofbits/manticore`
(accessed: November 06, 2020).

[Pro20] Remix Project. Remix, 2020. `https://remix-project.org/`
(accessed: December 12, 2020).

[Sec20] Secure, Reliable, and Intelligent Systems Lab at ETH Zurich. Securify v2.0, 2020. `https://github.com/eth-sri/securify2`
(accessed: December 19, 2020).

[Vit20] Vitalik Buterin. Ethereum Whitepaper, 2020.
`https://ethereum.org/whitepaper/`
(accessed: November 06, 2020).

[Vos20] Shermin Voshmgir. *Token Economy: How the Web3 reinvents the Internet (Second Edition)*. BlockchainHub Berlin, 2020.

A. Appendix

Contents

A.1. BadBet.sol

```
// Caution! For testing only.
// This smart contract contains many security vulnerabilities and logical errors.
// Author: Kurt Merbeth, 1168947

pragma solidity 0.7.4;

contract BadBet {
    bool public boolean;                            // for toggling random "functions"
    address payable public owner;                   // owner address
    address payable[] public userlist;
    address public toggleContract;
    mapping(address => uint) public balance;        // mapping for user balances
    bool public winner;
    uint256 private seed;
    uint256 public iteration;

    constructor(uint256 _seed, address _toggleContract) {
        owner = msg.sender;         // initialize owner of smart contract
        toggleContract = _toggleContract;
        boolean = true;
        winner = false;
        seed = _seed;
        iteration = 0;
    }

    modifier isWinner() {
        require(winner);
        _;
    }

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

    // deposit funds for bets
    function deposit() public payable {
        addUserToUserlist(msg.sender);
        balance[msg.sender] += msg.value;
    }


    // if user has enough balance for bet
    // withdraw bet amount +50% and substract bet amount from user balance
    // Vulnerability:
```

```
    // 3.5 Integer Over- and Underflow
    function bet(uint _amount) public {
        uint256 amount = 1000000000000000000*_amount;               //
            wei to eth, Integer Overflow
        if(balance[msg.sender]-amount > 0){                          //
            Integer Underflow
            winner = badRandomness();
        }

        // require(winner, "no winner");
        if(winner) {
            withdraw(amount+(amount/2));
            winner = false;
        }
        balance[msg.sender]-=amount;
    }


    // if user is winner, withdraw winner amount
    // Vulnerabilities:
    // 3.1 External Calls - 3.1.1 Unchecked Call Return Value
    // 3.3 Reentrancy
    function withdraw(uint amount) public isWinner payable {
                // isWinner: Reentrancy
        require(address(this).balance > amount);
        msg.sender.call{value: amount}("");                      //
           Unchecked Call Return Value, Reentrancy
    }


    // uses two different random algortihmns
    // returns random boolean
    // true if win
    // false if lose
    function badRandomness() private returns(bool) {
        bool b = false;
        if(boolean) {
            b =  randomWithSeed();
        } else {
            b = randomWithBlockHash();
        }
        callToggleBoolean(toggleContract);
        return b;
    }

    // generates randomly true or false
    // Vulnerability:
    //  3.6 Bad Randomness -  3.6.1 With Private Seed
    function randomWithSeed() private returns(bool) {
        iteration++;
        uint randomNumber = uint(keccak256(abi.encodePacked(seed+
           iteration)));
        return ((randomNumber % 2) == 0);
    }

    // generates randomly true or false
    // Vulnerability:
    //  3.6 Bad Randomness -  3.6.2 With Block Hash
    function randomWithBlockHash() private returns(bool) {
        return (uint(blockhash(block.number-1)) % 2 == 0);
    }

    // if first deposit, add user to userlist
    function addUserToUserlist(address payable _user) private {
        if(balance[_user] == 0) {
            userlist.push(_user);
        }
    }



    // toggles the toggle boolean. For Delegate Call Vulnerability
       only.
    // Vulnerability:
    // 3.1 External Calls - 3.1.2 Delegatecall to Untrusted Callee
    function callToggleBoolean(address _toggleContract) public {
        _toggleContract.delegatecall(abi.encodeWithSignature("
           toggleBoolean(bool)",boolean));
    }

    /*
     *
     * administrative functions
     *
     */

    // emergency payout for all depositors (only owner)
    // if depositor is in userlist, send deposited balance to user
       address
    // Vulnerabilities:
    // 3.2 Denial of Service - 3.2.1 DoS With Unexpected Revert
    //                       - 3.2.2 DoS With Block Gas Limit
    function payOutAll() public isOwner {
        uint256 i = 0;
        while (i < userlist.length  && gasleft() > 200000) {
            require(userlist[i].send(balance[userlist[i]]-20000));
        }
    }

    // set new owner
    // Vulnerability:
    // 3.4 Access Control - 3.4.1 Unprotected Owner Setter
    function setNewOwner(address payable _newOwner) public {
        owner = _newOwner;
    }

    // transfer the contract balance to msg.sender
    // Vulnerability:
    // 3.4 Access Control - 3.4.2 Unprotected Ether Withdrawl
    function getContractBalance() public {
        owner.transfer(address(this).balance);
    }

    // destroys the smart contract
    // sends the balance to the msg.sender
    // Vulnerability:
    // 3.4 Access Control - 3.4.3 Unprotected Selfdestruct
    function destroyContract() public {
        selfdestruct(owner);
    }

}

// Toggle contract, only to use delegatecall
// toggles boolean input
contract Toggle {
    bool public boolean;

    constructor() {
        boolean = false;
    }

    function toggleBoolean(bool _b) public {
        boolean = !_b;
    }
}


/*
Vulnerabilities:
x    3.1 External Calls
x        3.1.1 Unchecked Call Return Value
x        3.1.2 Delegate Call to Untrusted Callee
-        3.1.3 Failed Call Using Send (same as 3.2.1 DoS with
   Unexpected Revert)

x    3.2 Denial Of Service
x        3.2.1 DoS With Unexpected Revert
x        3.2.2 DoS With Block Gas Limit

x    3.3 Reentrancy

x    3.4 Access Control
x        3.4.1 Unprotected Owner Setter
x        3.4.2 Unprotected Ether Withdrawl
x        3.4.3 Unprotected Selfdestruct
```

```

x     3.5 Integer Overflow And Underflow

x      3.6 Bad Randomness
x        3.6.1 With Private Seed
x        3.6.2 With Block Hash
*/
```

