A Security Case Study for Blockchain Games

Tian Min
School of Science and Engineering
The Chinese University of Hong Kong, Shenzhen
Shenzhen, China
tianmin@link.cuhk.edu.cn

Wei Cai
School of Science and Engineering
The Chinese University of Hong Kong, Shenzhen
Shenzhen, China
caiwei@cuhk.edu.cn

***Abstract*—Blockchain gaming is an emerging entertainment paradigm. However, blockchain games are still suffering from security issues, due to the immature blockchain technologies and its unsophisticated developers. In this work, we analyzed the blockchain game architecture and reveal the possible penetration methods of cracking. We scanned more than 600 commercial blockchain games to summarize a security overview from the perspective of the web server and smart contract, respectively. We also conducted three case studies for blockchain games to show detailed vulnerability detection.**

***Index Terms*—Blockchain, Game, Architecture, Security**

I. Introduction

With the popularity of cryptocurrencies, e.g. BitCoin [1], the blockchain technology [2] is now recognized as the foundation of next-generation digital economy. However, the research community is looking forward to unleashing the full potential of blockchain for other businesses. To respond to the demand, Ethereum [3] brought smart contracts [4] to announce the start of Blockchain 2.0 era. A smart contract, written as immutable blockchain transactions, are transparent and auditable programs that can be automatically executed without any centralized control. With the support of the smart contract, decentralized applications (DApps) [5] became possible.

Blockchain games become one of the most active DApp in the ecosystem. According to the recent survey work [6], over 50% of network traffic in Ethereum and EOS[1] platforms are from blockchain game players. The phenomenon can be explained from different perspectives. First, the non-fungible nature of blockchain data and the transparency of the smart contracts enable game developers to better prove the rule transparency, guarantee the ownership of the virtual asset, enable assets reusability, and encourage user-generated contents. Second, the blockchain game builds the whole ecosystem in the virtual world, which avoids a lot of realistic constraints commonly exist in other DApps, including the Internet of Things and source tracking. According to the above advantages, the blockchain game is considered an emerging trend. The industry has started its exploration on this topic by integrating traditional games with blockchain systems. Until January 2019 on Ethereum[2], games contributed 1,113,516 transactions, which is 56.8% of the total, and carried a transaction volume of 198,457 ETH, which is equal to 22 million dollars. In the same month, games have 42,210 active users, which is 24.3% of all. From these statistic data, we can tell that blockchain games have already become an important component of DApps and have held a considerable market capitalization.

However, DApps are facing severe security issues. According to PeckShield 2018 annual report[3], the economic losses caused by blockchain security in 2018 amounted to 2.238 billion dollars, which is 253% of the 2017's. The blockchain security issues are mainly concentrated on the application layer and the contract layer, with 64 and 58 incidents respectively. The economic losses caused by these layers accounted for 98.87% of all. To be more specific, on Ethereum, there were 54 security incidents. Most of them happened because of the issues from the exchange trading system, wallet website security, smart contracts vulnerabilities, and blockchain design defects. On EOS, There were 49 security incidents. Most of the attacks directly targeted on EOS smart contracts. In EOS smart contracts, the vulnerabilities can be easily reproduced in others.

In this work, we use the blockchain game, the most popular public blockchain application, to conduct the security study in DApps. The remainder of this paper is organized as follows: we briefly introduce the previous works on blockchain technology and security in Section II. Then, we illustrate the system architecture for the blockchain games and provide possible attack methods in Section III. Next, an statistic of blockchain game security will be presented in Section IV. Afterward, we conduct security case studies in Section V to do practical demonstrations. Section VI concludes the article and envision the future of blockchain games.

II. Related Work

A. Blockchain and Games

Blockchain is a decentralized database system with the characteristic of transparency, immutability and traceability. Different from the traditional database, public blockchain is a system maintained by the public, which can be accessed and verified by anyone around the world. To be more specific, it is a series of continuously growing blocks. Each block contains a cryptographic hash of the previous block, a timestamp, and its conveyed data [2]. Due to the existence of the cryptographic

[1]https://eos.io/

[2]https://www.stateoftheDApps.com/stats

[3]https://www.huoxing24.com/newsdetail/2019012813264825241l.html

978-1-7281-2404-9/19/$31.00 ©2019 IEEE

hash, blockchains are immutable. If a block on chain is modified, all the descendants of this block should be regenerated with new hash value. The blockchain data structure, together with the peer-to-peer (P2P) system and the proof-of-work (PoW) [7] consensus model, forms the solid foundation for DApps.

