IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 6, 2019, accepted January 17, 2020, date of publication January 30, 2020, date of current version February 10, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2970495

Smart Contract: Attacks and Protections

SARWAR SAYEED, HECTOR MARCO-GISBERT, (Senior Member, IEEE), AND TOM CAIRA
School of Computing, Engineering and Physical Sciences, University of the West of Scotland, Paisley PA1 2BE, U.K.
Corresponding author: Hector Marco-Gisbert (hector.marco@uws.ac.uk)

ABSTRACT Smart contracts are programs that reside within decentralized blockchains and are executed pursuant to triggered instructions. A smart contract acts in a similar way to a traditional agreement but negates the necessity for the involvement of a third party. Smart contracts are capable of initiating their commands automatically, thus eliminating the involvement of a regulatory body. As a consequence of blockchain's immutable feature, smart contracts are developed in a manner that is distinct from traditional software. Once deployed to the blockchain, a smart contract cannot be modified or updated for security patches, thus encouraging developers to implement strong security strategies before deployment in order to avoid potential exploitation at a later time. However, the most recent dreadful attacks and the multifarious existing vulnerabilities which result as a consequence of the absence of security patches have challenged the sustainability of this technology. Attacks such as the Decentralized Autonomous Organization (DAO) attack and the Parity Wallet hack have cost millions of dollars simply as a consequence of naïve bugs in the smart contract code. In this paper, we classify blockchain exploitation techniques into 4 categories based on the attack rationale; attacking consensus protocols, bugs in the smart contract, malware running in the operating system, and fraudulent users. We then focus on smart contract vulnerabilities, analyzing the 7 most important attack techniques to determine the real impact on smart contract technology. We reveal that even adopting the 10 most widely used tools to detect smart contract vulnerabilities, these still contain known vulnerabilities, providing a dangerously false sense of security. We conclude the paper with a discussion about recommendations and future research lines to progress towards a secure smart contract solution.

INDEX TERMS Smart contracts, attack techniques, DApp, Ethereum, vulnerability.

I. INTRODUCTION

A blockchain is a distributed network that is leveraged for various purposes [1]. It is an immutable ledger technology where the recorded information is open and can be viewed by everyone. It does not involve any central authority to monitor the regular flow of the network, making it less prone to attacks. Miners' consent is required to verify the authenticity of any acts performed in the blockchain platform. Since its inception, blockchain has primarily been utilized for crypto transactions. However, blockchain is not all about cryptocurrencies, rather, it extends far beyond this. Over time, the remarkable advancement of blockchain has made it possible to apply it to various other activities. The smart contract is one of these applications which allows agreement to be formed and authentic transactions initiated between parties without the involvement of middlemen. This improves upon the traditional approach where users ended up expending an unreasonable amount of time and effort.

Ethereum is a blockchain platform which provides tools for developers to build decentralized applications which, unlike Bitcoin, can be utilized for various purposes [2], [3]. Bitcoin blockchain enhances a peer to peer digital cash system which allows the participant to perform online transactions [4], whereas, besides digital transactions, Ethereum is also utilized to execute smart contract code in decentralized applications which are deployed on the network.

A smart contract functions in a similar way a normal contract works among two or more parties. Parties do not need to rely on lawyers or banks to set up an agreement for them, rather, the smart contract gets executed automatically to issue payment once certain conditions are met. For instance, a lease agreement, insurance contract, or real estate payments can be in the form of a smart contract.

A smart contract is basically a piece of code that resides inside the blockchain, ensuring the stated conditions are met to fulfill the user's requirement [5]. The written code is pub-

The associate editor coordinating the review of this manuscript and approving it for publication was Tyson Brooks.

licly visible in the blockchain, and transparent to anyone who is connected to the network. Upon fulfilling the conditions by the desired time, the contract gets triggered to execute the digital transaction. Since the conditions are encrypted cryptographically, no party is able to alter the contents of a contract. The immutable nature of blockchain also ensures that every single device connected to the network contains a copy of the contract, thus securing a backup version of the contact.

Being open-source, the contract code enables the involved parties to determine what the contract does and how it is initiated [6]. It also guarantees the execution of the contract without being affected when certain parts of the network are down or being attacked by adversaries. Once the contract is placed within the blockchain, it is nearly impossible to have it removed or deleted unless the whole blockchain network is exploited by some significant attack techniques. However, such attacking efforts may involve a huge amount of capital, requiring an adversary to re-generate every block that is chained after the affected block.

Despite all the security enhancements and security tools [7], [8], blockchain still faces challenges to cope with various pernicious attacks [9]. A range of attacks are constantly initiated to obstruct the natural flow or even fully destroy the network [10]. Attacks relating to cryptocurrency wallets, smart contracts, transaction authentication, mining pools, and blockchain networks are frequently exploited by adversaries. DAO attacks, King of the Ether Throne, and Multi-player Games are some smart contract based attacks which occur due to the bugs in smart contract code [11].

This paper focuses on examining smart contract based attacks as well as the consequences of their exploitation. We do not aim to determine how much effort it takes to execute particular attacks. Rather, we focus on various attack types and available security tools to restrict those attacks, as well as limitations that exist to those security enhancements.

II. BACKGROUND

This section involves reviewing relevant context associated with Ethereum smart contracts and attacks launched on them. The investigation of the literature helps to grasp the concepts of blockchain technology and smart contract frameworks, and provides an overview of Ethereum, decentralized applications, and past smart contract-based attacks.

Figure 1 shows a comparison between client-server architecture and decentralized DApp architecture. TBC.com, an application, contains front-end and back-end parts [12]. The front-end is developed using HTML and can be viewed by any clients, whereas the back-end is developed using Node.js. Both ends communicate with each other via JSON using HTTP protocol. All the confidential data is saved to a central server. In the case of a smart contract-based decentralized application (DApp), the back-end is the smart contract and the confidential data is saved at Block 45 of the blockchain. Every node that is connected to the blockchain has a version of the smart contract code, which is immutable. Exploiting a single or a few nodes will not significantly affect the actions of TBC.com, conveying the security aspects of blockchain over a centralized system.

A. BLOCKCHAIN AND DIGITAL TRUST

The blockchain is one of the creative innovations of the era, which is able to distribute digital information securely. It combines three main technologies which ensures its proper function; private key cryptography, peer-to-peer networking, and the consensus protocol [13]. An intricate cryptography approach is implemented to secure transactions and a hashing method is used to provide a fixed-length output [14]. For instance, a transaction sent by Alice will be shown as

```
"0x1dc676eba2eed839c98fab067gt67e0.......",
```

Hence, securing the user's identity while only the public address and the transfer amount is visible to others. Using the public key it is possible to determine any transactions made by a participant. Similarly, the immutability of blockchain makes it a tamper-free system. Once any content is recorded in the blockchain, it can not be omitted from the ledger.

Moreover, the block validation process involves network miners to determine whether a particular block is valid. Anyone can join the network as a miner to participate in the validation process [15]. The block generation method differs in different platforms, although Ethereum's mining process is very similar to Bitcoin [16]. Ethereum, along with many other digital currencies, follows the Proof of Work (PoW) consensus protocol. The PoW protocol utilizes the 'ethash' algorithm for the mining task. The validation process makes use of powerful computers to solve a puzzle. A miner with a valid hash is awarded ether, and the generated block is then added to that particular blockchain. The block generation time of Ethereum is roughly 12-15 seconds.

B. SMART CONTRACT FRAMEWORKS

Ethereum is one of the major platforms which is used for the development of a smart contract. Smart contract developers are permitted to develop any decentralized application (DApp) they wish on the Ethereum platform. The decentralized applications trigger exactly as per the code conditions without having any risk of censorship, deception, or downtime. However, besides all the advantages, no one can claim it to be a fully secure platform. For instance, unexpected bugs in the smart contact code may lead the contract to trigger unintended tasks it is not set to perform. Hence, parties involved with the contract may experience huge loss as a consequence of the unresolved agreement. Besides Ethereum, there are other smart contract platforms that are utilized for the development of DApps.

