Interim Report

Final Year Project 2017-2018, Department of Computer Science, The University of Hong Kong

Topic: Blockchain research and implementation (fyp17064)

Name: NG King Pui

UID: 3035178820

Date of Submission: 21/1/2018

Abstract

Blockchain is a hot topic in the technology world. Bitcoin is one of the typical examples of applications of blockchain. The principle behind blockchain is to chain up blocks of data and let everyone have a copy of the chain, a.k.a. distributed ledger system. The blocks of data are chained up with hash, which is mathematically-proven secure. Once a single block changes a bit, the whole chain looks different. When a new block is going to be appended, everyone has to validate the chain by comparing with their own copy. Unauthorized data entries are then revoked by the crowd. Also, the data are transparent to everyone as they all have a copy. With these two features, blockchain breaks the gap of trust, which allows blockchain to play an important role in many different areas, especially monetary transaction. Besides Bitcoin, smart contracts are another application of blockchain. Smart contracts will execute automatically once the condition set is met, preventing breach of contracts. However, there are still loopholes in smart contracts. It may result in millions of monetary losses. In this project, the security holes of smart contracts on Ethereum will be studied and a tool of checking smart contracts and detecting such holes is expected to be a final deliverable.

Table of Content

1. Background

Blockchain is the backbone of Bitcoin, which is now gaining more and more attention in the world. The principle behind blockchain is to chain up blocks of data and let everyone have a copy of the chain, a.k.a. distributed ledger system. The blocks of data are chained up with hash, which is mathematically-proven secure and used in many other security measures. Every block will have its own unique hash value, which is generated based on its own data and the hash value of the last block. That is the reason why it is called blockchain, from chaining up blocks of data with hash. Therefore, if someone try to manipulate one block of data, e.g. change his own account balance, to fake others, his own copy of the chain will be different with others in terms of has value. Others can then tell his cop is not valid hence revoking that block. With the power of the crowd, it is difficult to create fake record in blockchain. Also as everyone has a copy, every transaction record is transparent to everyone. It then prevents under-the-table deals. With these 2 features, blockchain breaks the gap of trust. It can then be utilized in many areas, especially monetary transaction.

Smart contract is one of the typical examples of application of blockchain. Built on top of blockchain, smart contracts share the feature of security and transparency. Besides, smart contracts allow users to run their own script to make the contracts self-enforcing and self-executing. With the scripts, the application of smart contracts can be much wider across different industries. Apart from that, smart contracts reduce the cost and increase the efficiency significantly when compared with traditional contracts. When creating a traditional contract, legal consultancy is usually required to fit the interests of both sides and the regulations. This process will induce huge cost, in terms of both money and time. But with smart contracts, the consultancy cost can be eliminated. Additionally, based on the self-executing feature of smart contracts, the contract can become effective once the condition set is met.

For example, there is a contract between A and B, which is about A has to pay B $1 million to buy a house. With the traditional approach, A and B have to seek for relevant legal services in order to make the transaction. This induces a huge cost to both A and B. However, with smart contract, the script can check whether A has enough balance to pay B; and B really has the ownership of the house. Once both conditions above are met, the contract will

execute automatically. Also this contract is appended in a blockchain, which can be served as a proof of the fact that A owns the house after this transaction.

With such features, smart contract is expected to be more commonly used in the future. However, smart contract still has some flaws that may hinder its application. In this project, the focus will be on its security loopholes as security is the first concern of every digital transaction.

2. Methodology

In this section, the platform chosen Ethereum and the language of smart contracts Solidity are discussed. Moreover, the framework Geth and Web3.js, which are the major modules of developing the checking tool, are discussed and their roles in the tool are explained.

2.1 Ethereum

Ethereum is the chosen platform of development in this project. Ethereum is open-source blockchain platform. It is the most prominent platform for smart contracts (Buterin, 2013). It allows users to create their own scripts written in Solidity, a contract-oriented programming language for writing smart contracts. It is designed to create smart contracts on Ethereum. However, Solidity is also considered as one of the reasons why the implementation of smart contracts particularly prone to errors in Ethereum (Atezi, Bartoletti, Cimoli, 2017). In order to investigate these errors caused by Solidity, this is chosen as the language to use.