According to this definition, we only discuss blockchain games with decentralized nature in this paper. Therefore, those blockchain games which follow the centralized and close source principle will not be investigated. For example, the CryptoRabits[4] developed by Xiaomi Inc. will be out of the scope of our work, since it was published without any public open source code or white paper. In addition, according to CryptoRabits' user agreement[5]: 1. Users are not allowed to trade the currency in the game. 2. The operator has the right of making or adjusting the rules for the game. 3. If the user violates the agreement, the operator has the right to stop providing services to him immediately without his consent. These terms are completely contrary to the spirit of the blockchain. Players' ownership of virtual properties cannot be guaranteed.

B. Smart Contract Vulnerability

Initially proposed by Nick Szabo [8] in 1997, a smart contract is a protocol that can automatically verify and process the content of the contract. Thanks to the features of blockchain systems, the smart contract can ensure the code execution without the third-parties. Different blockchain platform may have different regulations on smart contracts programming. Nevertheless, all smart contracts have structures like object-oriented programs. On Ethereum, smart contracts are programs wrote in a JavaScript-like language called Solidity[6]. Each contract is like a class, which contains variables and methods. Contracts can also invoke each other to implement complicated tasks. Following is a simple example of a Solidity smart contract.

```
pragma solidity >=0.4.0 <0.7.0;

contract SimpleStorage {

    uint storedData;

    function set(uint x) public {
        storedData = x;
    }

    function get() public view returns (uint) {
        return storedData;
    }
}
```

However, a smart contract based program is susceptible. A study [9] pointed out that there are 34,200 contracts marked as vulnerable in a million samples. Some of the most essential reasons for the vulnerabilities are platform and the contract programming language's design defects. Take Ethereum and Solidity as an example. Most of the currently known vulnerabilities about the smart contracts are related to the fallback function, which is an unnamed function triggered when an external caller is sent ETH, the Ethereum token, to the contract, or calls a function that does not defined. When *fallback()* includes an external function or has potential vulnerabilities, the attackers could hijack the invoked contract, and force it to execute.

C. Smart Contract Audition Tools

Code auditing is not a new concept. It is an integral part of the defensive programming paradigm, which attempts to reduce errors before the software is released. However, Solidity is a high-level programming language with many potentially vulnerable functions. Due to the unchangeability of the blockchain, updating patches after deployment becomes especially troublesome. Hence, the smart contract audition becomes of paramount importance. As system developers and operators are gradually aware of the importance of blockchain security, more and more auditing tools have emerged. Different tools may have different advantages including automation degree, accuracy and efficiency. Audition tools detect the vulnerabilities in three main ways: 1) Code Feature Matching: Auditor collects and extracts malicious code's feature, and do semantic matching on other source code. 2) Formal Verification: Formal Verification is mathematical access to prove a system's completeness. Auditor specifies every possible input and exhaustive every situation that might happen. 3) Symbolic Execution: Auditor generate a control flow graph by contract's logic units (like determining statements). From this logic graph, The auditor can traverse all codes paths to reveal how the variables passing through the program in order to detect logical design flaws.

III. Blockchain Game Architecture

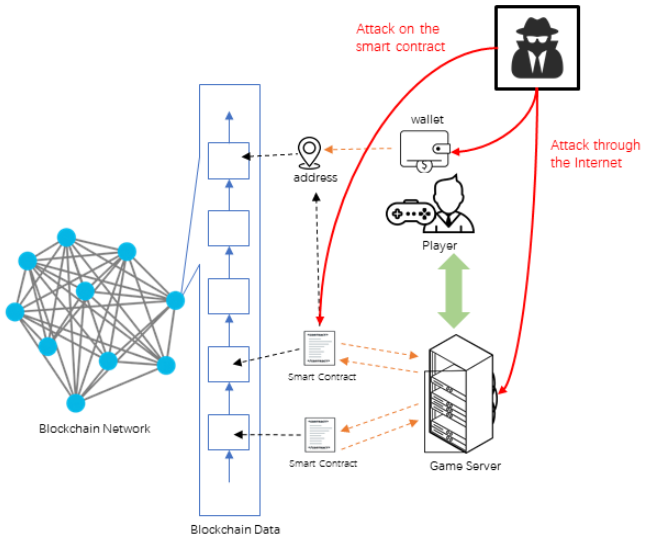

Fig. 1: Architecture for Blockchain Games

[4]https://jiamitu.mi.com/home
[5]https://jiamitu.mi.com/protocol
[6]https://solidity.readthedocs.io/en/develop/index.html

Fig. 1 illustrates the architecture of conventional blockchain game. Different from the traditional games, blockchain game players need to register an address in the corresponding blockchain platform before starting their gaming sessions. This blockchain address, accessed by a wallet program, will serve as a unique identity and a destination of the virtual assets for its corresponding player. On the other hand, the game server should offload some core functions, e.g. the ones which manipulate the players' virtual assets or critical game rules, to the blockchain as smart contracts in order to keep them transparent and immutable.