Code A.1: BadBet.sol

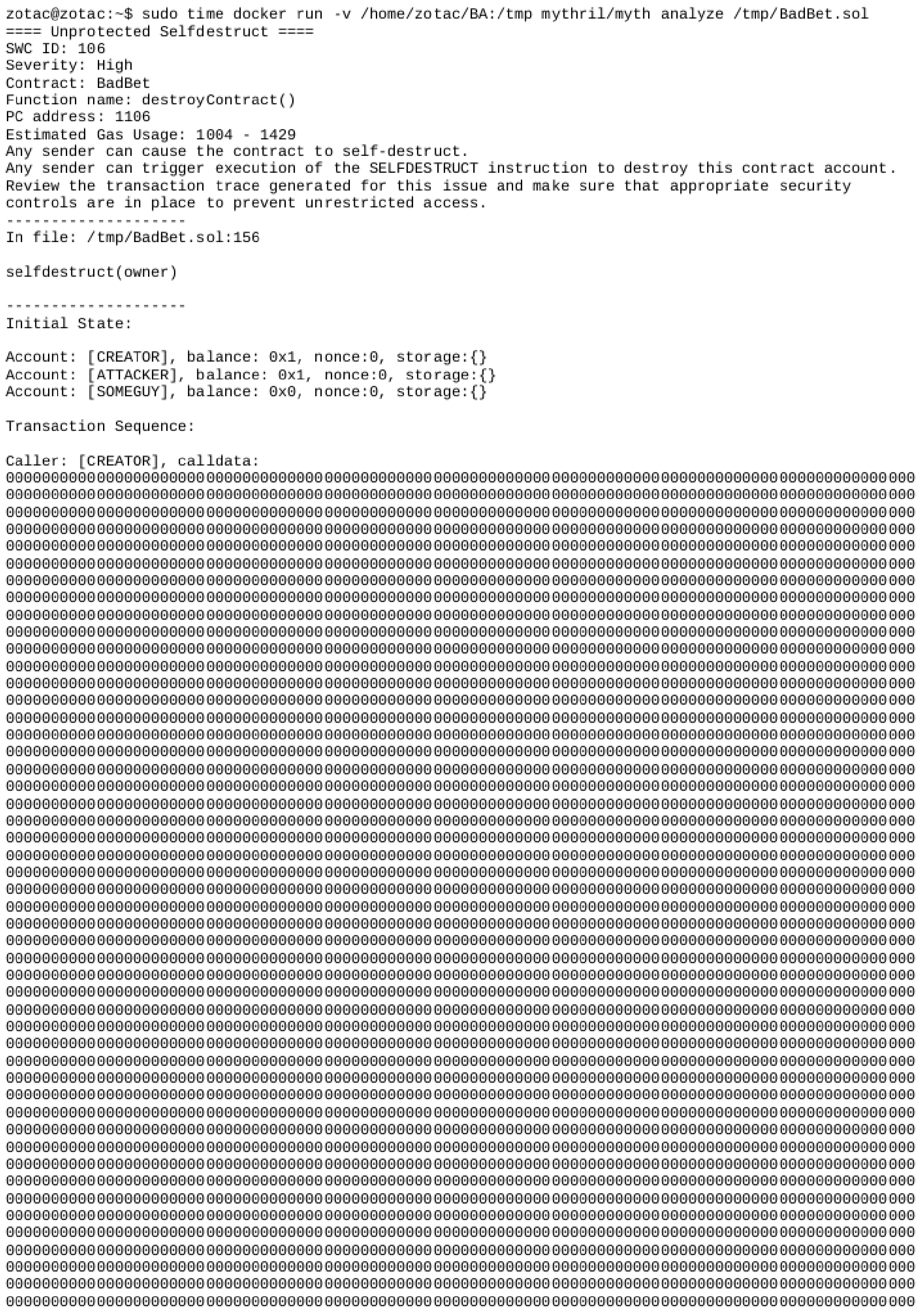
A.2. BadBet0512.sol

```
// Caution! For testing only.
// This smart contract contains many security vulnerabilities and logical errors.
// Kurt Merbeth 1168947

pragma solidity 0.5.12;

contract BadBet {
    bool public boolean;                            // for toggling random "functions"
    address payable public owner;                   // owner address
    address payable[] public userlist;
    address public toggleContract;
    mapping(address => uint) public balance;       // mapping for user balances
    bool public winner;
    uint256 private seed;
    uint256 public iteration;

    constructor(uint256 _seed, address _toggleContract) public {
        owner = msg.sender;         // initialize owner of smart contract
        toggleContract = _toggleContract;
        boolean = true;
        winner = false;
        seed = _seed;
        iteration = 0;
    }

    modifier isWinner() {
        require(winner);
        _;
    }

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

    // deposit funds for bets
    function deposit() public payable {
        addUserToUserlist(msg.sender);
        balance[msg.sender] += msg.value;
    }


    // if user has enough balance for bet
    // withdraw bet amount +50% and substract bet amount from user balance
    // Vulnerability:
```

```
    // 3.5 Integer Over- and Underflow
    function bet(uint _amount) public {
        uint256 amount = 1000000000000000000*_amount;                //
            wei to eth, Integer Overflow
        if(balance[msg.sender]-amount > 0){                          //
            Integer Underflow
            winner = badRandomness();
        }

        // require(winner, "no winner");
        if(winner) {
            withdraw(amount+(amount/2));
            winner = false;
        }
        balance[msg.sender]-=amount;
    }


    // if user is winner, withdraw winner amount
    // Vulnerabilities:
    // 3.1 External Calls - 3.1.1 Unchecked Call Return Value
    // 3.3 Reentrancy
    function withdraw(uint _amount) public isWinner payable {
                  // isWinner: Reentrancy
        require(address(this).balance > _amount);
        msg.sender.call.value(_amount);                       //
           Unchecked Call Return Value, Reentrancy
    }


    // uses two different random algortihmns
    // returns random boolean
    // true if win
    // false if lose
    function badRandomness() private returns(bool) {
        bool b = false;
        if(boolean) {
            b =  randomWithSeed();
        } else {
            b = randomWithBlockHash();
        }
        callToggleBoolean(toggleContract);
        return b;
    }

    // generates randomly true or false
    // Vulnerability:
    // 3.6 Bad Randomness - 3.6.1 With Private Seed
    function randomWithSeed() private returns(bool) {
        iteration++;
        uint randomNumber = uint(keccak256(abi.encodePacked(seed+
           iteration)));
        return ((randomNumber % 2) == 0);
    }

    // generates randomly true or false
    // Vulnerability:
    // 3.6 Bad Randomness - 3.6.2 With Block Hash
    function randomWithBlockHash() private returns(bool) {
        return (uint(blockhash(block.number-1)) % 2 == 0);
    }

    // if first deposit, add user to userlist
    function addUserToUserlist(address payable _user) private {
        if(balance[_user] == 0) {
            userlist.push(_user);
        }
    }



    // toggles the toggle boolean. For Delegate Call Vulnerability
       only.
    // Vulnerability:
    // 3.1 External Calls - 3.1.2 Delegatecall to Untrusted Callee
    function callToggleBoolean(address _toggleContract) public {
        _toggleContract.delegatecall(abi.encodeWithSignature("
           toggleBoolean(bool)",boolean));
    }

    /*
     *
     * administrative functions
     *
     */

    // emergency payout for all depositors (only owner)
    // if depositor is in userlist, send deposited balance to user
       address
    // Vulnerabilities:
    // 3.2 Denial of Service - 3.2.1 DoS With Unexpected Revert
    //                       - 3.2.2 DoS With Block Gas Limit
    function payOutAll() public isOwner {
        uint256 i = 0;
        while (i < userlist.length  && gasleft() > 200000) {
            require(userlist[i].send(balance[userlist[i]]-20000));
        }
    }

    // set new owner
    // Vulnerability:
    // 3.4 Access Control - 3.4.1 Unprotected Owner Setter
    function setNewOwner(address payable _newOwner) public {
        owner = _newOwner;
    }

    // transfer the contract balance to msg.sender
    // Vulnerability:
    // 3.4 Access Control - 3.4.2 Unprotected Ether Withdrawl
    function getContractBalance() public {
        owner.transfer(address(this).balance);
    }

    // destroys the smart contract
    // sends the balance to the msg.sender
    // Vulnerability:
    // 3.4 Access Control - 3.4.3 Unprotected Selfdestruct
    function destroyContract() public {
        selfdestruct(owner);
    }

}

// Toggle contract, only to use delegatecall
// toggles boolean input
contract Toggle {
    bool public boolean;

    constructor() public {
        boolean = false;
    }

    function toggleBoolean(bool _b) public {
        boolean = !_b;
    }
}


/*
Vulnerabilities:
x    3.1 External Calls
x        3.1.1 Unchecked Call Return Value
x        3.1.2 Delegate Call to Untrusted Callee
-        3.1.3 Failed Call Using Send (same as 3.2.1 DoS with
    Unexpected Revert)

x    3.2 Denial Of Service
x        3.2.1 DoS With Unexpected Revert
x        3.2.2 DoS With Block Gas Limit

x    3.3 Reentrancy

x    3.4 Access Control
x        3.4.1 Unprotected Owner Setter
x        3.4.2 Unprotected Ether Withdrawl
x        3.4.3 Unprotected Selfdestruct
```

```

x     3.5 Integer Overflow And Underflow

x     3.6 Bad Randomness
x        3.6.1 With Private Seed
x        3.6.2 With Block Hash
*/
```
Code A.2: BadBet0512.sol