Ethereum is a blockchain platform. To create incentive for others to compute the result together, a token Ether is created to pay for those who have computed. For example, to validate a block of data about A buying a house from B, the transaction need to be validated by the crowd. To pay for the computing power, both A and B have to pay in Ether, or namely gas, for their effort. Currently one Ether token equals around US$300.

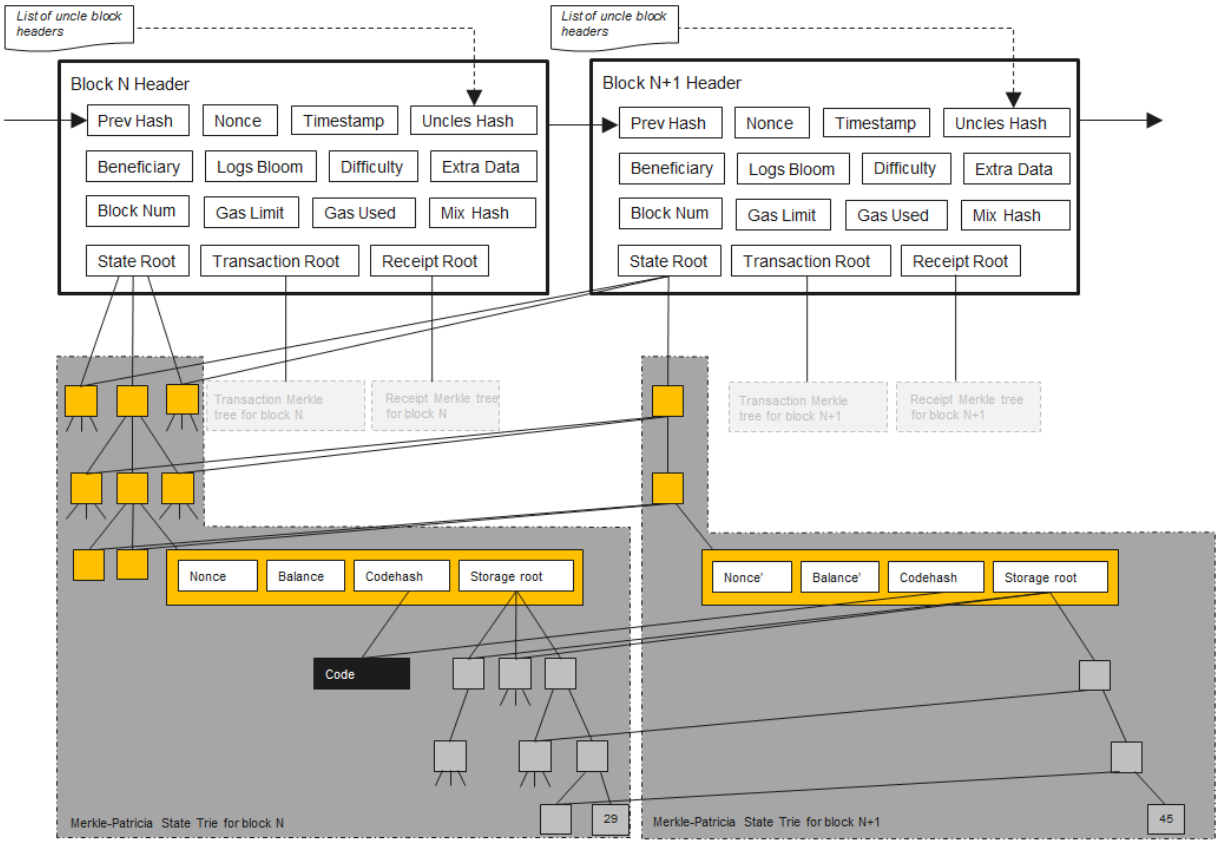

Figure 1: Architecture of Ethereum

Figure 1 shows the architecture of Ethereum. The header consists of many data which help identifying blocks. The PrevHash will help when chasing back the chain of data. This part makes Ethereum adopt the structure of blockchain, which make them share the same features mentioned in Background section. The state is the data stored in the block. These data can be balance or other data as shown in Fig.1.