The server plays an important role in this architecture. In addition to providing game service, it acts as a cache and indexing engine for smart contracts. Although the ultimate source of information is from the blockchain, clients rely on the server's searching and verifying capabilities of the data returned from the blockchain. Moreover, writing data to PoW blockchains, e.g. Ethereum, is expensive. The server still needs to store most of the data and only store a hash on the chain for verification. The server of a blockchain game interact with the Ethereum blockchain via web3.js, which is a collection of libraries which allow developers to interact with a local or remote Ethereum node, using an HTTP, WebSocket or IPC connection[7]. According to the architecture, there are mainly two methodologies of attacking a blockchain game:

A. Web

The focal point of penetrating a blockchain application is accessing the digital assets. Hence, the wallet becomes of great importance. The private key to the wallet shall be the optimal target of the attackers. Once the attacker obtains the private key, he could easily transfer assets away if there is no a two-Step verification. The secondary target shall be the game server since it is where the susceptible information may be stored. These information could help the attacker with further penetration. The ideology of penetrating a blockchain wallet or a DApp server may not has a great difference from traditional cyber attack.

B. Smart Contract

Since all contracts are open source, the attackers can identify the vulnerable spots directly by analyzing the source code. Although most smart contracts were compiled into bytecodes before deployment, there are various tools that help reverse engineering. Smart contract vulnerabilities may exist in many different layers, including Solidity language, execution logic, and Ethereum Virtual Machine (EVM) design. in TABLE: I, Nicola Atzei [10] summarized a taxonomy of smart contract vulnerability. It shows that the vulnerable spots can be found through the entire work-flow of smart contract execution.

IV. Statistics of Security Risks

We selected 610 games listed on the State-of-the-DApps[8] and collected URLs and smart contract codes for analysis.

Level	Cause of vulnerability
Solidity	Call to the unknown
	Gasless send
	Exception disorders
	Type casts
	Re-entrancy
	Keeping secrets
EVM	Immutable bugs
	Ether lost in transfer
	Stack size limit
Blockchain	Unpredictable state
	Generating randomness
	Time constraints

TABLE I: Smart Contract Vulnerability

Nikto2[9], a web scanner, was employed to detect the vulnerabilities on the server and the web application, and Mythril[10] was used to detect vulnerabilities in their smart contracts.

A. Web Overview

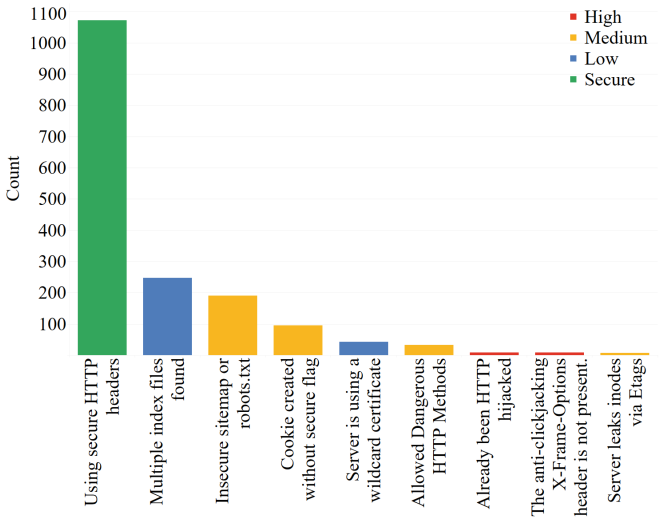

Fig. 2: Web App Issue Count and Severity

Nikto2 is not an aggressive scanning tool. It mainly detects the misconfiguration on the web servers and the protocols. In 610 sites, more than 1,700 URLs, mainly 8 kinds of issues were detected. The result in Fig. 2 shows that 62.91% of the samples are using secure HTTP headers like "Strict-Transport-Security" or "X-XSS-Protection" to prevent being attacked. 16.96% of them are under low-level risk, having multiple index files or using a wildcard certificate, which means if one server or sub-domain is compromised, others may also be exposed to the danger. 19.41% of the samples are at medium level risk, they allow some risky HTTP methods, create cookies without a secure flag, or leaving sitemaps may have potential

[7]https://github.com/ethereum/wiki/wiki/JavaScript-API
[8]https://www.stateoftheDApps.com/
[9]https://cirt.net/Nikto2
[10]https://github.com/ConsenSys/mythril-classic

vulnerabilities. Rest 0.72% of them have already been hijacked or haven't opened anti-clickjacking X-Frame-Options.