Hawk is another framework for developing privacy-preserving smart contracts [17]. Hawk does not require cryptography implementation, so it provides opportunity for non-programmers to write a Hawk program. A Hawk compiler is in place to compile Hawk programs. One-chain

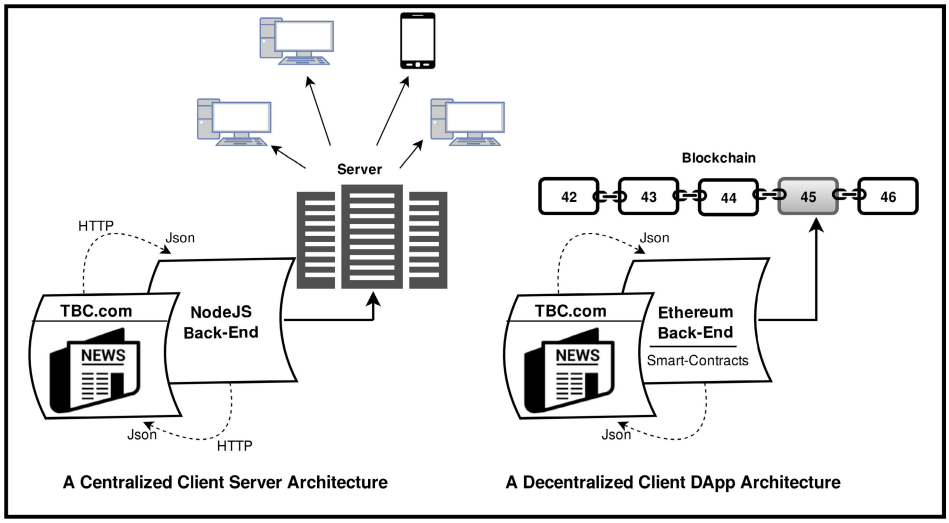

FIGURE 1. A comparison between Client-server architecture and Decentralized DApp architecture.

privacy and contractual security are two security approaches guaranteed by Hawk to enhance protection. Hawk is the first smart contract system which establishes a recognized, academic provision of the blockchain model of cryptography. It comprises a Universal Composability (UC) model, which consists of independent interest. The UC model is a formal model that can be leveraged, simulating security protocols in the blockchain.

Hawk utilizes zkSNARKs to verify smart contracts. However, SNARKs involves pre-circuit trusted setup, requiring a new setup in place for any program which is implemented by a contract [18]. A multi-party calculation can be adopted to diminish reliability on the setup, but this is inappropriate to utilize on the per-circuit basis that is desired by Hawk. Moreover, HAWK suffers from scalability challenges and, in addition, the privacy aspect is entirely handled by a third-party manager, making all confidential data insecure. Hawk, and other frameworks which are focused on achieving privacy-preserving contracts, suffer from serious flaws [19]. They cannot be affiliated with digital currencies. As such, all these frameworks may incur high costs for transaction processing.

EOS and Tron are also smart contract based platforms which incorporate scalability [20].

C. ETHEREUM SYNOPSIS

The Ethereum blockchain has its own cryptocurrency which is ether. Ether is the token which powers the Ethereum blockchain [21]. Ether operates in a slightly different way from Bitcoin, and is also utilized in smart contracts. A smart contract is computer code which incorproates an automated legal agreement [22]. Vyper, Bamboo, Serpent, and Mutan are a few programming languages that have been used to write smart contract code. However, currently, Solidity is the prime language adopted for writing smart contracts. The implementation of smart contracts within the blockchain makes it immutable, therefore, a deployed contract can never be voided or erased.

Gas is a term that is used as a fee in the Ethereum platform [23]. Gas is often estimated by the computational performance of a smart contract. Distinct smart contracts require different volumes of gas in order to execute a required task. The gas requirement of a smart contract can be determined by applying the following rule

```
Ether = Tx Fees = Gas Limit * Gas Price
```

Besides gas, the Ethereum Virtual Machine (EVM) is another significant aspect of the Ethereum blockchain. The EVM generates a degree of abstraction between the executing code and the machine that executes it [24]. The layer ensures that the DApps are detached from each other as well as from hosts. Solidity code needs to be compiled to opcode in order for the EVM to execute it. The EVM utilizes the opcodes to carry out various tasks. There are about 140 distinct opcodes that enhance the EVM to be Turing-complete, allowing it to evaluate anything. The opcodes are encoded to bytecode to determine proper security. The EVM dominates the inner part of the Ethereum blockchain and also consists of a detail list of the status to initiate a transaction [23].

D. DECENTRALIZED APPLICATION

A decentralized application is also referred to as a DApp [6]. DApps are open-source applications based on the Ethereum blockchain where a consensus is maintained between the user and programmer during the development process. The source code is available for examination and the application is stored in the blockchain to ensure trust and transparency. Miners are responsible for securing the application and are rewarded with tokens for the validation of the DApp. Bitcoin can be considered as a DApp on the Bitcoin blockchain platform. Nevertheless, the Ethereum blockchain is recognized as a bigger platform for decentralized applications.

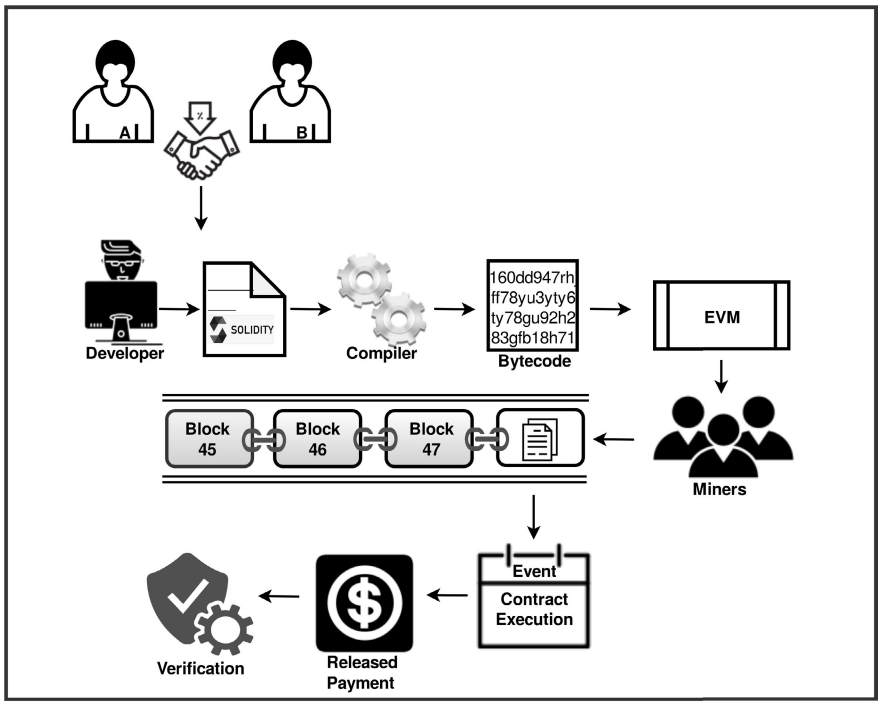

FIGURE 2. Total cycle of smart contract execution over Ethereum blockchain.

Likewise, a Decentralized Autonomous Organization (DAO) aims to categorize the policies of an organization by limiting the requirement for centralized activities with the use of a decentralized approach [25]. It may involve a few participants contributing to the writing of smart contract code. Having an initial funding period, participants are allowed to include funds to purchase tokens, and the DAO begins its execution after the funding period. Participants have the option to propose approaches to utilize the funds, and a few participants can also vote to determine the approval of the proposals.

Figure 2 shows the sequence of smart contract execution over Ethereum blockchain. Two parties reach an agreement, which is then written by a developer using Solidity code. The code is then compiled to bytecode for the EVM to process. Miners' involvement is required for processing the contract to the blockchain. Once included, the contract gets processed on the event scheduled date, triggered by the written code. The execution of the contract releases the payment to the appropriate party, which can later be verified by anyone.

E. SMART CONTRACT-BASED ATTACKS

There have been a lot of attacks on smart contracts, costing a large amount of money. However, the DAO attack and the Parity Wallet hacks are the most often discussed.