2.2 Solidity

Solidity is a programming language designed for writing smart contracts. Fig. 2 shows how does a simple wallet object is created with Solidity. Owner contains the unique address of the owner of the wallet. It initializes the owner as who creates this object. Also the Pay function takes in the amount to send and the address of recipient. Before the owner actually pays, the function will first check does the owner call this function and if the owner has enough balance to pay that amount. If these conditions are all met, the function will deduct the amount from the owner and call recipient.send(amount) to increase the balance of recipient. It is only a simple example. More functions can be added if required.

```
contract AWallet{
    address owner;
    mapping (address => uint) public outflow;

    function AWallet(){ owner = msg.sender; }

    function pay(uint amount, address recipient) returns (bool){
        if (msg.sender != owner ||  msg.value != 0) throw;
        if (amount > this.balance) return false;
        outflow[recipient] += amount;
        if (!recipient.send(amount)) throw;
        return true;
    }
}
```

Figure 2: Simple demo code of a wallet in Solidity (Atezi, Bartoletti, Cimoli, 2017)

2.3 Geth

Besides researching on the potential attacks on Ethereum smart contracts, another key deliverable of this project is a tool checking smart contracts whether the identified security loopholes exist in those contracts. In order to check those contracts, a private blockchain is needed. With a private blockchain, a isolated testing environment is created so that no real monetary transaction is involved. Additionally, all the factors in a private testing blockchain are under my control. In a public blockchain, due to the nature of decentralized system, it is impossible to control the behavior of every single user. Therefore, some extreme cases cannot be tested. On the other hand, virtual users can be created to simulate different scenario, including extreme cases. Due to these two reasons, a private blockchain is a desired testing environment of Ethereum smart contracts.

In order to create a private blockchain to simulate attacks on Ethereum smart contracts, Geth, or Go-Ethereum, is used to create such a isolated environment. Geth is the official Ethereum implementation written in a programming language Go. It is widely used to communicate with the Ethereum network. Its functionalities include managing different accounts, mining cryptocurrency Ether and executing smart contracts. In this project, the functionalities to be used are mainly managing different dummy accounts and executing smart contracts under testing.

In order to create a private chain without communicating with the main public Ethereum chain, all the nodes in the private chain should not be connected to the public chain and be discovered by other users. A network ID different from the main network is needed to make sure that the nodes out of our testing environment are not connected to our private chain. The network ID of the main chain is 1. Therefore, if the network ID of the private chain is not 1, the nodes in our private chain cannot connect to the main chain where many real monetary transactions are made.

Moreover, the maximum number of peer can be set to zero to further confirm that no other peer is connected to the private network. The corresponding command is:

```
--maxpeers 0
```

0 here is the parameter indicating the desired maximum number of peers. As we look for an isolated testing environment, 0 is the desired number of peers.

2.4 Web3.js

With Geth, an isolated testing environment is created. The next step is to deploy the smart contracts. Web3.js is the library to use. Web3.js is JavaScript API (application programming interface) which can compile smart contracts written Solidity and execute them in the private chain.