From the statistics, we can draw a conclusion that there has been systematic security development frameworks or toolkit for the developers when they started to build their own web server, thanks to the long-term development of web security. Followings we briefly introduce some common potential attacks which could be caused by these vulnerabilities:

Cookie Replay - caused by Cookie created without secure flag: The secure flag is an option that can be set by the application server when sending a new cookie to the user within an HTTP Response. This flag can prevent cookies from being observed by unauthorized parties due to the transmission of a the cookie in clear text. This vulnerability could allow the attacker impersonate the user as long as the cookie remains valid.

Injection - caused by Allow Dangerous HTTP Methods: Almost any data source can be an injection carrier, including environment variables, parameters, and external or internal web services. Blockchain games usually have frequent interaction with players, including lots of input box and complicated URL routes.

XSS - caused by X-Frame-Options not present: Cross-site scripting(XSS) is a kind of common vulnerabilities that happen on the web applications. Although most sites know to protect themselves with special filters, XSS attacks can be considered dangerous because they usually act as a springboard towards users' private keys. Secure headers like "X-Frame-Options" must be included to eliminate html script "¡iframe¿", which could lead to a clickjacking and fool players to input their passwords.

Broken Authentication: Apart from the scanning results above, directly cracking the authentication key is an important kinds of attacks. Attackers can do social engineering and use brute force to crack a wallet. Apart from weak passwords, poor session management also causes broken authentications. Especially for those sites exposing session ID in the URL, or creating token without encryption.

B. Smart Contract Overview

In 1,311 smart contracts, 12 kinds of vulnerabilities are detected as the result are shown in Fig. 3. Only 11.63% contracts are bug-free. 14.04% contracts have high-risk vulnerabilities like overflow and underflow, unprotected Ether withdrawal or unprotected self-destruct. 12.08% of them are medium risks like usage of *tx.origin* or applying multiple calls in a single transaction. Rest 62.25% of the contracts are in low-level risk, having flaws like exception state or allowing external calls.

Among all the results, "Exception State" possesses the highest proportion of 35.43%, which means that a large proportion of smart contract developers didn't handle the exceptions. The most common high severity vulnerability is "Integer Overflow", which take a proportion of 12.87%.

We list some most common attacks that could be caused by the scanning results blow. You can check more up-to-date

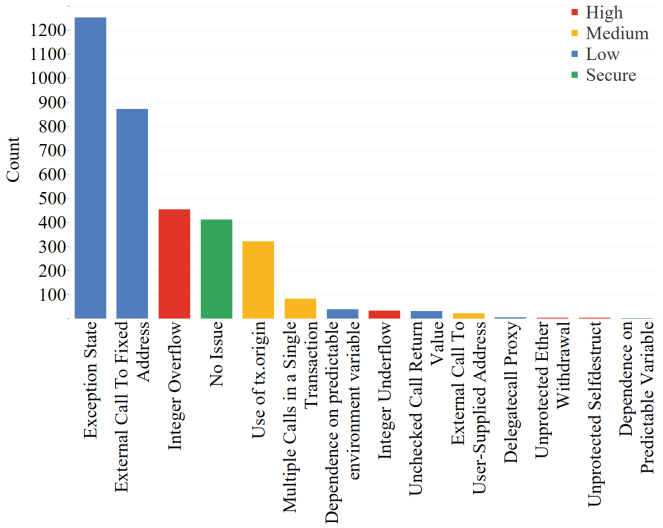

Fig. 3: Smart Contract Issue Count and Severity

known attack on the Smart Contract Weakness Classification Registry[11]:

Overflow/Underflow - caused by Integer Overflow/Underflow: Integer overflow and underflow are one of the most common vulnerabilities in the smart contract. If a UNIT256 reaches the maximum unit value(2^{256}), it will circle back to zero when it's added with another 1. Attackers can exploit this vulnerability by repeatedly invoking the function that increases value. Game data like character's attack, defense or health point could be modified frequently, blockchain game developer may put special attention on value control.