In May 2016, a few participants from the Ethereum society inaugurated the DAO [26]. The inception was known as genesis DAO. The DAO was an open-source smart contract that allowed anyone to exchange DAO tokens with ether. That method of exchange helped to gather around $150M, providing DAO with a large crowdfund. Participants with DAO tokens were permitted to cast their vote on propositions and receive rewards as long as it resulted in profit. However, the DAO contract contained severe flaws, allowing attackers to remove funds. A loophole existed which permitted an attacker to request funds from the smart contract numerous times before the balance was updated. The vulnerability occurred due to bugs in the code where the developers did not consider the potential for a recursive call. Hence, it enabled attackers to steal ether worth millions of dollars within the first few hours. The DAO attack scenario demonstrates how destructive a simple smart contract vulnerability can be.

Similarly, the Parity Wallet hack is another vulnerability which was discovered on the Parity Multisig Wallet with version 1.5+ [27], [28]. The flaw permitted an attacker to remove over 150,000 ETH (30M USD). In order to execute the attack, the adversary transmitted two transactions aiming to acquire ownership of Multisig so that all the currency could be drained. Once the attack was accomplished, the Parity Multisig Wallet Library contract was initiated. However, it contained a bug which authorized anyone to run initWallet [29]. The attack was executed twice; hence, it is referred to as Parity Wallet hack 1 and 2. In the first attack, the attacker was able to modify the status of the wallet by initiating a call to `initWallet`. As a result, the attacker was believed to be the owner and drained funds without any hindrance.

III. ATTACK CLASSIFICATION

In this section, we classify blockchain-based exploitations into 4 categories. Our study indicates that most blockchain

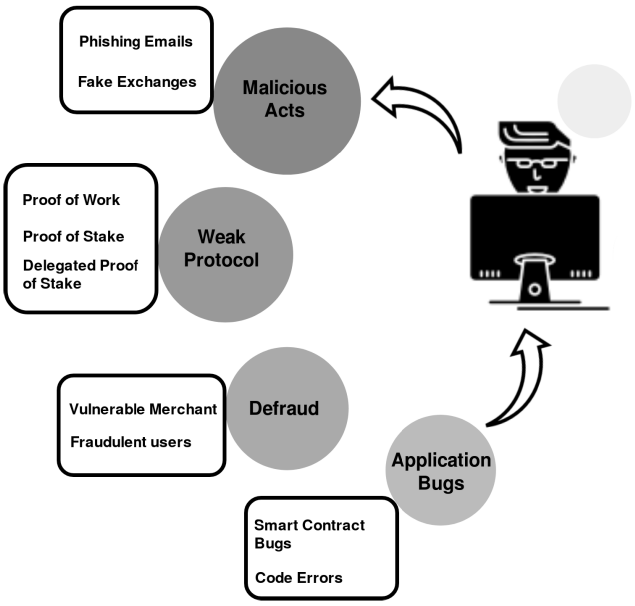

FIGURE 3. A classification of exploitation based on the fragile methods.

attacks fall under the listed categories. Potential adversaries utilize specific categories to initiate an attack in accordance with their attacking capability. Although this section focuses on classifying the exploitation techniques, the rest of the paper concentrates on a single category which is `application bugs`.

Figure 3 shows the classification of exploitation and attack vectors that fall under each classification. Our analysis does not determine how much effort is required to perform particular attacks, rather, it points out the main flaws of particular exploitations.

A. MALICIOUS ACTS

Malicious acts may comprise the action of spreading malware to deceive users. This type of exploitation is mostly initiated over the Internet to compromise user identity or conduct fraud through the use of malware or viruses. Such malicious activities can seriously impact a victim's financial circumstances [30]. Malicious attacks may arise in any form such as an email from the wallet asking to sync the account with a network that has just been hard-forked. Exploitation of user wallets through malicious attacks may allow an attacker to drain all the currency. Crypto-jacking, slack, and forums attacks are a few malicious techniques asking miners to log in through corrupted links [31]. Glupteba is another malware that utilizes the Bitcoin blockchain for its update. Thus, it remains active despite the server connection being terminated by the antivirus. This malware spreads through scripts to steal confidential information such as user id, passwords, browsing history, saved cookies, etc. [32].

B. WEAK PROTOCOL

Blockchain comprises a consensus protocol to keep the network flowing. Different blockchain platforms have adopted different protocols. Exploitations due to weak consensus have been very common in recent times, although it can often be very expensive to carry out attacks due to flaws in the consensus protocols. However, successfully executed attacks can remove blocks from the chain, destroy a blockchain fully, or acquire full control over the price of a cryptocurrency. 51% Attack, Selfish mining, and 34% Attack are some attack techniques that occur due to weak protocols.

The PoW protocol assumes that 50% of network miners will always to be honest miners. Thus, adversaries comprising more than 50% hashing can gain control of the network [33]. Weak consensus can also lead to numerous attacks related to the blockchain network. The Sybil attack permits an attacker to establish several malicious nodes over the Bitcoin blockchain network. The malicious nodes are then used to corrupt the network, conduct unprivileged transactions, or alter valid transactions. Similarly, an Eclipse attack can be executed to manipulate the Peer to Peer (P2P) network in order to gain full control over the information a node comprises. In addition, Border Gateway Protocol (BGP) hijacking makes false declarations over the routing system to divert the traffic. Thus, regardless of the decentralized feature, the blockchain network can still be compromised by various attack techniques due to weak consensus.

C. DEFRAUD

This exploitation tricks merchants to take advantage of the unstable actions of digital transactions. Defraud may influence the merchant to release goods prior to a transaction being fully confirmed. In a normal scenario, a Bitcoin transaction is confirmed after 6 transactions. However, a consumer may persuade a merchant to release goods without the wait for up to 6 transactions, so that attack techniques such as 1 confirmation or n confirmation could be initiated to double spend. Similarly, in recent times, various retailers are accepting cryptocurrencies, allowing consumers to receive their product instantly [34]. For example, purchasing a coffee from a coffee shop. Consider a scenario where an adversary manages to spend the same cryptocurrency within a short span of time, which will onset a race between both transactions. If the second transaction is adopted by the pool miners for processing, then the first transaction will be discarded, potentially leaving the merchant unpaid for the provided goods.

D. APPLICATION BUGS

An application bugs exploitation emerges when there is an error in the smart contract code. This exploitation mainly occurs in smart contracts. It arises when developers fail to identify code errors in the decentralized application. Attackers are able to drain all the money from the contract wallet through simple code bugs. Smart contract applications are similar to web applications that run over the blockchain. Like web application bugs, they also comprise errors, however, these bugs can lead to serious challenges. For example, the DAO was able to raise $150m, whilst the attacker was able to steal about $60m due to code bugs. [35]. Rubixi and

TABLE 1. Classification of attacks based on exploitation.

Malicious Acts	Weak Protocol	Defraud	Application Bugs
Mining Malware	51% Attack	Double Spending	Reentrancy
Cryptojacking	34% Attack	1 Confirmation Attack	DELEGATECALL
Slack and Forums Attack	Sybil Attack	n Confirmation Attack	Overflow/Underflow

```
function revoke() remote{
   uint256 value = balances[msg.sender];
   require(msg.sender.call.value(value)());
   balances[msg.sender] = 0;
}
```

Listing 1. A vulnerable function that can be exploited by Reentrancy.

GovernMental are some of the smart contract applications which had flaws due to code bugs [11]. Application bugs may not only allow attackers to steal money, but also influence an application to function differently.

IV. ATTACK TECHNIQUES

In this section, we define seven attack techniques which can have a serious impact on a smart contract application. Successful execution of such attacks may lead the smart contract to perform in an expected manner. Hence, parties associated with the contract agreement might incur a severe loss.

A. REENTRANCY

Reentrancy is considered to be one of the most catastrophic attack techniques in the smart contract [36]. This attack technique is able to fully destroy the contract or steal valuable information. Reentrancy may occur when a function calls for another contract through an outer call. Listing 1 below presents a code snippet which can be exploited to execute a Reentrancy attack. The exploitation allows an attacker to execute a recursive callback of the main function, making an unintended loop which is repeated many times. For instance, when a vulnerable contract contains a `revoke` function, the contract may call the `revoke` function illicitly numerous times in order to drain any available balance the contract comprises. Single function Reentrancy attacks and cross-function Reentrancy attacks are two different types that can be exploited by the attackers. The exploitation allows the attacker to use external calls to execute the desired tasks.