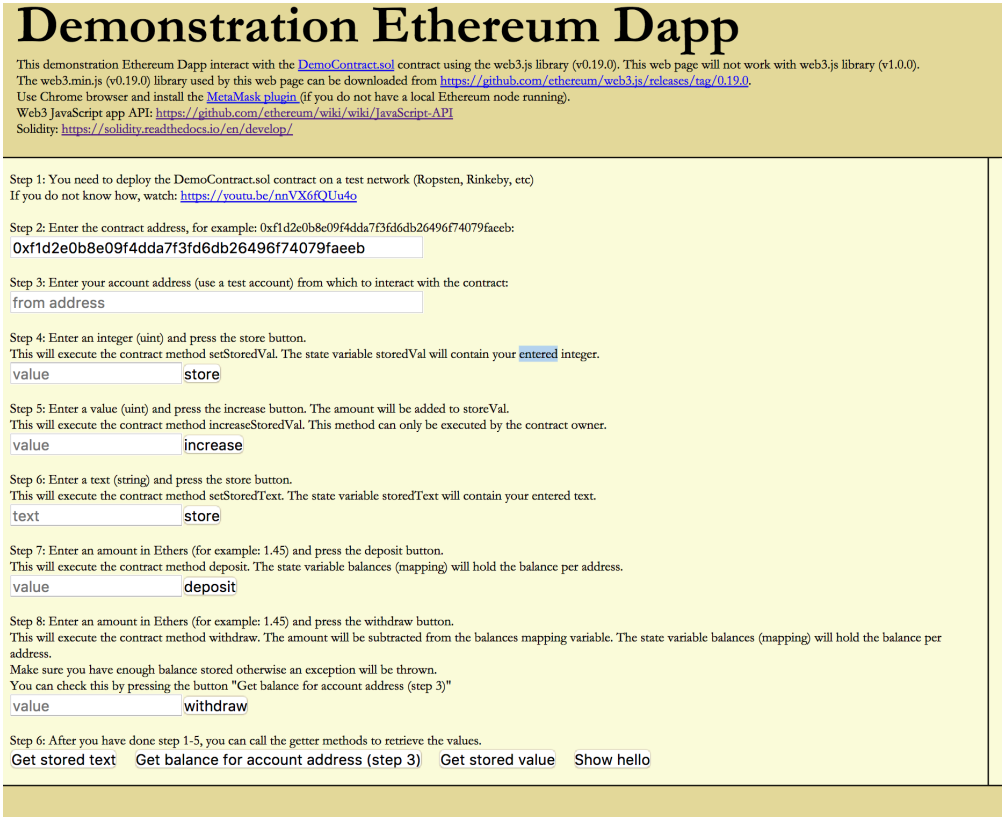

Figure 3: A screenshot of web-based application in Web3.js by mobilefish.com

Additionally, as it is a module in Node.js, another framework building web application, this tool can be launched as a web application. It makes the tool more user-friendly as no command input is needed to use tool. The entry barrier of using this tool is lowered by a simple web interface. The above figure is a screenshot of a web-application of Ethereum. It is easier to use compared with traditional command line tool.

To compile a Solidity smart contract, the command to use is:

```
web3.eth.compile.solidity(sourceString [, callback])
```

sourceString is the contract itself in string form. The callback is the function called after the contract is successfully compiled. After that an instance of contract object is created to execute the contract. The contract can then be executed and be tested under different potential attacks.