A.3. Mythril Analysis Results



```
zotac@zotac:~$ sudo time docker run -v /home/zotac/BA:/tmp mythril/myth analyze /tmp/BadBet.sol
==== Unprotected Selfdestruct ====
SWC ID: 106
Severity: High
Contract: BadBet
Function name: destroyContract()
PC address: 1106
Estimated Gas Usage: 1004 - 1429
Any sender can cause the contract to self-destruct.
Any sender can trigger execution of the SELFDESTRUCT instruction to destroy this contract account.
Review the transaction trace generated for this issue and make sure that appropriate security
controls are in place to prevent unrestricted access.
--------------------
In file: /tmp/BadBet.sol:156

selfdestruct(owner)

--------------------
Initial State:

Account: [CREATOR], balance: 0x1, nonce:0, storage:{}
Account: [ATTACKER], balance: 0x1, nonce:0, storage:{}
Account: [SOMEGUY], balance: 0x0, nonce:0, storage:{}

Transaction Sequence:

Caller: [CREATOR], calldata:
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

```
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000, value: 0x0
Caller: [ATTACKER], function: destroyContract(), txdata: 0x092a5cce, value: 0x0

==== Delegatecall to user-supplied address ====
SWC ID: 112
Severity: High
Contract: BadBet
Function name: callToggleBoolean(address)
PC address: 1612
Estimated Gas Usage: 2282 - 38444
The contract delegates execution to another contract with a user-supplied address.
The smart contract delegates execution to a user-supplied address.This could allow an attacker to
execute arbitrary code in the context of this contract account and manipulate the state of the
contract account or execute actions on its behalf.
--------------------
In file: /tmp/BadBet.sol:116

_toggleContract.delegatecall(abi.encodeWithSignature("toggleBoolean(bool)",boolean))

--------------------
Initial State:

Account: [CREATOR], balance: 0x1, nonce:0, storage:{}
Account: [ATTACKER], balance: 0x0, nonce:0, storage:{}
Account: [SOMEGUY], balance: 0x0, nonce:0, storage:{}

Transaction Sequence:

Caller: [CREATOR], calldata:
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

```
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000, value: 0x0
Caller: [ATTACKER], function: callToggleBoolean(address), txdata:
0x57342e82efefefefefefefefefefefefdeadbeefdeadbeefdeadbeefdeadbeefdeadbeef, value: 0x0

==== Unchecked return value from external call. ====
SWC ID: 104
Severity: Medium
Contract: BadBet
Function name: callToggleBoolean(address)
PC address: 1612
Estimated Gas Usage: 2349 - 38794
The return value of a message call is not checked.
External calls return a boolean value. If the callee halts with an exception, 'false' is returned
and execution continues in the caller. The caller should check whether an exception happened and
react accordingly to avoid unexpected behavior. For example it is often desirable to wrap external
calls in require() so the transaction is reverted if the call fails.
--------------------
In file: /tmp/BadBet.sol:116

_toggleContract.delegatecall(abi.encodeWithSignature("toggleBoolean(bool)",boolean))
```

```
--------------------
Initial State:

Account: [CREATOR], balance: 0x0, nonce:0, storage:{}
Account: [ATTACKER], balance: 0x0, nonce:0, storage:{}
Account: [SOMEGUY], balance: 0x0, nonce:0, storage:{}

Transaction Sequence:

Caller: [CREATOR], calldata:
0100000000000000000000000000000000000000000000000000000000000000d0d0d0d0d0d0d0d000000000000000000000
0000000000000000000000000000d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
```

```
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0, value: 0x0
Caller: [SOMEGUY], function: deposit(), txdata: 0xd0e30db0, value: 0x0
Caller: [SOMEGUY], function: callToggleBoolean(address), txdata:
0x57342e82d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0, value: 0x0

==== Integer Arithmetic Bugs ====
SWC ID: 101
Severity: High
Contract: BadBet
Function name: bet(uint256)
PC address: 2157
Estimated Gas Usage: 2115 - 2730
The arithmetic operator can overflow.
It is possible to cause an integer overflow or underflow in the arithmetic operation.
--------------------
In file: /tmp/BadBet.sol:48

1000000000000000000*_amount

--------------------
Initial State:

Account: [CREATOR], balance: 0x0, nonce:0, storage:{}
Account: [ATTACKER], balance: 0x1, nonce:0, storage:{}
Account: [SOMEGUY], balance: 0x0, nonce:0, storage:{}

Transaction Sequence:

Caller: [CREATOR], calldata:
0000000000000000000000000000010000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

```
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000, value: 0x0
Caller: [ATTACKER], function: bet(uint256), txdata:
0x7365870b0040000000000000000000000000000000000000000000000000000000000000, value: 0x0


0.37user 0.09system 25:16.99elapsed 0%CPU (0avgtext+0avgdata 58268maxresident)k
0inputs+0outputs (0major+7138minor)pagefaults 0swaps
```

A.4. Slither Analysis Report

```
Compilation warnings/errors on /tmp/BadBet.sol:
Warning: SPDX license identifier not provided in source file. Before publishing, consider adding a comment containing "SPDX-
License-Identifier: <SPDX-License>" to each source file. Use "SPDX-License-Identifier: UNLICENSED" for non-open-source code.
Please see https://spdx.org for more information.
--> /tmp/BadBet.sol

Warning: Return value of low-level calls not used.
  --> /tmp/BadBet.sol:68:9:
   |
68 |        msg.sender.call{value: amount}("");                // Unchecked Call Return Value, Reentrancy
   |        ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Warning: Return value of low-level calls not used.
   --> /tmp/BadBet.sol:116:9:
    |
116 |        _toggleContract.delegatecall(abi.encodeWithSignature("toggleBoolean(bool)",boolean));
    |        ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Warning: Function state mutability can be restricted to view
  --> /tmp/BadBet.sol:99:5:
   |
99 |    function randomWithBlockHash() private returns(bool) {
   |    ^ (Relevant source part starts here and spans across multiple lines).