B. SMART CONTRACT OVERFLOW AND UNDERFLOW

This vulnerability is relatively easy to initiate and occurs in transactions that accept unauthorized input data or value [37]. Smart contract overflow mainly occurs when more value is provided than the maximum value [38]. The contracts are mainly written in Solidity which can handle up to 256-bit numbers, thus, an increment by 1 would cause an overflow. Conventional testing approaches are inadequate for determining overflow vulnerability in smart contacts.

Listing 2 shows smart contract code which comprises bugs at the following line [39],

```
uint256 total = uint256(cnt) * _value;
```

```
function batchTransfer(address[] _acceptors,
     uint256 _value) public whenNotPaused
    returns (bool) {

   uint cnt = _acceptors.length;
   uint256 total = uint256(cnt) * _value;
   require(cnt > 0 && cnt <= 20);
   require(_value > 0 && balances[msg.sender
       ] >= total);
   balances[msg.sender] = balances[msg.
       sender].sub(total);
   for (uint i = 0; i < cnt; i++) {
      balances[_acceptors[i]] = balances[
          _acceptors[i]].add(_value);
      Transfer(msg.sender, _acceptors[i],
          _value);
   }
   return true;
}
```

Listing 2. A vulnerable contract which can be exploited by Smart contract overflow.

An attacker is able to call this function with parameters to exploit the vulnerability. For instance, the code below shifts the check to `balances[msg.sender] >= total`. An attacker can input 2 addresses in the receivers function in order for the token smart contract to transmit ether to both addresses.

```
total = _value * 2 = 0
```

Smart contract underflow occurs in the opposite way to overflow [40]. However, the underflow attacks are more simple to perform as achieving the required token to cause overflow is often challenging for attackers.

C. SHORT ADDRESS ATTACK

This vulnerability occurs due to the weakness of the Ethereum Virtual Machine (EVM) [41]. The EVM permits imprecise padded arguments allowing adversaries to dispatch specially crafted addresses which result in the exploitation. The Short Address Attack follows a similar attacking strategy as a SQL injection bug [42]. When an underflow is detected the EVM includes a zero at the end of the address in order to ensure that it comprises a 256-bit datatype. However, an adversary can take advantage of this vulnerability by omitting the very last zero from the ether address. This vulnerability is an input validation bug and mainly occurs from the sender's side due to weak transaction generation code.

D. DELEGATECALL

Smart code developers leverage the CALL and DELEGATECALL to modularise written code [43]. The DELEGATE

```
contract GuessAddress {

   function gainEther() {
      // Winner while the last 8 hex
          characters of the address are 0.
      require(uint32(msg.sender) == 0);
      _sendEther();
    }

    function _sendEther) {
       msg.sender.transfer(this.balance);
    }
}
```

Listing 3. A vulnerable contract drained by attacker to steal Ether.

opcode comprises a similar function to the message CALL, however, other than the code executed to call for a contract, the `msg.sender` and `msg.value` does not get altered. This attribute allows developers to generate re-usable code, enhancing the chance of abrupt code-execution by the use of DELEGATECALL. The DELEGATECALL feature shows that it is possible to introduce flaws while building custom libraries and it can also lead to new vulnerabilities. DELEGATECALL vulnerabilities can be avoided by observing for a lapse on both the library contract and calling contract and, in addition, developing state-less libraries whenever feasible.

E. DEFAULT VISIBILITIES

The visibility specifiers in the Solidity function control the manner in which a function is to be called [43]. The visibility specifier also takes control when permitting users to call for external functions by derived contracts. Improper implementation of the visibility specifiers can cause serious effects in the smart contract. The default visibility is always set to public for functions, allowing external contracts to call for visibility when functions do not explicitly mention it. This vulnerability arises when developers neglect to set the visibility specifier to private.

Listing 3 shows a smart contract based on an address guessing game [43]. A participant can win a reward by producing an Ethereum address which must contain zeroes in its last 8 hex characters. Once the requirements are fulfilled, the `gainEther()` function can be executed to receive the reward. Since the vulnerable code does not specify the visibility and the `_sendEther()` function is set as public, an attacker will be able to steal the reward.

F. TRANSACTION ORDERING DEPENDENCE(TOD)

Transaction Ordering Dependence (TOD) is a vulnerability that can allow corrupt miners to have a serious effect on smart contracts [44]. This vulnerability is a very common security bug in the smart contract, relying on the order of transaction execution [45]. For example, a newly generated block contains 2 transactions enforcing the same smart contract. Such plots do not provide enough information to users to determine the state of the contract or when the individual invocation is initiated. Therefore, when the output of both transactions is dependent on the order, the contract results in a TOD vulnerability.

In Ethereum blockchain, the miners are in charge of controlling the order of transactions, prioritizing transactions with higher gas. Hence, any miner that closes a block can influence the order of a transaction. The ability for potential miners to influence the transaction order for illicit activities is an outcome of Transaction Ordering Dependence (TOD).

G. TIMESTAMP DEPENDENCE

Timestamp Dependence is another vulnerability that can be exploited by corrupt miners [44]. In order to gain a benefit, a miner may re-arrange the timestamp by a few seconds. The timestamp dependence vulnerability occurs from a flawed comprehension of timekeeping [46]. It enables the Ethereum network to be detached from the synchronized global clock. For example, a smart contract utilizes the current timestamp to produce random numbers in order to determine the lottery result. Since the smart contract permits miners to put up a timestamp within 30 seconds of block validation, this gives a miner more opportunity for exploitation. Hence, the outcome of the random number generator can be altered to gain benefits.

V. SECURITY TECHNIQUES

In this section, we discuss 10 major security analysis tools which are in place to find vulnerabilities in the smart contract. Most tools are mainly utilized for static and dynamic analysis of smart contract codes.

A. SLITHER

Slither is a static analysis framework for smart contract code [47]. Its security detection techniques for potential bugs are fast and reliable. Slither can be used to perform main tasks such as automated vulnerablity detection, automated optimization detection, code understanding, and assisted code review. A multi-stage procedure is initiated for the security analysis. The Solidity compiler produces a Solidity Abstract Syntax Tree (AST) from the contract source code and the AST is used as an input to Slither. During the initial stage, Slither obtains significant contract information such as the inheritance graph, Control-flow graph (CFG) [48], etc. The next stage includes converting the full code to SlithIR. In the following stage, the code analysis task is performed by computing a list of pre-defined analyses.

B. MYTHX

MythX is a security analysis service that scans EVM-based smart contracts for vulnerabilities [49]. It comprises various analysis techniques which include static, dynamic, as well as symbolic execution. The main objective of MythX is to support DApp developers with the development of smart contracts to ensure a safer platform. MythX does not serve the requirements by itself, rather it is integrated with development tools such as Truffle and Remix. It is not only

compatible with the Ethereum platform - developers associated with Tron, Vechain, Quorum, Roostock and a few other EVM-based platforms can also take advantage of these security tools to find bugs in a smart contract. MythX goes through three stages to analyze smart contract code. First, it requires developers to submit their code; second, a complete suite of analysis techniques needs to be activated; finally, it generates an analysis report demonstrating if any errors exist.

C. MYTHRIL

Mythril is a security tool that analyzes smart contracts written by Solidity [50]. Mythril, an open-source tool, takes advantage of the symbolic execution technique in order to determine the errors in code. The examination of security flaws involves executing smart contract bytecode in a custom built EVM. Mythril goes through four major working stages to accomplish its security analysis. When a flaw in a program is discovered, the input transactions are analyzed to determine the possible reasons. This security method helps to deduce the main cause of the program vulnerability, and also mitigate exploitation. If a developer produces the source code of the contract, Mythril is able to locate the bugs within the code.