Tx Origin Attack - caused by Use of *tx.origin*: *tx.origin* is a global variable that return the address which initially invoked the smart contract. Some developer use this variable to do the authentication. For example, the attacker wants to pass an authentication in a target contract. First, he might find a way to trick the victim into transferring to the malicious middle contract. Then, the middle contract will call the target contract in its *fallback()*. So, the tx.origin address in the target contract should be the victim's address and authenticated.

Predictable Variable - caused by Dependence on predictable environment variable: Some smart contracts write functions that depend on variables like timestamp or block's height(distance from the genesis block), which are predictable. For example, when a blockchain casino generate dice points using the timestamp. The attacker can easily crack these functions and win the game.

Denial of Service (DoS): DoS is a kind of common vulnerability which closely related to the fallback function and revert mechanism. The attacker can create a dead loop by logic vulnerabilities in the contracts. Take a bidding system as an example, if the system can only set a new bid leader after refunding to the previous one, the attacker can write a

[11]https://smartcontractsecurity.github.io/SWC-registry

function that reverts any transaction in *fallback()*, in order to keep being the bid leader.

Re-entrancy: Re-entrancy is a serious issue related to calling malicious external contracts, which may take over the control flow. This kind of vulnerabilities varies in different forms: every external call can be potentially dangerous. Nowadays, blockchain games are getting more and more complex, calling external smart contracts is unavoidable. If a developer must do an external call, he should cautiously verify the contract author, and try to arrange the call after the execution of the internal code.

Apart from making malicious calls to those vulnerable contracts, attackers can also take advantages of Ethereum's logical design flaws. Most of these flaws are dilemmas. For example, Gas is of great necessity to the PoW consensus model. However, attackers can manipulate the transaction by offering an expensive Gas fee. For example, Fomo3D is a gambling game with a 24-hour countdown. 30 seconds will be added every time when a token is sold. When the countdown touches 0, the last token buyer wins the jackpot. Due to the Ethereum's Gas mechanism, the attacker can do several transactions with expensive gas to jam the mining system, so thqt he/she could keep getting the top priority of the blockchain packing queue. As a result, other buyers' transaction cannot be successfully verified and written into blocks.

V. Case Studies

In this section, we conducted case studies to demonstrate the security issues in current commercial blockchain games. The first three cases are historical accomplished attacks, including EOSFomo 3D[12], Pandemica[13] and EOSlots[14]. We analyze their vulnerabilities and methods the attackers used. The rest of three cases are scanning result analysis, including Cryptokitties[15], 0xUniverse[16] and Mythereum[17]. We will showcase their high and medium level vulnerabilities in terms of web application aspect and smart contracts aspect.

A. EOSFomo 3D

EOSFomo 3D is a Fomo3D[18]-like game based on EOS platform. The players purchase keys on different teams and the last one receive rewards from jackpot. In July 2018, EOSFomo was attacked through an overflow vulnerability. As shown in Fig. 4, the bonus displayed on the website became negative after the attack. In this incident, 60,686 EOS were stolen from ordinary users.

Vulnerability - **Overflow/Underflow**: Because the game has already been shut down. We are unable to analyze its source code. However, we can infer that the developers did not verify

Fig. 4: EOSFomo 3D's Hompage After the Attack

the results or use a secure library. As shown in Fig. 4, the overflow is triggered by a player, which means the rights management system has design flaws. The player can exploit an overflows by repeatedly calling a public function in the contract.

B. Pandemica

Pandemica is a Ethereum-based game following a simple Ponzi Scheme: players transfer ETH to the contract, and the owner randomly return 3% of the collected fund to the players at 6:00 p.m. everyday. In August 2018, ETH worth 120 thousand USD was frozen in this contract[19] shown in Fig. 5.

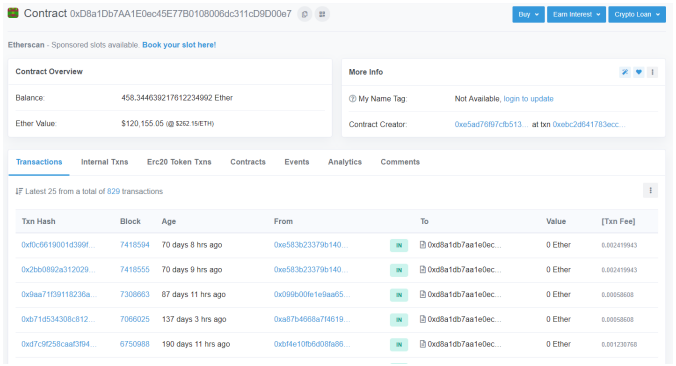

Fig. 5: Pandemica Contract on Etherscan

Vulnerability - **Gas Overflow**: The contract developer used a loop to implement the paying method to the users:

```
function Count() onlyowner {
        while (counter>0) {
            Tx[counter].txuser.send((Tx[counter].
                txvalue/100)*3);
            counter-=1;
        }
    }
```