INFO:Detectors:
BadBet.withdraw(uint256) (../../tmp/BadBet.sol#66-69) sends eth to arbitrary user
        Dangerous calls:
        - msg.sender.call{value: amount}() (../../tmp/BadBet.sol#68)
BadBet.getContractBalance() (../../tmp/BadBet.sol#147-149) sends eth to arbitrary user
        Dangerous calls:
        - owner.transfer(address(this).balance) (../../tmp/BadBet.sol#148)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations
INFO:Detectors:
BadBet.callToggleBoolean(address) (../../tmp/BadBet.sol#115-117) uses delegatecall to a input-controlled function id
        - _toggleContract.delegatecall(abi.encodeWithSignature(toggleBoolean(bool),boolean)) (../../tmp/BadBet.sol#116)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#controlled-delegatecall
INFO:Detectors:
Reentrancy in BadBet.bet(uint256) (../../tmp/BadBet.sol#47-59):
        External calls:
        - winner = badRandomness() (../../tmp/BadBet.sol#50)
                - _toggleContract.delegatecall(abi.encodeWithSignature(toggleBoolean(bool),boolean))
(../../tmp/BadBet.sol#116)
        - withdraw(amount + (amount / 2)) (../../tmp/BadBet.sol#55)
                - msg.sender.call{value: amount}() (../../tmp/BadBet.sol#68)
        External calls sending eth:
        - withdraw(amount + (amount / 2)) (../../tmp/BadBet.sol#55)
                - msg.sender.call{value: amount}() (../../tmp/BadBet.sol#68)
        State variables written after the call(s):
        - balance[msg.sender] -= amount (../../tmp/BadBet.sol#58)
        - winner = false (../../tmp/BadBet.sol#56)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities
INFO:Detectors:
BadBet.destroyContract() (../../tmp/BadBet.sol#155-157) allows anyone to destruct the contract
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#suicidal

INFO:Detectors:
BadBet.randomWithBlockHash() (../../tmp/BadBet.sol#99-101) uses a dangerous strict equality:
        - (uint256(blockhash(uint256)(block.number - 1)) % 2 == 0) (../../tmp/BadBet.sol#100)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
BadBet.withdraw(uint256) (../../tmp/BadBet.sol#66-70) ignores return value by msg.sender.call{value: amount}()
(../../tmp/BadBet.sol#68)
BadBet.callToggleBoolean(address) (../../tmp/BadBet.sol#115-117) ignores return value by
_toggleContract.delegatecall(abi.encodeWithSignature(toggleBoolean(bool),boolean)) (../../tmp/BadBet.sol#116)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-low-level-calls
```

```
INFO:Detectors:
BadBet.payOutAll() (../../tmp/BadBet.sol#130-135) has external calls inside a loop: require(bool)(userlist[i].send(balance[userlist[i]]
- 20000)) (../../tmp/BadBet.sol#133)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop
INFO:Detectors:
Pragma version0.7.4 (../../tmp/BadBet.sol#5) necessitates a version too recent to be trusted. Consider deploying with 0.6.11
solc-0.7.4 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in BadBet.withdraw(uint256) (../../tmp/BadBet.sol#66-69):
        - msg.sender.call{value: amount}() (../../tmp/BadBet.sol#68)
Low level call in BadBet.callToggleBoolean(address) (../../tmp/BadBet.sol#115-117):
        - _toggleContract.delegatecall(abi.encodeWithSignature(toggleBoolean(bool),boolean)) (../../tmp/BadBet.sol#116)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter BadBet.bet(uint256)._amount (../../tmp/BadBet.sol#47) is not in mixedCase
Parameter BadBet.addUserToUserlist(address)._user (../../tmp/BadBet.sol#104) is not in mixedCase
Parameter BadBet.callToggleBoolean(address)._toggleContract (../../tmp/BadBet.sol#115) is not in mixedCase
Parameter BadBet.setNewOwner(address)._newOwner (../../tmp/BadBet.sol#140) is not in mixedCase
Parameter Toggle.toggleBoolean(bool)._b (../../tmp/BadBet.sol#170) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
BadBet.bet(uint256) (../../tmp/BadBet.sol#47-59) uses literals with too many digits:
        - amount = 1000000000000000000 * _amount (../../tmp/BadBet.sol#48)
BadBet.payOutAll() (../../tmp/BadBet.sol#130-135) uses literals with too many digits:
        - i < userlist.length && gasleft()() > 200000 (../../tmp/BadBet.sol#132)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
deposit() should be declared external:
        - BadBet.deposit() (../../tmp/BadBet.sol#37-40)
bet(uint256) should be declared external:
        - BadBet.bet(uint256) (../../tmp/BadBet.sol#47-59)
payOutAll() should be declared external:
        - BadBet.payOutAll() (../../tmp/BadBet.sol#130-135)
setNewOwner(address) should be declared external:
        - BadBet.setNewOwner(address) (../../tmp/BadBet.sol#140-142)
getContractBalance() should be declared external:
        - BadBet.getContractBalance() (../../tmp/BadBet.sol#147-149)
destroyContract() should be declared external:
        - BadBet.destroyContract() (../../tmp/BadBet.sol#155-157)
toggleBoolean(bool) should be declared external:
        - Toggle.toggleBoolean(bool) (../../tmp/BadBet.sol#170-172)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:/tmp/BadBet.sol analyzed (2 contracts with 46 detectors), 27 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration

real    0m2.021s user    0m1.775s sys     0m0.254s
```

A.5. Securify Analysis Report

```
time docker run -it -v /home/zotac/BA:/tmp securify /tmp/BadBet0512.sol


Severity:    MEDIUM
Pattern:     Dangerous Strict Equalities
Description: Strict equalities that use account's balance, timestamps
             and block numbers should be avoided
Type:        Violation
Contract:    BadBet
Line:        100
Source:
>     function randomWithBlockHash() private returns(bool) {
>         return (uint(blockhash(block.number-1)) % 2 == 0);
>                ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
>     }


Severity:    HIGH
Pattern:     Delegatecall or callcode to unrestricted address
Description: The address of a delegatecall or callcode must be
             approved by the contract owner.
Type:        Violation
Contract:    BadBet
Line:        116
Source:
>     function callToggleBoolean(address _toggleContract) public {
>         _toggleContract.delegatecall(abi.encodeWithSignature("toggleBoolean(bool)",boolean));
>         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
>     }


Severity:    LOW
Pattern:     External Calls of Functions
Description: A public function that is never called within the
             contract should be marked as external
Type:        Violation
Contract:    BadBet
Line:        37
Source:
>     // deposit funds for bets
>     function deposit() public payable {
>     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
>         addUserToUserlist(msg.sender);


Severity:    LOW
Pattern:     External Calls of Functions
Description: A public function that is never called within the
             contract should be marked as external
Type:        Violation
Contract:    BadBet
Line:        47
Source:
>     // 3.5 Integer Over- and Underflow
>     function bet(uint _amount) public {
>     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
>         uint256 amount = 1000000000000000000*_amount;          // wei to eth, Integer Overflow
```

```
Severity:    LOW
Pattern:     External Calls of Functions
Description: A public function that is never called within the
             contract should be marked as external
Type:        Violation
Contract:    BadBet
Line:        130
Source:
>     //                        - 3.2.2 DoS With Block Gas Limit
>     function payOutAll() public isOwner {
>     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
>         uint256 i = 0;


Severity:    LOW
Pattern:     External Calls of Functions
Description: A public function that is never called within the
             contract should be marked as external
Type:        Violation
Contract:    BadBet
Line:        140
Source:
>     // 3.4 Access Control - 3.4.1 Unprotected Owner Setter
>     function setNewOwner(address payable _newOwner) public {
>     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
>         owner = _newOwner;


Severity:    LOW
Pattern:     External Calls of Functions
Description: A public function that is never called within the
             contract should be marked as external
Type:        Violation
Contract:    BadBet
Line:        147
Source:
>     // 3.4 Access Control - 3.4.2 Unprotected Ether Withdrawl
>     function getContractBalance() public {
>     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
>         owner.transfer(address(this).balance);


Severity:    LOW
Pattern:     External Calls of Functions
Description: A public function that is never called within the
             contract should be marked as external
Type:        Violation
Contract:    BadBet
Line:        155
Source:
>     // 3.4 Access Control - 3.4.3 Unprotected Selfdestruct
>     function destroyContract() public {
>     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
>         selfdestruct(owner);


Severity:    LOW
Pattern:     External Calls of Functions
Description: A public function that is never called within the
             contract should be marked as external
Type:        Violation
Contract:    Toggle
Line:        170
Source:
>
>     function toggleBoolean(bool _b) public {
>     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
>         boolean = !_b;
```