D. MANTICORE

Manticore is a Solidity audit tool that performs a symbolic analysis of smart contracts [51]. The main functions of manticore involve tracing inputs that terminate a program, logging instruction-level implementation, and providing access to its analysis engine through Python API. It has a dynamic symbolic execution feature which analyzes binaries as well as Ethereum smart contracts [52]. The primary attributes in Manticore's architecture comprise the Core Engine, Native Execution Modules, and Ethereum Execution Modules. The Satisfiability Modulo Theories (SMT-LIB) module, Event System, and API are regarded as secondary attributes.

E. SECURIFY

Securify is a smart contract security analyzer tool [53]. Securify is an automated tool able to determine whether the contract performs accordingly, based on the provided attributes. Securify is an open-source product whose security analysis function goes through two stages to perform the required task [54]. Up to this point, around 18000 contracts have been submitted to Securify for security analysis. Securify accepts EVM bytecode for security analysis. Contracts written in Solidity are also accepted as an input, however, the code needs to be compiled to EVM bytecode for the security process to be effected. When a security violation is triggered, Securify produces a command which induces the violation pattern to match. Similarly, when both the violation and compliance pattern do not match, it generates a warning. The security analysis technique of Securify is unique when compared with other tools such as Oyente and Mythril [55]. While Oyente and Mythril symbolically enumerate distinct paths of a contract, Securify utilizes static analysis to analyze every path of the smart contract.

F. SMARTCHECK

SmartCheck is an automated extensive vulnerability analysis tool for Solidity smart contracts [56], [57]. SmartCheck is an open-source engine which not only points out the vulnerabilities in the smart contract code but also clarifies the cause of the vulnerabilities with proper description and recommendation. SmartCheck was implemented by utilizing XPath [xpa] queries on the intermediate representation (IR) to detect vulnerability patterns. SmartCheck protects any analyzed code that has been converted to IR and elements associated with it are determined with XPath matching.

A security experiment was initiated by SmartCheck on over 4600 valid contracts. It was determined that 86.6% of the contacts comprised zero balance, whereas a single contract consisted of a balance of only 38.4% of the total balance. The SmartCheck analysis indicated that 99.9% of analyzed contracts contained some kind of security flaw, with 63.2% of contracts being severely vulnerable.

G. ECHIDNA

Echidna is an EVM smart fuzzer that identifies bugs in Solidity code [58]. This tool only requires the Solidity propositions to conduct deep analysis for bugs and provides a clear user interface (UI) to simplify its output. Echidna utilizes different combinations of inputs until it manages to break the provided property. Echidna contains a few similar attributes to Manticore, which allows it to function at the EVM level [59]. In addition, it can also be consolidated to continuous integration (CI) in order to identify code bugs whilst development is in process. A myriad of tools are supplied by Echidna in order to compose custom analyses for dealing with complicated contracts. This tool utilizes stack, therefore, the required dependency will be based on the solc version that the contract employs.

H. OYENTE

Oyente is a symbolic execution tool which is used to find security bugs in smart contracts [60]. Oyente examines Ethereum smart contracts to identify security loopholes which can cause potential threats. Oyente not only detects unsafe bugs but also investigates every practical execution path. An experiment carried out by Oyente on 19,366 smart contracts resulted in 8,833 of them being identified as vulnerable. The symbolic execution method symbolically represents the nature of an execution path as a mathematical formula. OYENTE carries out a comparison between the new formula and formulas that comprise ordinary bugs to figure out if both formulas are valid simultaneously.

I. VANDAL

Vandal is another security analysis framework for smart contracts. Vandal comprises an analysis pipeline which transforms EVM bytecode into semantic logic relations [61].

Vandal is a very fast and efficient security analysis tool that has examined over 95% of 141000 smart contracts with an average run-time overhead of only 4.15 seconds. The low overhead beats the overall performance of major existing security analysis tools. The security design of Vandal comprises a declarative language called Soufflé. Performing security analysis in a declarative language helps security analysts with the prototype of the latest analysis.

J. ZEUS

Zeus is a practical framework to examine the validity of smart contracts [62]. It takes advantage of abstract interpretation, and symbolic model checking for analyzing the safety of smart contracts. The Zeus prototype has tested over 22400 smart contracts, showing that about 94.6% of these contracts are vulnerable. Zeus accepts the smart contract code and generates the authentic version in an XACML-styled template. The smart contract code and the policy specifications are translated to LLVM bitcode to enhance the contract's behavior. Zeus performs static analysis of the furnished smart contract code to append the assert statement policy at the right spot of the program.

VI. SECURITY ANALYSIS AND LIMITATIONS

Having bugs in smart contract code can have serious consequences. Attacks such as DAO or Parity Wallet hacks, discussed in section II show the effects of such exploitations. In this section, we analyze the 10 security techniques discussed in section V. Our analysis reveals the limitations of particular techniques, and also determines their ability to discover vulnerabilities.

Slither includes a few limitations. It lacks formal semantics, which limits its ability to perform more detailed security analysis [47]. It also fails to determine low-level information precisely, for instance, the gas computation. Slither's vulnerability detection process is similar to SmartCheck [63]. It misses vulnerable codes and terminates the scanning process when the security regulations do not coincide in a severe external call. However, besides these limitations, an experiment on detection capability demonstrates that Slither can detect major vulnerabilities such as Reentrancy, contract suicidal, an abuse of Tx origin, and time manipulation.

MythX is able to detect some critical vulnerabilities such as access controls, integer overflow, and integer underflow [64]. The Remix Integrated Development Environment (IDE) can be enhanced by a MythX plugin. The Mythx plugin uses the trial account credentials. The main limitation of the trial account is that it is able to examine only a limited number of vulnerabilities.

Although using a heuristic, Mythril is known for its high accuracy in security analysis. However, experiments suggest that Mythril consists of a few limitations [65]. For instance, Mythril is unable to extend taints over memory fields when analyzed with taint analysis. Issues can be exacerbated when the parameters accept pass by reference. Moreover, the definition of the pattern is complicated in searching for the best approximation for the behavior. Another experiment indicates that although Mythril is able to defend against vulnerabilities such as TOD, Reentrancy, and TX.origin, it was able to recognize only 12 vulnerabilities out of 18 [66].

Manticore defends against the popular Reentrancy vulnerability as well as Abuse of TX origin [63]. However, it is unable to detect contract suicidal and time manipulation. It also does not analyze various security issues such as TOD, Random number, visibility, costly pattern, etc. One of the major disadvantages of Manticore is that it performs analysis for different types of attack techniques; hence, the implementation is quite sluggish [51]. A Solidity compiler and state-of-the-art theorem prover z3 are the prerequisites for running Manticore. Although symbolic analysis techniques are being widely reviewed from a security perspective, they are not being fully exercised due to the limited flexibility and user-availability. An experiment on smart contracts from Ethereum blockchains with a set time out of 90 minutes on each contract shows that Manticore was able to produce an average coverage of 65.64%.

Securify is an advanced tool comprising formal guarantees. A security experiment suggests that Securify only targets 7 issues for security analysis among 18 blockchain based challenges [66]. Besides some security advantages, Securify contains severe flaws. Securify does not comprehend numerical analysis [67]. Hence, it is unable to recognize overflows, allowing for potential bugs in the smart contract code [53]. Similarly, Securify determines that all contract instructions can be reachable. Moreover, some of the attributes for property violations are also vulnerable and can be compromised by potential adversaries.

SmartCheck is unable to detect some severe program bugs, which can only be detected by taint analysis or handled through manual audits [56]. Taint analysis is a way of checking program variables that can be affected by user input [68]. One of the possible reasons for a program to crash can be illicit user input. Hence, in order for a program to run effectively, user input must be thoroughly checked. However, SmartCheck is an effective tool for identifying simple program bugs. An experiment among 4 security analysis tools, namely, Oyente, Securify, Remix, and SmartCheck, indicates that SmartCheck is not very consistent in terms of performance and that additional security features must be included for accuracy in vulnerability checks [69]. SmartCheck only identifies vulnerabilities that are low risk to the contract. For instance, incorrect compiler version, improper style guide, and redundant functions. Similarly, another experiment based on the detection capability of various security tools shows that SmartCheck is unable to detect some serious attacks such as Reentrancy and contract suicidal [63].