The number of loops is determined by the number of participants. However, the amount of Gas that can be consumed in each block has a upper limit of 8,000,000 Gas.The *counter* variable will grow as the increase of players. When the value

[12]https://eosfo.io
[13]https://pandemica.online/
[14]https://www.eoslots.com/
[15]https://www.cryptokitties.co/
[16]https://0xuniverse.com/
[17]https://www.mythereum.io/
[18]https://exitscam.me

[19]https://etherscan.io/address/0xd8a1db7aa1e0ec45e77b0108006dc311cd9d00e7

of *counter* reaches a certain threshold, Gas fee executing the *Count()* function will exceed 8,000,000. This fund can only unfreeze when the Ethereum raise the upper bound of the Gas fee in the future.

C. EOSlots

EOSlots is a slot machine game on EOS platform as illustrated in Fig. 6. The developers claim it to be a fee-less and trust-less game where players can place bets in EOS at zero cost and have absolute certainty the game is fair, since the player's funds go directly into a smart contract without the need for a middleman.

Fig. 6: Screenshot of EOSlots

As shown in Fig. 7, on April 3rd 2019, the attacker cracked the pseudo random number of EOSlots[20] and kept winning the game illegally. The attacker got ten times of the value he bet.

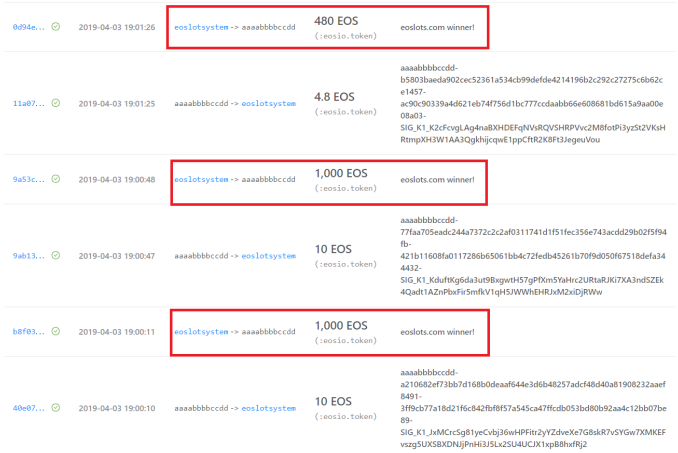

Fig. 7: Attacker aaaabbbbccdd Kept Winning the Game

Vulnerability - **Predictable Variable**: Currently, Ethereum and EOS officials didn't provide a standardized random number interface, which causes a negative impact on blockchain games, especially the lottery games. In order to implement a random number generator, developers have to write their own functions, which often use the current block information as generator parameters. However, the attackers can generate the exact same value from the same parameter. Attackers can deploy a testing contract to keep generating random numbers and join the game after they got the numbers to look like the correct results.

D. Cryptokitties

CryptoKitties is a blockchain game developed by Axiom Zen that allows players to purchase, collect, breed and sell various types of virtual cats. It is one of the earliest and most successful blockchain games on Ethereum.

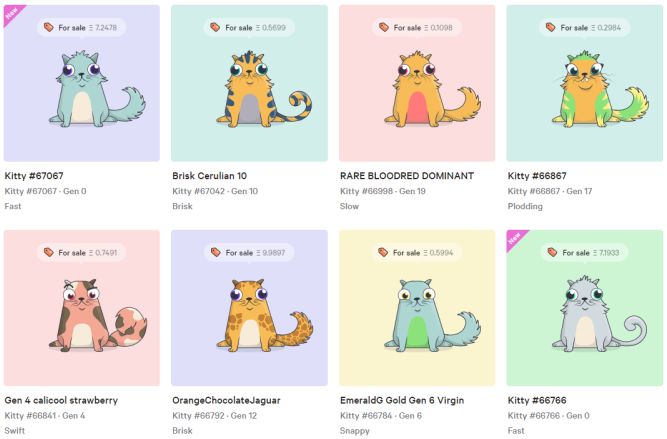

Fig. 8: Screenshot of Cryptikitties

1) Web Scan: The web assessment of CryptoKitties is shown in TABLE II. The result shows that there are 6 pages on CryptoKitties site have not set a X-Frame-Options header. This header is not included in the HTTP response to protect against "clickjacking" attacks, which meaning attackers have chances to use a transparent "iframe" to overlay the page and entice users to unwittingly click on the malicious options.