```
Severity:    HIGH
Pattern:     External call in loop
Description: If a single call in the loop fails or revers, it will
             cause all other calls to fail as well.
Type:        Violation
Contract:    BadBet
Line:        133
Source:
>         while (i < userlist.length  && gasleft() > 200000) {
>             require(userlist[i].send(balance[userlist[i]]-20000));
>                     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
>         }


Severity:    INFO
Pattern:     Low Level Calls
Description: Usage of <address>.call should be avoided
Type:        Violation
Contract:    BadBet
Line:        116
Source:
>     function callToggleBoolean(address _toggleContract) public {
>         _toggleContract.delegatecall(abi.encodeWithSignature("toggleBoolean(bool)",boolean));
>         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
>     }


Severity:    MEDIUM
Pattern:     Missing Input Validation
Description: Method arguments must be sanitized before they are used
             in computations.
Type:        Warning
Contract:    BadBet
Line:        47
Source:
>     // 3.5 Integer Over- and Underflow
>     function bet(uint _amount) public {
>     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
>         uint256 amount = 1000000000000000000*_amount;               // wei to eth, Integer Overflow


Severity:    MEDIUM
Pattern:     Missing Input Validation
Description: Method arguments must be sanitized before they are used
             in computations.
Type:        Warning
Contract:    BadBet
Line:        140
Source:
>     // 3.4 Access Control - 3.4.1 Unprotected Owner Setter
>     function setNewOwner(address payable _newOwner) public {
>     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
>         owner = _newOwner;

Severity:    MEDIUM
Pattern:     Missing Input Validation
Description: Method arguments must be sanitized before they are used
             in computations.
Type:        Violation
Contract:    BadBet
Line:        66
Source:
>     // 3.3 Reentrancy
>     function withdraw(uint _amount) public isWinner payable {            // isWinner: Reentrancy
>     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
>         require(address(this).balance > _amount);
```

```
Severity:    MEDIUM
Pattern:     Missing Input Validation
Description: Method arguments must be sanitized before they are used
             in computations.
Type:        Violation
Contract:    BadBet
Line:        115
Source:
>     // 3.1 External Calls - 3.1.2 Delegatecall to Untrusted Callee
>     function callToggleBoolean(address _toggleContract) public {
>     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
>         _toggleContract.delegatecall(abi.encodeWithSignature("toggleBoolean(bool)",boolean));


Severity:    MEDIUM
Pattern:     Missing Input Validation
Description: Method arguments must be sanitized before they are used
             in computations.
Type:        Violation
Contract:    Toggle
Line:        170
Source:
>
>     function toggleBoolean(bool _b) public {
>     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
>         boolean = !_b;


Severity:    INFO
Pattern:     Too Many Digit Literals
Description: Usage of assembly in Solidity code is discouraged.
Type:        Violation
Contract:    BadBet
Line:        48
Source:
>     function bet(uint _amount) public {
>         uint256 amount = 1000000000000000000*_amount;               // wei to eth, Integer Overflow
>                          ^^^^^^^^^^^^^^^^^^^^^
>         if(balance[msg.sender]-amount > 0){                        // Integer Underflow


Severity:    INFO
Pattern:     Too Many Digit Literals
Description: Usage of assembly in Solidity code is discouraged.
Type:        Violation
Contract:    BadBet
Line:        132
Source:
>         uint256 i = 0;
>         while (i < userlist.length  && gasleft() > 200000) {
>                                                    ^^^^^^
>             require(userlist[i].send(balance[userlist[i]]-20000));


Severity:    CRITICAL
Pattern:     Transaction Order Affects Ether Amount
Description: The amount of ether transferred must not be influenced by
             other transactions.
Type:        Violation
Contract:    BadBet
Line:        133
Source:
>         while (i < userlist.length  && gasleft() > 200000) {
>             require(userlist[i].send(balance[userlist[i]]-20000));
>                     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
>         }
```

```
Severity:    CRITICAL
Pattern:     Transaction Order Affects Ether Amount
Description: The amount of ether transferred must not be influenced by
             other transactions.
Type:        Violation
Contract:    BadBet
Line:        148
Source:
>     function getContractBalance() public {
>         owner.transfer(address(this).balance);
>         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
>     }


Severity:    CRITICAL
Pattern:     Transaction Order Affects Ether Receiver
Description: The receiver of ether transfers must not be influenced by
             other transactions.
Type:        Violation
Contract:    BadBet
Line:        133
Source:
>         while (i < userlist.length  && gasleft() > 200000) {
>             require(userlist[i].send(balance[userlist[i]]-20000));
>                     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
>         }


Severity:    CRITICAL
Pattern:     Transaction Order Affects Ether Receiver
Description: The receiver of ether transfers must not be influenced by
             other transactions.
Type:        Violation
Contract:    BadBet
Line:        148
Source:
>     function getContractBalance() public {
>         owner.transfer(address(this).balance);
>         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
>     }


Severity:    CRITICAL
Pattern:     Transaction Order Affects Execution of Ether Transfer
Description: Ether transfers whose execution can be manipulated by
             other transactions must be inspected for unintended
             behavior.
Type:        Violation
Contract:    BadBet
Line:        133
Source:
>         while (i < userlist.length  && gasleft() > 200000) {
>             require(userlist[i].send(balance[userlist[i]]-20000));
>                     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
>         }


Severity:    CRITICAL
Pattern:     Transaction Order Affects Execution of Ether Transfer
Description: Ether transfers whose execution can be manipulated by
             other transactions must be inspected for unintended
             behavior.
Type:        Violation
Contract:    BadBet
Line:        148
Source:
>     function getContractBalance() public {
>         owner.transfer(address(this).balance);
>         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
>     }
```

```
Severity:    CRITICAL
Pattern:     Unrestricted Ether Flow
Description: The execution of ether flows should be restricted to an
             authorized set of users.
Type:        Warning
Contract:    BadBet
Line:        116
Source:
>     function callToggleBoolean(address _toggleContract) public {
>         _toggleContract.delegatecall(abi.encodeWithSignature("toggleBoolean(bool)",boolean));
>         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
>     }


Severity:    CRITICAL
Pattern:     Unrestricted Ether Flow
Description: The execution of ether flows should be restricted to an
             authorized set of users.
Type:        Violation
Contract:    BadBet
Line:        148
Source:
>     function getContractBalance() public {
>         owner.transfer(address(this).balance);
>         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
>     }


Severity:    CRITICAL
Pattern:     Unrestricted call to selfdestruct
Description: Calls to selfdestruct that can be triggered by any user
             must be inspected.
Type:        Violation
Contract:    BadBet
Line:        156
Source:
>     function destroyContract() public {
>         selfdestruct(owner);
>         ^^^^^^^^^^^^^^^^^^^
>     }


Severity:    CRITICAL
Pattern:     Unrestricted write to storage
Description: Contract fields that can be modified by any user must be
             inspected.
Type:        Warning
Contract:    BadBet
Line:        106
Source:
>         if(balance[_user] == 0) {
>             userlist.push(_user);
>             ^^^^^^^^^^^^^^^^^^^^
>         }


Severity:    CRITICAL
Pattern:     Unrestricted write to storage
Description: Contract fields that can be modified by any user must be
             inspected.
Type:        Violation
Contract:    BadBet
Line:        141
Source:
>     function setNewOwner(address payable _newOwner) public {
>         owner = _newOwner;
>         ^^^^^^^^^^^^^^^^^
>     }
```

```
Severity:    CRITICAL
Pattern:     Unrestricted write to storage
Description: Contract fields that can be modified by any user must be
             inspected.
Type:        Violation
Contract:    Toggle
Line:        171
Source:
>     function toggleBoolean(bool _b) public {
>         boolean = !_b;
>         ^^^^^^^^^^^^^^
>     }