Echidna generates inputs to fuzz smart contract code. However, one of the major limitations of Echidna is that it does not offer any direct application program interface (API) endorsing security checks of smart contracts [70]. Moreover, Echidna fails to provide satisfactory security results [71]. The randomness of inputs makes only a portion of the path space

obtainable, whereas some complicated parts of the program are secured by branch conditions. Random mutation does not fulfill the requirements of branch conditions, hence the program remains exploitable. In addition, Echidna exercises various generation methods for different data types [72]. While the Haskell is ignored, the `address` is the only data type that can produce an impact. When a list of addresses is determined as a yaml list, those addresses provide more chances to discover bugs as compared to fully random addresses.

Oyente is a smart contract auto-auditing security analyzer tool which is able to detect severe smart contract bugs. However, there still lie challenges as Oyente is able to detect only 20.2% of Parity Wallet hacks [53]. An analysis indicates that Oyente generates false positives and also underestimates some serious bugs. It does not provide full protection to smart contract code and fails to log 72.9% of TOD vulnerabilities. Moreover, Oyente also consistently fails to determine other critical vulnerabilities. An experiment on Oyente's vulnerability detection capability suggests that it is only able to defend against attacks such as Reentrancy and Time manipulation [63]. However, it is wholly unsuccessful in identifying vulnerabilities such as contract suicidal and Abuse of TX origin. Research suggests that Oyente protects against only 4 out of 18 blockchain-based security challenges [66]. Hence, this security approach is not fully protective.

The vandal security design faces challenges while translating smart contract code into logic relations [61]. An analysis pipeline is used to transform Ethereum bytecode into logic relations. The challenge lies when the low-level stack-based abstract machine executes the EVM bytecode. Moreover, the Vandal decompiler cannot cope with transforming the EVM's stack-based operations into a register-based intermediate representation; hence, it crashes when decompiling the major portion of the smart contract [73]. The implementation of Vandal also suffers from engineering limitations, thus proper control of timeout may not be achieved [74]. A security experiment shows that Vandal is able to detect only 5 out of 18 critical blockchain-based security issues [66].

ZEUS comprises a few limitations. Attributes involving mathematical equations cannot be fully validated [62]. For such operations, ZEUS relies entirely on users to test practices involving mathematical attributes. Solidity constructs, such as `throw` and `selfdestruct`, are simulated as a program termination. The run-time behavior of ZEUS does not consider such parameters. Similarly, it fails to reinforce virtual functions and examine contracts which contain assembly blocks. The validation of safety properties are acknowledged by ZEUS. However, verification of liveness is not endorsed by ZEUS. Static analysis tools may not be fully able to detect cross-function Reentrancy vulnerabilities because every external function is required to be checked to keep the contracts function safe [75]. In addition, ZEUS does not involve policies to execute cross-function analysis.

VII. CONCLUSION

Smart contract technologies enable users to form decentralized digital agreements without the need for a third party. The smart contract technology attracted sectors such as health, business management, shareholder agreement and insurance. However, the more this technology expands, the more it catches the attention of potential attackers, resulting in several severe exploitations.

In this paper, we revealed that this technology is not free from vulnerabilities and attacks. Based on the attack vector, we proposed an attack categorization to focus on vulnerabilities in the code of smart contracts. After analyzing 10 security tools to detect vulnerabilities in order to assess their effectiveness, we found that not all vulnerabilities were detected, providing a dangerous false sense of security that attackers can abuse.

Our research points out that a proper solution to secure smart contracts remains a challenge and future work will involve developing strategies to detect and mitigate the major security flaws presented in this paper.

REFERENCES

[1] A. Rosic. (2016). *What is Blockchain Technology? A Step-by-Step Guide For Beginners*. Accessed: Jul. 29, 2018. [Online]. Available: https://blockgeeks.com/guides/what-is-blockchain-technology/

[2] A. Rosic. (2016). *What is Ethereum?* Accessed: Oct. 17, 2019. [Online]. Available: https://blockgeeks.com/guides/ethereum/

[3] J. A. Kassem, S. Sayeed, H. Marco-Gisbert, Z. Pervez, and K. Dahal, "DNS-IdM: A blockchain identity management system to secure personal data sharing in a network," *Appl. Sci.*, vol. 9, no. 15, p. 2953, Jul. 2019.

[4] (2009). *How does Bitcoin Work?* Accessed: Jul. 29, 2017. [Online]. Available: https://bitcoin.org/en/how-it-works

[5] A. Rosic. (2016). *Smart Contracts: The Blockchain Technology That Will Replace Lawyers*. Accessed: Aug. 3, 2019. [Online]. Available: https://blockgeeks.com/guides/smart-contracts/

[6] S. Velu. (2019). *What Are Dapps? The New Decentralized Future*. Accessed: Oct. 14, 2019. [Online]. Available: https://blockgeeks.com/guides/dapps/

[7] F. Ma, Y. Fu, M. Ren, M. Wang, Y. Jiang, K. Zhang, H. Li, and X. Shi, "EVM: From offline detection to online reinforcement for Ethereum virtual machine," in *Proc. IEEE 26th Int. Conf. Softw. Anal., Evol. Reeng. (SANER)*, Feb. 2019, pp. 554–558.

[8] Y. Fu, M. Ren, F. Ma, H. Shi, X. Yang, Y. Jiang, H. Li, and X. Shi, "EVMFuzzer: Detect EVM vulnerabilities via fuzz testing," in *Proc. 27th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.-ESEC/FSE*, 2019, pp. 1110–1114.

[9] I. C. Lin and T. C. Liao, "A survey of blockchain security issues and challenges," *Int. J. Netw. Secur.*, vol. 19, no. 5, pp. 653–659, 2017.

[10] A. Soundararajan. (2019). *10 Blockchain and New Age Security Attacks You Should Know*. Accessed: Jul. 29, 2018. [Online]. Available: https://blogs.arubanetworks.com/solutions/10-blockchain-and-new-age-security-attacks-you-should-know/

[11] N. Atzei, M. Bartoletti, and T. Cimoli, "A survey of attacks on ethereum smart contracts," in *Proc. Int. Conf. Princ. Secur. Trust*, 2017, pp. 164–186.

[12] Pluralsight. (2015). *What's Difference Between Front-End Back-End?* Accessed: Jul. 29, 2018. [Online]. Available: https://www.pluralsight.com/blog/film-games/whats-difference-front-end-back-end

[13] M. Andreessen. *What is Blockchain Technology?* Accessed: Sep. 25, 2019. [Online]. Available: https://www.coindesk.com/information/what-is-blockchain-technology

[14] A. Rosic. (2016). *What is Blockchain Technology? A Step-by-Step Guide For Beginners*. Accessed: Sep. 27, 2019. [Online]. Available: https://blockgeeks.com/guides/what-is-blockchain-technology/

[15] D. Cosset. (2018). *Blockchain: What is Mining?* Accessed: Jul. 29, 2018. [Online]. Available: https://dev.to/damcosset/blockchain-what-is-mining-2eod