2.5 Summary of checking tool

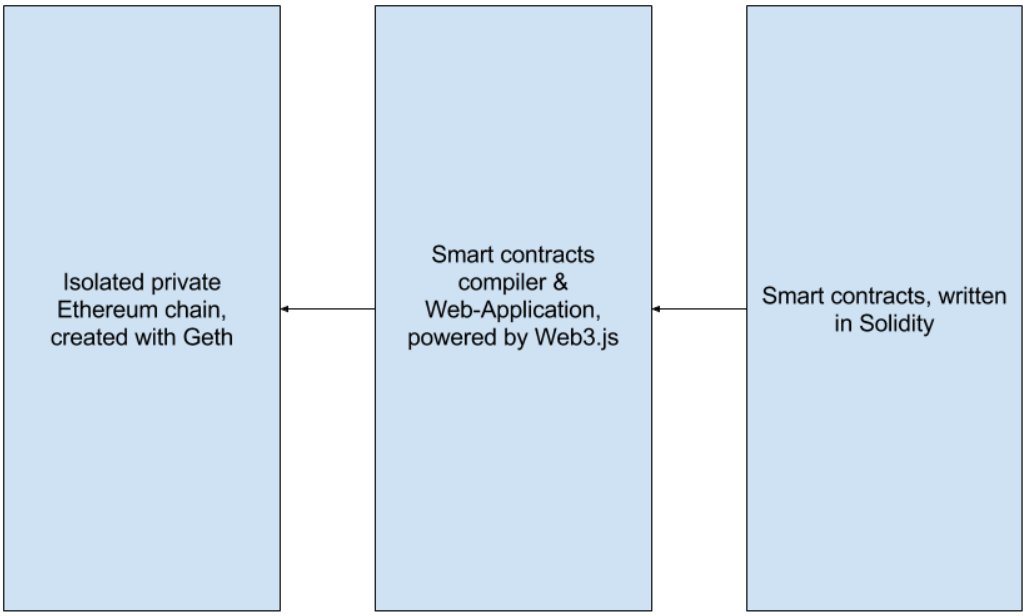

Figure 4: Architecture and logic flow of the checking tool

To conclude, the architecture of the checking tool is shown as above. The smart contracts in Solidity are the input of the system. They will be compiled with Web3.js and be executed on an isolated private chain created with Geth. The identified security loopholes are then tested under that environment. The tool is launched as a web-based application powered by Web3.js and Node.js to make the tool more user-friendly.

3. Current Status

3.1 Research on security loopholes

At the current moment, the major work done is research on the topic, including the principle of blockchain and smart contracts, the skills to write smart contracts on Ethereum with Solidity and the existing and identified security loopholes of smart contracts. The principle of blockchain is mentioned in Background section and introduction of Ethereum is included in Methodology section. Therefore, this section will focus on the findings on the identified security loopholes of smart contracts.

In "A Survey on Attacks on Ethereum smart contracts, it provides a list of known vulnerability of Ethereum Smart Contracts.

3.1.1 Wrong Typecast

```
function sweepCommission(uint amount)  {
  owner.send(amount);
}
```

Figure 5; A sample code of wrong typecast

First, wrong typecast to variables is one of the vulnerabilities identified. Solidity compiler dons not check whether a function takes in a correct type of variable. A sample code is shown in Figure 5. The function sweepCommission takes in a parameter with typr uint, which stands for unsigned integer, like 20. If parameters with other types, like string or decimal number, error should be prompted to notify the developer. However in Solidity, no error will be returned and the developer cannot notice such an error as usually the other language compile will check the type of variables. So the developer may think that the contract is correctly executed. It could bring chaining effect if the number of parties involved is huge. The whole system may fall.

3.1.2 Immutable bugs

Next, the bugs are immutable once it is on the blockchain. Based on the mechanism of blockchain, it is nearly impossible to change a single block of data once it is appended. That

means once a bug is on the blockchain, it is difficult to remove it. Once the bugs or vulnerability stack up, the system may fall.

3.1.3 Unpredictable States

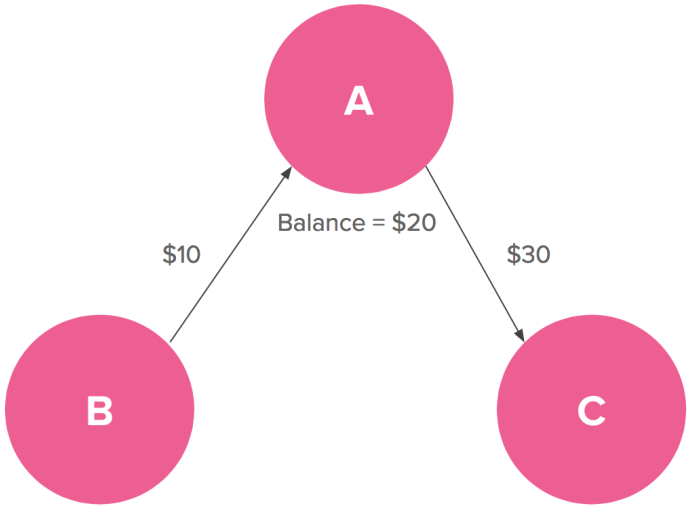

Figure 6: An example of unpredictable state scenario

Finally, unpredictable state is another problem to be considered. As mentioned in Methodology section, state is the data contained in a block, e.g. balance. However, sometimes the smart contracts cannot be executed immediately. An example scenario is shown in above figure. A has 20 dollars as balance. B sends 10 dollars to A and A pays C 30 dollars. These transactions will be valid if A first collects money from B followed by paying C. However, it is not guaranteed that A collects money first. The time taken for checking processes of two different transactions and network speed of different users are some potential factors affecting the order of executing those contracts.