Severity:    MEDIUM
Pattern:     Unused Return Pattern
Description: The value returned by an external function call is never
             used
Type:        Violation
Contract:    BadBet
Line:        116
Source:
>     function callToggleBoolean(address _toggleContract) public {
>         _toggleContract.delegatecall(abi.encodeWithSignature("toggleBoolean(bool)",boolean));
>         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
>     }


Severity:    INFO
Pattern:     Unused State Variable
Description: Unused state variables should be removed.
Type:        Violation
Contract:    Toggle
Line:        164
Source:
> contract Toggle {
>     bool public boolean;
>     ^^^^^^^^^^^^^^^^^^^^
>
0.11 user 0.10 system 0:22.89 elapsed
```

A.6. Remix Analysis Report

Low level calls:
Use of "call": should be avoided whenever possible.
It can lead to unexpected behavior if return value is not handled properly.
Please use Direct Calls via specifying the called contract's interface.
more[2]
Pos: 68:8:

Low level calls:
Use of "delegatecall": should be avoided whenever possible.
External code, that is called can change the state of the calling contract and send ether from the caller's balance.
If this is wanted behaviour, use the Solidity library feature if possible.
more[3]
Pos: 116:8:

Low level calls:
Use of "send": "send" does not throw an exception when not successful, make sure you deal with the failure case accordingly. Use "transfer" whenever failure of the ether transfer should rollback the whole transaction. Note: if you "send/transfer" ether to a contract the fallback function is called, the callees fallback function is very limited due to the limited amount of gas provided by "send/transfer". No state changes are possible but the callee can log the event or revert the transfer. "send/transfer" is syntactic sugar for a "call" to the fallback function with 2300 gas and a specified ether value. more[4]
Pos: 133:20:

[2]https://solidity.readthedocs.io/en/v0.7.4/control-structures.html?#external-function-calls
[3]https://solidity.readthedocs.io/en/v0.7.4/contracts.html#libraries
[4]https://solidity.readthedocs.io/en/v0.7.4/security-considerations.html#sending-and-receiving-ether

Block Hash:
Use of ”blockhash”: ”blockhash(uint blockNumber)” is used to access the last 256 block hashes.
A miner computes the block hash by ”summing up” the information in the current block mined.
By ”summing up” the information cleverly, a miner can try to influence the outcome of a transaction in the current block.
This is especially easy if there are only a small number of equally likely outcomes.
Pos: 100:21:

Selfdestruct:
Use of selfdestruct: Can block calling contracts unexpectedly. Be especially careful if this contract is planned to be used by other contracts (i.e. library contracts, interactions).
Selfdestruction of the callee contract can leave callers in an inoperable state.
more[1]
Pos: 156:8:

Gas & Economy

Gas Costs:
Gas requirement of function BadBet.bet is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 47:4:

Gas Costs:
Gas requirement of function BadBet.withdraw is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 66:4:

[1]https://paritytech.io/blog/security-alert.html

Gas Costs:
Gas requirement of function BadBet.callToggleBoolean is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 115:4

Gas Costs:
Gas requirement of function BadBet.payOutAll is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 130:4:

Gas Costs:
Gas requirement of function BadBet.getContractBalance is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 147:4:

Miscellaneous

Constant/View/Pure functions:
BadBet.randomWithBlockHash() : Potentially should be constant/view/pure but is not.
Note: Modifiers are currently not considered by this static analysis.
more[1]
Pos: 99:4:

[1]https://solidity.readthedocs.io/en/v0.7.4/contracts.html#view-functions

Constant/View/Pure functions:
BadBet.addUserToUserlist(address payable) : Potentially should be constant/view/pure but is not.
Note: Modifiers are currently not considered by this static analysis.
more[1]
Pos: 104:4:

Guard conditions:
Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code).
Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more[2]
Pos: 27:8

Guard conditions:
Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code).
Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more[2]
Pos: 32:8

Guard conditions:
Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code).
Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more[2]
Pos: 67:8

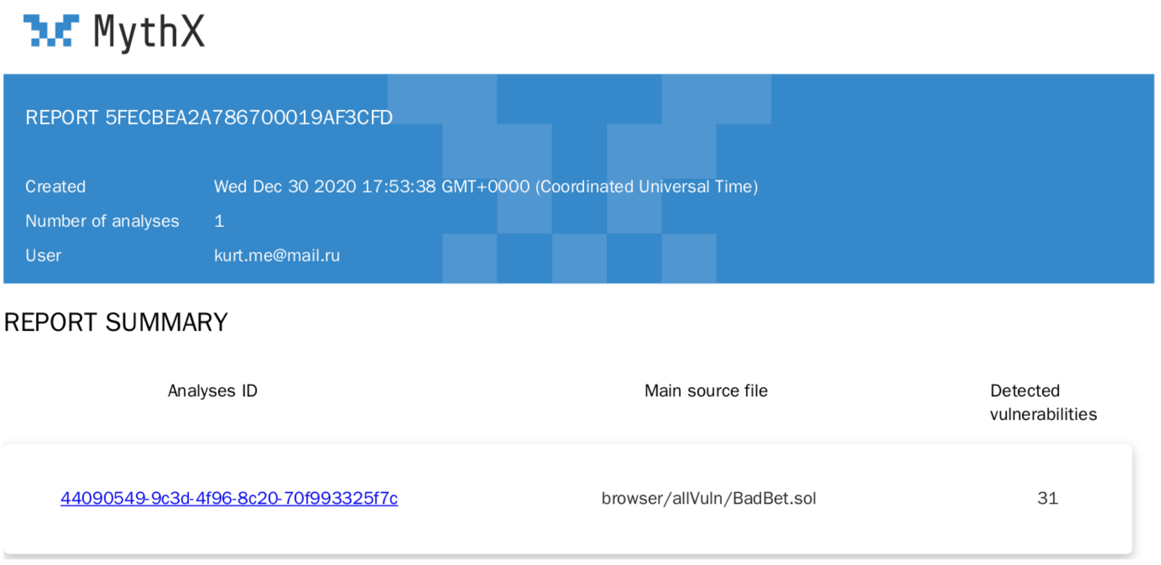
[1]https://solidity.readthedocs.io/en/v0.7.4/contracts.html#view-functions
[2]https://solidity.readthedocs.io/en/v0.7.4/control-structures.html#error-handling-assert-require-revert-and-exceptions

Guard conditions:
Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code).
Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more[1]
Pos: 133:12

Data truncated:
Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again.
This does not hold for division of (only) literal values since those yield rational constants.
Pos: 55:29:

[1]https://solidity.readthedocs.io/en/v0.7.4/control-structures.html#error-handling-assert-require-revert-and-exceptions

A.7. MythX Analysis Report



MythX

REPORT 5FECBEA2A786700019AF3CFD

Created	Wed Dec 30 2020 17:53:38 GMT+0000 (Coordinated Universal Time)
Number of analyses	1
User	kurt.me@mail.ru

REPORT SUMMARY

Analyses ID	Main source file	Detected vulnerabilities
44090549-9c3d-4f96-8c20-70f993325f7c	browser/allVuln/BadBet.sol	31

Analysis 44090549-9c3d-4f96-8c20-70f993325f7c MythX

Started	Wed Dec 30 2020 17:53:41 GMT+0000 (Coordinated Universal Time)
Finished	Wed Dec 30 2020 18:38:49 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Remythx
Main Source File	Browser/AllVuln/BadBet.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
7	11	13

ISSUES

HIGH

SWC-101

The arithmetic operation can overflow.

It is possible to cause an arithmetic overflow. Prevent the overflow by constraining inputs using the require() statement or use the OpenZeppelin SafeMath library for integer arithmetic operations. Refer to the transaction trace generated for this issue to reproduce the overflow.

Source file

browser/allVuln/BadBet.sol

Locations