[16] A. Hertig. (2019). *How Ethereum Mining Works*. Accessed Oct. 20, 2019. [Online]. Available: https://www.coindesk.com/information/ethereum-mining-works
[17] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," Cryptology ePrint Archive, Santa Barbara, CA, USA, Tech. Rep. 2015/675, 2015. [Online]. Available: https://eprint.iacr.org/2015/675
[18] H. Kalodner, S. Goldfeder, X. Chen, S. M. Weinberg, and E. W. Felten, "Arbitrum: Scalable, private smart contracts," in *Proc. 27th {USENIX} Secur. Symp. ({USENIX} Security)*, 2018, pp. 1353–1370.
[19] P. Das, L. Eckey, T. Frassetto, D. Gens, K. Hostáková, P. Jauernig, S. Faust, and A.-R. Sadeghi, "Fastkitten: Practical smart contracts on bitcoin," in *Proc. 28th USENIX Secur. Symp. (USENIX Security)*, 2019, pp. 801–818.
[20] J. M. Duffy. (2019). *Connecting Ethereum, EOS, Tron: Making Blockchain Interoperability a Reality*. Accessed: Oct. 29, 2019. [Online]. Available: https://medium.com/loom-network/connecting-ethereum-eos-and-tron-making-blockchain-interoperability-a-reality-e5ef6c67716
[21] Bitcoin Magazine. (2019). *What Is Ether?* Accessed: Sep. 12, 2019. [Online]. Available: https://bitcoinmagazine.com/guides/what-ether
[22] Freshfields. (2019). *What's a Smart Contract?* Accessed: Oct. 23, 2019. [Online]. Available: https://www.freshfields.com/en-gb/our-thinking/campaigns/digital/fintech/whats-in/whats-in-a-smart-contract/
[23] Prasanna. (2019). *What is Ethereum Virtual Machine?* Accessed: Oct. 23, 2019. [Online]. Available: https://cryptoticker.io/en/ethereum-virtual-machine/
[24] L. Hollander. (2019). *The Ethereum Virtual Machine—How does it work?* Accessed: Oct. 26, 2019. [Online]. Available: https://medium.com/mycrypto/the-ethereum-virtual-machine-how-does-it-work-9abac2b7c9e
[25] D. Siegel. (2016). *Understanding The DAO Attack*. Accessed: Sep. 27, 2019. [Online]. Available: https://www.coindesk.com/understanding-dao-hack-journalists
[26] S. Falkon. (2017). *The Story of the DAO—Its History and Consequences*. Accessed: Oct. 19, 2018. [Online]. Available: https://medium.com/swlh/the-story-of-the-dao-its-history-and-consequences-71e6a8a551ee
[27] S. Palladino. (2017). *The Parity Wallet Hack Explained*. Accessed: Oct. 20, 2019. [Online]. Available: https://blog.openzeppelin.com/on-the-parity-wallet-multisig-hack-405a8c12e8f7/
[28] B. Mueller. (2019). *What Caused Accidental Killing Parity Multisig Wallet & How to Detect Similar Bugs*. Accessed: Oct. 25, 2019. [Online]. Available: https://hackernoon.com/what-caused-the-latest-100-million-ethereum-bug-and-a-detection-tool-for-similar-bugs-7b80f8ab7279
[29] M. Condon. (2019). *Parity Wallet Hack 2: Electric Boogaloo*. Accessed: Oct. 20, 2019. [Online]. Available: https://hackernoon.com/parity-wallet-hack-2-electric-boogaloo-e493f2365303
[30] J. J. Xu, "Are blockchains immune to all malicious attacks?" *Financial Innov.*, vol. 2, no. 1, pp. 1–9, 2016.
[31] S. Sayeed and H. Marco-Gisbert, "On the effectiveness of blockchain against cryptocurrency attacks," in *Proc. UBICOMM*, 2018, pp. 9–14.
[32] B. Bambrough. (2019). *Warning Issued After Malware Is Found To Have Hijacked Bitcoin Blockchain*. Accessed: Oct. 29, 2019. [Online]. Available: https://www.forbes.com/sites/billybambrough/2019/09/07/serious-malware-warning-over-bitcoin-blockchain/#cc2d8347c286
[33] S. Sayeed and H. Marco-Gisbert, "Assessing blockchain consensus and security mechanisms against the 51% attack," *Appl. Sci.*, vol. 9, no. 9, p. 1788, Apr. 2019.
[34] M. del Castillo. (2019). *Customers Can Spend Bitcoin At Starbucks, Nordstrom and Whole Foods, Whether They Like It Or Not*. Accessed: Aug. 27, 2019. [Online]. Available: https://www.forbes.com/sites/michaeldelcastillo/2019/05/13/starbucks-nordstrom-and-whole-foods-now-accept-bitcoin-just-dont-ask-them/659a4e592252
[35] O. G. Güçlütürk. (2018). *The DAO Hack Explained: Unfortunate Take-off Smart Contracts*. Accessed: Oct. 19, 2019. [Online]. Available: https://medium.com/@ogucluturk/the-dao-hack-explained-unfortunate-take-off-of-smart-contracts-2bd8c8db3562
[36] W. Shahda. (2019). *Protect Your Solidity Smart Contracts From Reentrancy Attacks*. Accessed: Oct. 5, 2019. [Online]. Available: https://medium.com/coinmonks/protect-your-solidity-smart-contracts-from-reentrancy-attacks-9972c3af7c21
[37] J. Gao, H. Liu, C. Liu, Q. Li, Z. Guan, and Z. Chen, "Easyflow: Keep Ethereum away from overflow," in *Proc. 41st Int. Conf. Softw. Eng., Companion*. Piscataway, NJ, USA: IEEE Press, 2019, pp. 23–26.
[38] S. Sayeed and H. Marco-Gisbert, "On the effectiveness of control-flow integrity against modern attack techniques," in *Proc. ICT Syst. Secur. Privacy Protection*, G. Dhillon, F. Karlsson, K. Hedström, and A. Zúquete, Eds. Cham, Switzerland: Springer, 2019, pp. 331–344.
[39] L. Y. Thanh. (2018). *Prevent Integer Overflow Ethereum Smart Contracts*. Accessed: Jun. 19, 2019. [Online]. Available: https://medium.com/@yenthanh/prevent-integer-overflow-in-ethereum-smart-contracts-a7c84c30de66
[40] Blockgeeks. (2018). *Understanding Overflow and Underflow Attacks on Smart Contracts*. Accessed: Jul. 19, 2019. [Online]. Available: https://blockgeeks.com/guides/underflow-attacks-smart-contracts/
[41] A. Bryk. (2018). *Blockchain Attack Vectors: Vulnerabilities Most Secure Technology*. Accessed: Sep. 14, 2019. [Online]. Available: https://www.apriorit.com/dev-blog/578-blockchain-attack-vectors
[42] S. Esra. (2018). *ICO Smart Contract Vulnerability: Short Address Attack*. Accessed: Oct. 14, 2019. [Online]. Available: https://medium.com/huzzle/ico-smart-contract-vulnerability-short-address-attack-31ac9177eb6b
[43] A. Manning. (2018). *Solidity Security: Comprehensive List Known Attack Vectors Common Anti-Patterns*. Accessed: Jul. 19, 2019. [Online]. Available: https://blog.sigmaprime.io/solidity-security.html
[44] S. Pro. (2019). *Smart Contract Security Issues: What are Smart Contract Vulnerabilities How to Protect*. Accessed: Sep. 19, 2019. [Online]. Available: https://smartym.pro/blog/smart-contract-security-issues-smart-contract-vulnerabilities-and-how-to-protect/
[45] A. Das, S. Balzer, I. Santurkar, J. Hoffmann, and F. Pfenning, "Resource-aware session types for digital contracts," 2019, *arXiv:1902.06056*. [Online]. Available: https://arxiv.org/abs/1902.06056
[46] H. Olickel. (2016). *Why Smart Contracts Fail: Undiscovered Bugs What We Can do About Them*. Accessed: Jul. 29, 2019. [Online]. Available: https://medium.com/hrishiolickel/why-smart-contracts-fail-undiscovered-bugs-and-what-we-can-do-about-them-119aa2843007
[47] J. Feist, G. Grieco, and A. Groce, "Slither: A static analysis framework for smart contracts," in *Proc. IEEE/ACM 2nd Int. Workshop Emerg. Trends Softw. Eng. Blockchain (WETSEB)*, May 2019, pp. 8–15.
[48] S. Sayeed, H. Marco-Gisbert, I. Ripoll, and M. Birch, "Control-flow integrity: Attacks and protections," *Appl. Sci.*, vol. 9, no. 20, p. 4229, Oct. 2019, doi: 10.3390/app9204229.
[49] (2019). *MythX: Smart Contract Security Tool for Ethereum*. Accessed: Oct. 24, 2019. [Online]. Available: https://mythx.io/
[50] B. Mueller. (2019). *Practical Smart Contract Security Analysis and Exploitation*. Accessed: Oct. 25, 2019. [Online]. Available: https://medium.com/hackernoon/practical-smart-contract-security-analysis-and-exploitation-part-1-6c2f2320b0c
[51] HaloBlock Official. (2018). *Introduction to Manticore, a Symbolic Analysis Tool for Smart Contract*. Accessed: Oct. 26, 2019. [Online]. Available: https://medium.com/haloblock/introduction-to-manticore-a-symbolic-analysis-tool-for-smart-contract-9de08dae4e1e
[52] M. Mossberg, F. Manzano, E. Hennenfent, A. Groce, G. Grieco, J. Feist, T. Brunson, and A. Dinaburg, "Manticore: A user-friendly symbolic execution framework for binaries and smart contracts," 2019, *arXiv:1907.03890*. [Online]. Available: https://arxiv.org/abs/1907.03890
[53] P. Tsankov, A. M. Dan, D. Drachsler-Cohen, A. Gervais, F. Buenzli, and M. T. Vechev, "Securify: Practical security analysis of smart contracts," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, New York, NY, USA: ACM, Oct. 2018, pp. 67–82.
[54] Securify. (2018). *Securify: Security Scanner for Ethereum Smart Contracts*. Accessed: Oct. 26, 2019. [Online]. Available: https://securify.chainsecurity.com/
[55] ChainSecurity. (2018). *Securify is Now GitHub*. Accessed: Oct. 10, 2019. [Online]. Available: https://medium.com/chainsecurity/securify-is-now-on-github-d3bec281eafc
[56] S. Tikhomirov, E. Voskresenskaya, I. Ivanitskiy, R. Takhaviev, E. Marchenko, and Y. Alexandrov, "SmartCheck: Static analysis of Ethereum smart contracts," in *Proc. 1st Int. Workshop Emerging Trends Softw. Eng. Blockchain-WETSEB*, 2018, pp. 9–16.
[57] SmartDec. (2018). *SmartCheck*. Accessed: Oct. 27, 2019. [Online]. Available: https://tool.smartdec.net/
[58] (2018). *Echidna, a Smart Fuzzer for Ethereum*. Accessed: Sep. 10, 2019. [Online]. Available: https://blog.trailofbits.com/2018/03/09/echidna-a-smart-fuzzer-for-ethereum/
[59] J. Feist. (2019). *Watch Your Language: Our First Vyper Audit*. Accessed: Nov. 11, 2019. [Online]. Available: https://securityboulevard.com/2019/10/watch-your-language-our-first-vyper-audit/