Medium	X-Frame-Options Header Not Set(6)
Method	GET
Parameter	X-Frame-Options

TABLE II: Web Risks In CryptoKitties

2) Smart Contract Audition: Part of the assessment of CryptoKitties contracts was shown in TABLE III.

Low	**Exception State(5)**
Function	isPregnant(unit256)
Function	canBreedWith(unit256, unit256)
Function	giveBirth(unit256)
Function	cooldowns(unit256)

TABLE III: Smart Contract Vulnerabilities In CryptoKitties

We detected five reachable exceptions in four categories: division by zero, out-of-bounds array access, or assert violations. Solidity uses a *require()* function to check the validity

[20]https://eoslots.com/

of determining statement. We extract the *require()* statements, which may trigger exceptions, from four functions that alerted by the Mythril.

```
pragma solidity ^0.4.11;

function isPregnant(uint256 _kittyId)
    {
        require(_kittyId > 0);
        // A kitty is pregnant if and only if this
            field is set
    }
function canBreedWith(uint256 _matronId, uint256
    _sireId)
    {
        require(_matronId > 0);
        require(_sireId > 0);
    }
function giveBirth(uint256 _matronId)
    {
        // Check that the matron is a valid cat.
        require(matron.birthTime != 0);
        // Check that the matron is pregnant, and
            that its time has come!
        require(_isReadyToGiveBirth(matron));
    }
function setSecondsPerBlock(uint256 secs)
    {
        require(secs < cooldowns[0]);
    }
```

After we examine the source code above, we found that in Cryptokittes, there is actually a little risk of triggering an exception, because Variables like "_kittyId", "_matronId" or "_matron.birthTime" were designed to fit the requirements. For example, there are no minus options on these variables that may give them any chance smaller than zero. Thus, these "Exception State" can be defined as secure.

E. 0xUniverse

0xUniverse is a blockchain game where players can build spaceships, explore the galaxy, and colonize planets. It is among the most popular blockchain game in 2019, ranked the top 3 games on Ethereum in terms of popularity.

Fig. 9: Screenshot of 0xUniverse

1) Web Scan: The web assessment of it is shown in TABLE IV. There are two high-risk vulnerabilities of remote OS injection, which have a potential risk of executing unauthorized operating system commands. When the web application takes in unauthorized input to OS command lines or doing improper call of external codes, there can be an OS injection. The second vulnerability, application error disclosure, which means the page contains an error message that discloses sensitive information. This sensitive information may help hackers in further attacks. However, in this case, it is a misjudgment of recognizing a included JavaScript class, tron-web[21], as GET method's error message.

High	Remote OS Command Injection(2)
URL	https://play.0xuniverse.com/js?query=query%22%26sleep+15%26%22
Method	GET
Parameter	query
Attack	query"&sleep 15&"
URL	https://play.0xuniverse.com/js/blockchain/TronWeb.js?query=query%26sleep+15%26
Method	GET
Parameter	query
Attack	query&sleep 15&
Medium	Application Error Disclosure(1)
URL	https://play.0xuniverse.com/js/blockchain/TronWeb.js
Method	GET
Evidence	Invalid parameter type
Medium	X-Frame-Options Header Not Set(1)
Method	GET
Parameter	X-Frame-Options

TABLE IV: Web Risks In 0xUniverse

2) Smart Contract Audition: As introduced in the Section III, integer overflow/underflow is a common vulnerability in programming. To prevent overflow/underflow, on one hand, developers can do verification before and after the calculation, or use the SafeMath[22] library provided by OpenZeppelin. On the other hand, for functions that can trigger overflows, developers should pay more attention to authentic management.

High	**Integer Overflow(1)**
Function	name()
call data	0x06fdde03
call value	0x0

TABLE V: Smart Contract Vulnerabilities In 0xUniverse

F. Mythereum

Mythereum is a multiplayer digital trading card game built on the Ethereum blockchain where players build unique decks of collectible cards and challenge others to engage in battle. The players can launch attacks while attempting to protect their own Health and outlive every other player, earning Mythereum XP along the way.