```
// 3.5 Integer Over- and Underflow
function bet(uint _amount) public {
uint256 amount = 1000000000000000000*_amount; // wei to eth, Integer Overflow
if(balance[msg.sender]-amount > 0){ // Integer Underflow
winner = badRandomness();
```

HIGH

SWC-101

The arithmetic operation can underflow.

It is possible to cause an arithmetic underflow. Prevent the underflow by constraining inputs using the require() statement or use the OpenZeppelin SafeMath library for integer arithmetic operations. Refer to the transaction trace generated for this issue to reproduce the underflow.

Source file

browser/allVuln/BadBet.sol

Locations

```
function bet(uint _amount) public {
uint256 amount = 1000000000000000000*_amount; // wei to eth, Integer Overflow
if(balance[msg.sender]-amount > 0){ // Integer Underflow
winner = badRandomness();
}
```

HIGH

SWC-101

The arithmetic operator can overflow.

It is possible to cause an integer overflow or underflow in the arithmetic operation.

Source file

`browser/allVuln/BadBet.sol`

Locations

```
function randomWithSeed() private returns(bool) {
iteration++;
uint randomNumber = uint(keccak256(abi.encodePacked(seed+iteration)));
return ((randomNumber % 2) == 0);
}
```

HIGH

SWC-101

The arithmetic operation can underflow.

It is possible to cause an arithmetic underflow. Prevent the underflow by constraining inputs using the require() statement or use the OpenZeppelin SafeMath library for integer arithmetic operations. Refer to the transaction trace generated for this issue to reproduce the underflow.

Source file

`browser/allVuln/BadBet.sol`

Locations

```
winner = false;
}
balance[msg.sender]-=amount;
}
```

HIGH

SWC-101

The arithmetic operator can overflow.

It is possible to cause an integer overflow or underflow in the arithmetic operation.

Source file

`browser/allVuln/BadBet.sol`

Locations

```
// require(winner, "no winner");
if(winner) {
withdraw(amount+(amount/2));
winner = false;
}
```

HIGH

SWC-106

Any sender can cause the contract to self-destruct.

Any sender can trigger execution of the SELFDESTRUCT instruction to destroy this contract account. Review the transaction trace generated for this issue and make sure that appropriate security controls are in place to prevent unrestricted access.

Source file

`browser/allVuln/BadBet.sol`

Locations

```
// 3.4 Access Control - 3.4.3 Unprotected Selfdestruct
function destroyContract() public {
selfdestruct(owner);
}
```

HIGH

SWC-112

The contract delegates execution to another contract with a user-supplied address.

The smart contract delegates execution to a user-supplied address.This could allow an attacker to execute arbitrary code in the context of this contract account and manipulate the state of the contract account or execute actions on its behalf.

Source file

`browser/allVuln/BadBet.sol`

Locations

```
// 3.1 External Calls - 3.1.2 Delegatecall to Untrusted Callee
function callToggleBoolean(address _toggleContract) public {
_toggleContract.delegatecall(abi.encodeWithSignature("toggleBoolean(bool)",boolean));
}
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "deposit" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

`browser/allVuln/BadBet.sol`

Locations

```

// deposit funds for bets
function deposit() public payable {
addUserToUserlist(msg.sender);
balance[msg.sender] += msg.value;
}

```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "bet" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

`browser/allVuln/BadBet.sol`

Locations

```
// Vulnerability:
// 3.5 Integer Over- and Underflow
function bet(uint _amount) public {
uint256 amount = 1000000000000000000*_amount; // wei to eth, Integer Overflow
if(balance[msg.sender]-amount > 0){ // Integer Underflow
winner = badRandomness();
}

// require(winner, "no winner");
if(winner) {
withdraw(amount+(amount/2));
winner = false;
}
balance[msg.sender]-=amount;
}

```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "payOutAll" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

`browser/allVuln/BadBet.sol`

Locations

```
// 3.2 Denial of Service - 3.2.1 DoS With Unexpected Revert
// - 3.2.2 DoS With Block Gas Limit
function payOutAll() public isOwner {
uint256 i = 0;
while (i < userlist.length && gasleft() > 200000) {
require(userlist[i].send(balance[userlist[i]]-20000));
}
}

// set new owner
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "setNewOwner" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

`browser/allVuln/BadBet.sol`

Locations

```
// Vulnerability:
// 3.4 Access Control - 3.4.1 Unprotected Owner Setter
function setNewOwner(address payable _newOwner) public {
owner = _newOwner;
}

// transfer the contract balance to msg.sender
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "getContractBalance" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

`browser/allVuln/BadBet.sol`

Locations

```
// Vulnerability:
// 3.4 Access Control - 3.4.2 Unprotected Ether Withdrawl
function getContractBalance() public {
owner.transfer(address(this).balance);
}

// destroys the smart contract
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "destroyContract" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

`browser/allVuln/BadBet.sol`

Locations

```
// Vulnerability:
// 3.4 Access Control - 3.4.3 Unprotected Selfdestruct
function destroyContract() public {
selfdestruct(owner);
}

}
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "toggleBoolean" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

`browser/allVuln/BadBet.sol`

Locations

```
}

function toggleBoolean(bool _b) public {
boolean = !_b;
}
}
```

MEDIUM

SWC-104

Unchecked return value from low-level external call.

Low-level external calls return a boolean value. If the callee halts with an exception, 'false' is returned and execution continues in the caller. The caller should check whether an exception happened and react accordingly to avoid unexpected behavior. For example it is often desirable to wrap low-level external calls in require() so the transaction is reverted if the call fails.

Source file

`browser/allVuln/BadBet.sol`

Locations

```
// 3.1 External Calls - 3.1.2 Delegatecall to Untrusted Callee
function callToggleBoolean(address _toggleContract) public {
_toggleContract.delegatecall(abi.encodeWithSignature("toggleBoolean(bool)",boolean));
}
```

MEDIUM

SWC-104

Unchecked return value from low-level external call.

Low-level external calls return a boolean value. If the callee halts with an exception, 'false' is returned and execution continues in the caller. The caller should check whether an exception happened and react accordingly to avoid unexpected behavior. For example it is often desirable to wrap low-level external calls in require() so the transaction is reverted if the call fails.

Source file

`browser/allVuln/BadBet.sol`

Locations

```
function withdraw(uint amount) public isWinner payable { // isWinner: Reentrancy
require(address(this).balance > amount);
msg.sender.call{value: amount}(""); // Unchecked Call Return Value, Reentrancy
}
```

MEDIUM

SWC-113

Multiple calls are executed in the same transaction.

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source file

`browser/allVuln/BadBet.sol`

Locations

```
uint256 i = 0;
while (i < userlist.length && gasleft() > 200000) {
require(userlist[i].send(balance[userlist[i]]-20000));
}
}
```

MEDIUM

SWC-113

Multiple calls are executed in the same transaction.