[60] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.-CCS*, New York, NY, USA, 2016, pp. 254–269, doi: 10.1145/2976749.2978309.

[61] L. Brent, A. Jurisevic, M. Kong, E. Liu, F. Gauthier, V. Gramoli, R. Holz, and B. Scholz, "Vandal: A scalable security analysis framework for smart contracts," Sep. 2018, *arXiv:1809.03981*. [Online]. Available: https://arxiv.org/abs/1809.03981

[62] S. Kalra, S. Goel, M. Dhawan, and S. Sharma, "ZEUS: Analyzing safety of smart contracts," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2018, pp. 1–12.

[63] S. Jarzabek, A. Poniszewska-Marańda, and L. Madeyski, *Integrating Research and Practice in Software Engineering* (Studies in computational intelligence). Cham, Switzerland: Springer, 2019. [Online]. Available: https://books.google.co.uk/books?id=LR2nDwAAQBAJ

[64] S. Bomko. (2019). *Detecting Critical Smart Contract Vulnerabilities with re:MythX*. Accessed: Oct. 19, 2019. [Online]. Available: https://medium.com/@sergiibomko/detecting-critical-smart-contract-vulnerabilities-with-re-mythx-c543615bc216

[65] I. Goldberg and T. Moore, *Financial Cryptography and Data Security: 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18–22, 2019, Revised Selected Papers* (Lecture Notes in Computer Science). Cham, Switzerland: Springer, 2019. [Online]. Available: https://books.google.co.uk/books?id=Gcm1DwAAQBAJ

[66] M. Di Angelo and G. Salzer, "A survey of tools for analyzing Ethereum smart contracts," in *Proc. IEEE Int. Conf. Decentralized Appl. Infrastruct. (DAPPCON)*, Apr. 2019, pp. 69–78.

[67] Enigmatic. (2019). *Using Securify for Safer Smart Contracts*. Accessed: Oct. 20, 2019. [Online]. Available: https://medium.com/coinmonks/using-securify-for-safer-smart-contracts-8d59de22a762

[68] J. Salwan. (2013). *Taint Analysis and Pattern Matching With Pin*. Accessed: Jul. 19, 2019. [Online]. Available: http://shell-storm.org/blog/Taint-analysis-and-pattern-matching-with-Pin/

[69] A. Dika, "Ethereum smart contracts: Security vulnerabilities and security tools," M.S. thesis, NTNU, Trondheim, Norway, 2017.

[70] B. Jiang, Y. Liu, and W. K. Chan, "ContractFuzzer: Fuzzing smart contracts for vulnerability detection," in *Proc. 33rd ACM/IEEE Int. Conf. Automated Softw. Eng.-ASE*, 2018, pp. 259–269.

[71] M. Fu, L. Wu, Z. Hong, F. Zhu, H. Sun, and W. Feng, "A critical-path-coverage-based vulnerability detection method for smart contracts," *IEEE Access*, vol. 7, pp. 147327–147344, 2019.

[72] D. Guido. (2018). *Echidna, Basic Echidna Usage*. Accessed: Sep. 11, 2019. [Online]. Available: https://github.com/crytic/slightly-smarter-contracts/wiki/echidna

[73] N. Grech, L. Brent, B. Scholz, and Y. Smaragdakis, "Gigahorse: Thorough, declarative decompilation of smart contracts," in *Proc. 41st Int. Conf. Softw. Eng.* Piscataway, NJ, USA: IEEE Press, 2019, pp. 1176–1186.

[74] N. Grech, M. Kong, A. Jurisevic, L. Brent, B. Scholz, and Y. Smaragdakis, "MadMax: Surviving out-of-gas conditions in Ethereum smart contracts," *Proc. ACM Program. Lang.*, vol. 2, no. OOPSLA, pp. 1–27, Oct. 2018.

[75] M. Rodler, W. Li, G. O. Karame, and L. Davi, "Sereum: Protecting existing smart contracts against re-entrancy attacks," 2018, *arXiv:1812.05934*. [Online]. Available: https://arxiv.org/abs/1812.05934

SARWAR SAYEED received the bachelor's degree in computing from the University of East London, U.K., the master's degree in IT from Cardiff Metropolitan University, U.K., and the MBA degree from Anglia Ruskin University, U.K. He is currently a Ph.D. researcher with the University of the West of Scotland, U.K. His research interests involve control-flow integrity, blockchain security, and blockchain based-attacks. He has published a few articles related to blockchain and control-flow integrity. He has also participated in a few research projects as a co-investigator.

HECTOR MARCO-GISBERT (Senior Member, IEEE) received the Ph.D. degree in computer science, cybersecurity from the Universitat Politecnica de Valencia, Spain. He was a Research Associate at the Universitat Politecnica de Valencia, where he co-founded the "Cybersecurity Research Group." He is currently an Associate Professor and a Cybersecurity Researcher with the University of the West of Scotland, U.K. He was a part of the team developing the multiprocessor version of the XtratuM hypervisor to be used by the European Space Agency in its space crafts. He has participated in multiple research projects as a Principal Investigator and Co-Investigator. He is the author of many articles of computer security and cloud computing. He has been invited multiple times to reputed cybersecurity conferences such as Black Hat and DeepSec. He has published more than ten Common Vulnerabilities and Exposures (CVE) affecting important software such as the Linux kernel. He is a member of the Engineering and Physical Sciences Research Council (EPSRC), U.K. He has received honors and awards from Google, Packet Storm Security, and IBM for his security contributions to the design and implementation of the Linux ASLR.

TOM CAIRA received the B.Sc. degree in computing science from the University of Glasgow, U.K., and a Postgraduate Diploma in advanced research and professional practice from the University of the West of Scotland, U.K. He is currently completing a Professional Doctorate with the University of the West of Scotland. He is also a Senior Lecturer and Researcher with the University of the West of Scotland, U.K. His research interests include network and data security, data governance, personal data stores, business process improvement, and digital transformation.