The above are some of the vulnerabilities of smart contracts. There are more in "A survey on attacks on Ethereum smart contracts" not studied and remained not identified. The research will mainly focus on the holes identified in "A survey on attacks on Ethereum smart contracts". If some other holes are identified during the research in later stage, they will be included in this project as well.

3.2 Developing the checking tool

Besides the research on attacks on Ethereum smart contracts, the research on how to develop a checking is necessary. The working principle of the checking tool is discussed in the Methodology part. Therefore in this section, only the current progress of the development is reported.

At this stage, the research of how to develop the tool is done. The research result can refer to Geth and Web3.js in Methodology.

Besides the research, the development phase has begun. A private chain is set up in my own Wprivate chain. I can now perform simulated transactions under my control. However some parts, like the designing scenarios of different attacks These parts will be further discussed in the Upcoming Milestone part.

3.2.1 Launching a private chain

As mentioned above, the testing environment will be a private chain. At this stage, a private chain is set up in my own machine using Geth. After testing, simple Ethereum smart contracts can be executed in that chain. Additionally, as a private chain in a completely closed environment is used, all the user activities, like transactions, are all under control.

3.2.2 Launching a prototype of the web-based tool

Besides setting up the chain, a web-based tool is required to provide the best user experience.

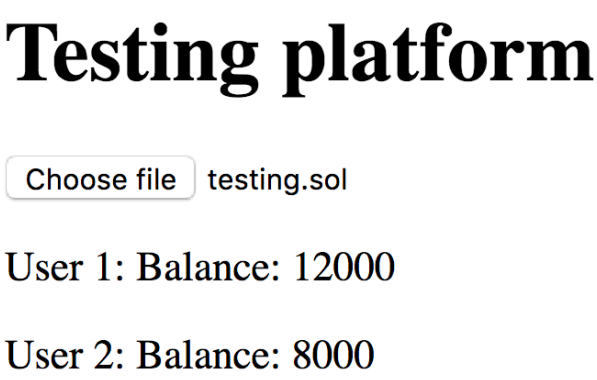

Figure 7: A screenshot of prototype of the web tool

The above is a screen shot of the prototype of the web tool. It allows the users to upload the smart contract file with extension .sol. The smart contract is then passed to the private chain and executed. Then the result, in this case the new balance, is shown. At this stage, the interface is not designed and implemented. Moreover only the basic functions, like uploading and running a smart contract, are implemented in this prototype. The other functions, like simulating different attacks, are to be implemented in later stage. These works remaining will be further discussed in Upcoming Milestones section.

4. Problems Encountered

There are several problems encountered in this project. This section will discuss those problems in both the research and development.

4.1 Insufficient security loopholes

Smart contract is still quite new to the society. The reliable research in this field is still not enough. The main reference I take in this project is “A Survey of Attacks on Ethereum Smart contracts”. However, there are some loopholes not covered. If some reports on the Internet are also considered, more research will be needed to verify the identified loophole. Due to time constraint of this project, the research will be stopped by Dec 2017. It is a pity that some unverified and potential security holes are not covered in this project.

4.2 Computing Resources of the tool

The checking tool requires an isolated environment for testing. However, the computing power and storage required to run a private chain is not negligible. Moreover, if a private chain is created for every test, the storage will be soon full. Therefore, a way to solve this problem is needed for making the tool scalable for future use. A possible approach is to clean up the chain when the test ends. But this is to be tested.