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source file

`browser/allVuln/BadBet.sol`

Locations

```
function withdraw(uint amount) public isWinner payable { // isWinner: Reentrancy
require(address(this).balance > amount);
msg.sender.call{value: amount}(""); // Unchecked Call Return Value, Reentrancy
}
```

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

`browser/allVuln/BadBet.sol`

Locations

```
// 3.3 Reentrancy
function withdraw(uint amount) public isWinner payable { // isWinner: Reentrancy
require(address(this).balance > amount);
msg.sender.call{value: amount}(""); // Unchecked Call Return Value, Reentrancy
}
```

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

`browser/allVuln/BadBet.sol`

Locations

```
uint256 amount = 1000000000000000000*_amount; // wei to eth, Integer Overflow
if(balance[msg.sender]-amount > 0){ // Integer Underflow
winner = badRandomness();
}
```

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

`browser/allVuln/BadBet.sol`

Locations

```

modifier isWinner() {
require(winner);
_;
}
```

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

`browser/allVuln/BadBet.sol`

Locations

```
if(winner) {
withdraw(amount+(amount/2));
winner = false;
}
balance[msg.sender]-=amount;
```

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

`browser/allVuln/BadBet.sol`

Locations

```
winner = false;
}
balance[msg.sender]-=amount;
}
```

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

`browser/allVuln/BadBet.sol`

Locations

```

// require(winner, "no winner");
if(winner) {
withdraw(amount+(amount/2));
winner = false;
```

LOW

SWC-107

Write to persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

`browser/allVuln/BadBet.sol`

Locations

```
function withdraw(uint amount) public isWinner payable { // isWinner: Reentrancy
require(address(this).balance > amount);
msg.sender.call{value: amount}(""); // Unchecked Call Return Value, Reentrancy
}
```

LOW

SWC-120

Potential use of "blockhash" as source of randonmness.

The environment variable "blockhash" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

`browser/allVuln/BadBet.sol`

Locations

```
// 3.6 Bad Randomness - 3.6.2 With Block Hash
function randomWithBlockHash() private returns(bool) {
return (uint(blockhash(block.number-1)) % 2 == 0);
}
```

LOW

SWC-120

Potential use of "block.number" as source of randonmness.

The environment variable "block.number" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

`browser/allVuln/BadBet.sol`

Locations

```
// 3.6 Bad Randomness - 3.6.2 With Block Hash
function randomWithBlockHash() private returns(bool) {
return (uint(blockhash(block.number-1)) % 2 == 0);
}
```

LOW

SWC-123

Requirement violation.

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

Source file

`browser/allVuln/BadBet.sol`

Locations

```
// 3.1 External Calls - 3.1.2 Delegatecall to Untrusted Callee
function callToggleBoolean(address _toggleContract) public {
_toggleContract.delegatecall(abi.encodeWithSignature("toggleBoolean(bool)",boolean));
}
```

LOW

SWC-123

Requirement violation.

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

Source file

`browser/allVuln/BadBet.sol`

Locations

```
// 3.4 Access Control - 3.4.2 Unprotected Ether Withdrawl
function getContractBalance() public {
owner.transfer(address(this).balance);
}
```

Source file

`browser/allVuln/BadBet.sol`

Locations

```
pragma solidity 0.7.4;

contract BadBet {
bool public boolean; // for toggling random "functions"
address payable public owner; // owner address
address payable[] public userlist;
address public toggleContract;
mapping(address => uint) public balance; // mapping for user balances
bool public winner;
uint256 private seed;
uint256 public iteration;

constructor(uint256 _seed, address _toggleContract) {
owner = msg.sender; // initialize owner of smart contract
toggleContract = _toggleContract;
boolean = true;
winner = false;
seed = _seed;
iteration = 0;
}

modifier isWinner() {
require(winner);
_;
}

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

// deposit funds for bets
function deposit() public payable {
addUserToUserlist(msg.sender);
balance[msg.sender] += msg.value;
}


// if user has enough balance for bet
// withdraw bet amount +50% and substract bet amount from user balance
// Vulnerability:
// 3.5 Integer Over- and Underflow
function bet(uint _amount) public {
uint256 amount = 1000000000000000000*_amount; // wei to eth, Integer Overflow
if(balance[msg.sender]-amount > 0){ // Integer Underflow
winner = badRandomness();
}

// require(winner, "no winner");
if(winner) {
withdraw(amount+(amount/2));
winner = false;
}
balance[msg.sender]-=amount;
}


// if user is winner, withdraw winner amount
// Vulnerabilities:
// 3.1 External Calls - 3.1.1 Unchecked Call Return Value
// 3.3 Reentrancy
function withdraw(uint amount) public isWinner payable { // isWinner: Reentrancy
require(address(this).balance > amount);
msg.sender.call{value: amount}(""); // Unchecked Call Return Value, Reentrancy
}


// uses two different random algortihmns
// returns random boolean
// true if win
// false if lose
function badRandomness() private returns(bool) {
bool b = false;
if(boolean) {
b = randomWithSeed();
} else {
b = randomWithBlockHash();
}
callToggleBoolean(toggleContract);
return b;
}

// generates randomly true or false
// Vulnerability:
// 3.6 Bad Randomness - 3.6.1 With Private Seed
function randomWithSeed() private returns(bool) {
iteration++;
uint randomNumber = uint(keccak256(abi.encodePacked(seed+iteration)));
return ((randomNumber % 2) == 0);
}

// generates randomly true or false
// Vulnerability:
// 3.6 Bad Randomness - 3.6.2 With Block Hash
function randomWithBlockHash() private returns(bool) {
return (uint(blockhash(block.number-1)) % 2 == 0);
}

// if first deposit, add user to userlist
function addUserToUserlist(address payable _user) private {
if(balance[_user] == 0) {
userlist.push(_user);
}
}



// toggles the toggle boolean. For Delegate Call Vulnerability only.
// Vulnerability:
```

```
// 3.1 External Calls - 3.1.2 Delegatecall to Untrusted Callee
function callToggleBoolean(address _toggleContract) public {
_toggleContract.delegatecall(abi.encodeWithSignature("toggleBoolean(bool)",boolean));
}

/*
*
* administrative functions
*
*/

// emergency payout for all depositors (only owner)
// if depositor is in userlist, send deposited balance to user address
// Vulnerabilities:
// 3.2 Denial of Service - 3.2.1 DoS With Unexpected Revert
// - 3.2.2 DoS With Block Gas Limit
function payOutAll() public isOwner {
uint256 i = 0;
while (i < userlist.length && gasleft() > 200000) {
require(userlist[i].send(balance[userlist[i]]-20000));
}
}

// set new owner
// Vulnerability:
// 3.4 Access Control - 3.4.1 Unprotected Owner Setter
function setNewOwner(address payable _newOwner) public {
owner = _newOwner;
}

// transfer the contract balance to msg.sender
// Vulnerability:
// 3.4 Access Control - 3.4.2 Unprotected Ether Withdrawl
function getContractBalance() public {
owner.transfer(address(this).balance);
}

// destroys the smart contract
// sends the balance to the msg.sender
// Vulnerability:
// 3.4 Access Control - 3.4.3 Unprotected Selfdestruct
function destroyContract() public {
selfdestruct(owner);
}

}

// Toggle contract, only to use delegatecall
```

LOW

SWC-134

Call with hardcoded gas amount.

The highlighted function call forwards a fixed amount of gas. This is discouraged as the gas cost of EVM instructions may change in the future, which could break this contract's assumptions. If this was done to prevent reentrancy attacks, consider alternative methods such as the checks-effects-interactions pattern or reentrancy locks instead.

Source file

`browser/allVuln/BadBet.sol`

Locations

```
uint256 i = 0;
while (i < userlist.length && gasleft() > 200000) {
require(userlist[i].send(balance[userlist[i]]-20000));
}
}
```

LOW

SWC-134

Call with hardcoded gas amount.

The highlighted function call forwards a fixed amount of gas. This is discouraged as the gas cost of EVM instructions may change in the future, which could break this contract's assumptions. If this was done to prevent reentrancy attacks, consider alternative methods such as the checks-effects-interactions pattern or reentrancy locks instead.

Source file

`browser/allVuln/BadBet.sol`

Locations

```
// 3.4 Access Control - 3.4.2 Unprotected Ether Withdrawl
function getContractBalance() public {
owner.transfer(address(this).balance);
}
```