[21]https://github.com/tronprotocol/tron-web
[22]https://github.com/OpenZeppelin/openzeppelin-solidity

Fig. 10: Screenshot of Mythereum

1) Web Scan: The web security assessment of Mythereum is shown in TABLE VI. We found that Mythereum is including raven.min.js from CryptoKitties. Unless the target domain name is 100% trusted, do not use external resources on other domain names as much as possible. In our detection, Japanwar[23], another decentralized game used parlous resources is now being hijacked.

Medium	Application Error Disclosure(1)
URL	https://www.mythereum.io/main.bundle.2b361e86f68ad022f065.js
Method	GET
Parameter	X-Frame-Options
Medium	Cross-Domain JavaScript Source File Inclusion(1)
URL	https://cryptokitties.co
Method	GET
Parameter	https://cdn.ravenjs.com/3.20.1/raven.min.js

TABLE VI: Web Risks Mythereum

2) Smart Contract Audition: The smart contract vulnerability assessment of Mythereum is shown in TABLE VII.

High	**Integer Underflow(2)**
Function	tokenURI(uint256), _function_0xf319428d
call data	0xf319428d, 0xc87b56dd, 0xa22cb4650000000000000000000000101010201010102020101010101010101010101010101
call value	0x0
Medium	**Multiple Calls in a Single Transaction(6)**
Function	claim()

TABLE VII: Smart Contract Vulnerabilities In Mythereum

In the result, there was a medium vulnerability called "Multiple calls in a single transaction". When developers want to implement a payment function, they may first think of *send()*. However, external calls may fail unexpectedly or intentionally. It is usually the best isolating each external call into their own transaction that can be called by the recipient. It is safer letting users withdraw funds rather than sending funds to them. Apart from that, we also found a possible underflow vulnerability shown below. Although the developer used *assert()* to guarantee that minuend must be greater than the subtrahend, they didn't compare the result with the minuend. A filter should be added to ensure the result is smaller than the minuend, or simply use the SafeMath library[24].

```
function minus(
  uint256 minuend,
  uint256 subtrahend
) public pure returns (uint256 difference) {
  assert(minuend >= subtrahend);
  difference = minuend - subtrahend;
}
```

VI. Conclusion

Blockchain games benefit from the features of DApps. Compare to the infinite number of items that sold by the operators in traditional games, the items in the blockchain game are real properties, just like in the reality. Therefore, developers of blockchain games need to pay special attention to the security issue. However, according to the 2018 security report, the security situation of the DApps is still not optimistic. In this article, we introduced the background and related works on blockchain and smart contracts. Then, we discussed possible attack methods based on blockchain game architecture. After that, we illustrated overviews of the blockchain game security in terms of web application and the smart contracts. Finally, in the case studies, we demonstrated the result of security analysis on three games and described how to avoid these vulnerabilities in development.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *White Paper: https://bitcoin.org/bitcoin.pdf*, 2008.
[2] M. Nofer, P. Gomber, O. Hinz, and D. Schiereck, "Blockchain," *Business & Information Systems Engineering*, vol. 59, pp. 183–187, Jun 2017.
[3] V. Buterin, "Ethereum white paper: a next generation smart contract & decentralized application platform," *https://github.com/ethereum/wiki/wiki/White-Paper*, 2013.
[4] B. K. Mohanta, S. S. Panda, and D. Jena, "An overview of smart contract and use cases in blockchain technology," in *2019*, pp. 1–4, July 2018.
[5] W. Cai, Z. Wang, J. B. Ernst, Z. Hong, C. Feng, and V. C. M. Leung, "Decentralized applications: The blockchain-empowered software system," *IEEE Access*, vol. 6, pp. 53019–53033, 2018.
[6] T. Min, H. Wang, Y. Guo, and W. Cai, "Blockchain games: A survey," in *2019 IEEE Conference on Games (COG)*, August 2019.
[7] A. Back, "Hashcash - a denial of service counter-measure," 09 2002.
[8] N. Szabo, "Formalizing and securing relationships on public networks," *First Monday*, 1997.
[9] I. Nikolic, A. Kolluri, I. Sergey, P. Saxena, and A. Hobor, "Finding The Greedy, Prodigal, and Suicidal Contracts at Scale," 2018.
[10] N. Atzei, M. Bartoletti, and T. Cimoli, "A survey of attacks on Ethereum smart contracts SoK," in *Proceedings of the 6th International Conference on Principles of Security and Trust - Volume 10204*, (New York, NY, USA), pp. 164–186, Springer-Verlag New York, Inc., 2017.

[23] janponwar.xyz

[24] https://github.com/OpenZeppelin/openzeppelin-solidity/blob/master/contracts/math/SafeMath.sol