4.3 Deploying the tool to the cloud

To make the tool available to public, the ideal approach is to deploy the tool to cloud. However, as mentioned above in 4.2, the storage and computing power are major concerns. Since it is currently in development phase, the development and testing will be done in local machines to reduce development costs like running a machine on cloud. However, the tool can be migrated to cloud when conducting pressure test on the tool.

5. Upcoming Milestones

At this stage, the basic research of the topic is done. Afterwards, the project will be in development phase, which is designing, developing and testing the tool of checking a smart contract.

The design phase will take about half a month. The design of mechanism is done when the research is completed as the way to check a smart contract based on the security holes found is research is known. Therefore this phase will mainly focus on designing the interface of the tool and merging the check result with the interface.

After the design phase, the development work will be the next phase. It is also the most important part in the project. Based on the findings in the research phase, the tool will take in different smart contracts and put them into different attacks scenarios. In this phase the focus will be on designing and developing different attacks scenarios. As the implementation of these scenarios are currently under progress, they are not included in the prototype of the web tool.

Finally, the tool will be under testing. Sample dummy smart contracts written by others and me will be processed by the tool. Based on the test result, the bugs can be identified and fixed later. The process will be repeated until the tool is bug-free. Also some other users will be invited to use the tool and asked about the experience and check result of the tool. These comments can improve the tool. Also the other deliverables like final report and poster should be ready by the end of testing phase.

The schedule of expected upcoming milestones is:

Time	Project Status
Feb – Mar 2018	Developing the tool and different attacks scenarios
Mar - Apr 2018	Testing and fixing bugs of the tool, Writing the final report

Figure 8: the schedule of expected upcoming milestones

6. Limitation

During the research of the project, there are some challenges and limitations of the project.

6.1 Uncovered security loopholes

First, there may be other security loopholes of smart contracts that are not included in this tool. Although the essay “A Survey Attacks on Ethereum Smart Contracts” provides various identified attacks on smart contracts, it is foreseeable that there will be new attacks in the future. As a checking tool of smart contracts, it should identify as many security holes as possible to prevent false-positive situation. Therefore in the future this tool needs regular update to deal with new security loopholes. However, due to time constraint, the development work will take up most of the time. So not much research on new attacks can be done.

6.2 Cross platform checking

Next, this tool can only detect security holes within the same blockchain. On Ethereum, all the transaction is done with the currency Ether, a cryptocurrency produced in Ethereum computation. However, to make smart contracts practical, the smart contracts must get data from other sources to validate a contract. Like in the example of A buying a house from B, the contract must see if B has the ownership of the house from other source like the government if the data required is not on the same blockchain. So API (application programming interface) must be used for data exchange. However, communication of different APIs or the API itself may create security holes which cannot be checked with this tool.

7. Conclusion

With the rise of blockchain and the convenience brought by smart contracts, the coverage of smart contracts is expected to grow rapidly. That is the reason why security loopholes of smart contracts are concerned. In this project, the loopholes are studied and a tool checking the vulnerability of smart contracts from those security holes will be developed as the final deliverable.

8. Reference

1. Atzei, N, Bartoletti, M, and Cimol, T.i: A Survey of Attacks on Ethereum Smart contracts, Universita degli Studi di Cagliari, Cagliari, Italy (2017)
2. Buterin, V.: Ethereum: a next generation smart contract and decentralized application platform. https://github.com/ethereum/wiki/wiki/White-Paper (2013)
3. Creating a Private Chain/Testnet, https://souptacular.gitbooks.io/ethereum-tutorials-and-tips-by-hudson/content/private-chain.html (2017)
4. Geth – Ethereum Wiki, https://github.com/ethereum/go-ethereum/wiki/geth (2017)
5. Javascript API – Ethereum Wiki, https://github.com/ethereum/wiki/wiki/JavaScript-API#web3ethcontract (2017)
6. Mobilefish.com, Demonstration Ethereum Dapp, https://www.mobilefish.com/download/ethereum/DemoDapp.html (2017)

9. Table of Figures