ÆGIS: Smart Shielding of Smart Contracts

Christof Ferreira Torres*
SnT, University of Luxembourg
Luxembourg, Luxembourg
christof.torres@uni.lu

Mathis Baden
SnT, University of Luxembourg
Luxembourg, Luxembourg
mathis.steichen@uni.lu

Robert Norvill
SnT, University of Luxembourg
Luxembourg, Luxembourg
robert.norvill@uni.lu

Hugo Jonker
[1]Open University of the Netherlands
Heerlen, Netherlands
[2]Radboud University
Nijmegen, Netherlands
hugo.jonker@ou.nl

ABSTRACT

In recent years, smart contracts have suffered major exploits, losing millions of dollars. Unlike traditional programs, smart contracts cannot be updated once deployed. Though various tools were proposed to detect vulnerable smart contracts, they all fail to protect contracts that have already been deployed on the blockchain. Moreover, they focus on vulnerabilities, but do not address scams (e.g., honeypots). In this work, we introduce ÆGIS, a tool that shields smart contracts and users on the blockchain from being exploited. To this end, ÆGIS reverts transactions in real-time based on pattern matching. These patterns encode the detection of malicious transactions that trigger exploits or scams. New patterns are voted upon and stored via a smart contract, thus leveraging the benefits of tamper-resistance and transparency provided by blockchain. By allowing its protection to be updated, the smart contract acts as a smart shield.

CCS CONCEPTS

• **Security and privacy → Software and application security**; *Domain-specific security and privacy architectures*; Systems security.

KEYWORDS

Ethereum, smart contracts, exploit prevention, security updates

ACM Reference Format:
Christof Ferreira Torres, Mathis Baden, Robert Norvill, and Hugo Jonker. 2019. ÆGIS: Smart Shielding of Smart Contracts. In *2019 ACM SIGSAC Conference on Computer andCommunications Security (CCS '19), November 11–15, 2019, London, UK*. ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

*Corresponding author.

CCS '19, November 11–15, 2019, London, UK

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Since the inception of Bitcoin [7], a broad range of blockchain implementations have emerged. Ethereum [13] is currently the most popular blockchain technology with respect to smart contracts. Smart contracts are programs that are stored and executed across blockchain nodes. They are deployed and invoked via transactions. Deployed smart contracts are immutable, but still prone to bugs. Moreover, since contract owners are anonymous, responsible disclosure is usually infeasible. Though smart contracts can be implemented with upgradeability and destroyability in mind, this is not compulsory. In fact, Ethereum already faced several devastating attacks on vulnerable smart contracts. In 2016, an attacker exploited a reentrancy bug in a crowdfunding smart contract called the DAO, draining over $150 million [10]. In 2017, the Parity wallet was hacked twice due to a logic bug in the access control of the smart contract, causing a combined loss of over $130 million [8]. In 2018, a blockchain security company called PeckShield reported that multiple smart contracts have been attacked or are vulnerable to integer overflows [3]. In 2019, Torres et al. reported an emerging trend among scammers, that try to lure their victims into traps by deploying seemingly vulnerable contracts that in reality contain hidden traps (i.e. honeypots), making users lose their funds if they attempt to exploit or interact with the smart contract [12].

In response to these events, academia proposed a plethora of different tools that allow users to scan smart contracts for vulnerabilities and scams, prior to deploying them on the blockchain or interacting with them (see e.g. [4, 5, 11, 12]). However, all of these tools fail to protect inattentive users and contracts that have already been deployed on the blockchain. In order to protect already deployed contracts, Rodler et al. [9] leverage the principle that every exploit is performed via a transaction. They propose Sereum, a modified Ethereum client that detects and reverts transactions that trigger reentrancy attacks. Unfortunately, Sereum has three major drawbacks. First, it solely detects reentrancy attacks, despite there being many other types of vulnerabilities and scams. Second, it requires the client to be modified whenever a new type of vulnerability or scam is found. Third, not only the tool itself but also any updates to it must be manually adopted by the majority of nodes for its security provisions to become effective.

Contributions. Our main contributions are:

- We introduce a novel domain specific language (DSL), which enables the description of *vulnerability patterns*. These patterns reflect malicious control and data flows that occur during execution of malicious Ethereum transactions.
- We present a tool called ÆGIS, that reverts[1] malicious transactions based on vulnerability patterns, thereby preventing attacks on vulnerable smart contracts.
- We propose the use of a smart contract to store and vote upon new vulnerability patterns, in order to quickly and safely propagate security updates, without relying on client-side update mechanisms. This ensures integrity, brings democracy and provides full transparency on the proposed vulnerability patterns.

2 BACKGROUND

The Ethereum blockchain is a decentralized public ledger that is maintained by a network of nodes that distrust one another. Every node runs one of several existing Ethereum clients, for example geth[2]. With the use of these clients, users can send transactions in order to create and invoke smart contracts. Transactions are broadcast through the blockchain network and are processed by *miners*. These are a specific type of nodes that propose new blocks and execute smart contracts via the Ethereum Virtual Machine (EVM). The EVM is a purely stack-based, register-less virtual machine that supports a Turing-complete instruction set of opcodes. These opcodes allow smart contracts to perform memory operations and interact with the blockchain, such as retrieving specific information (e.g., the current block number). Ethereum makes use of *gas* to make sure that contracts terminate and to prevent denial-of-services attacks. Thus, it assigns a cost to the execution of an opcode. The execution of a smart contract results in the modification of its state. The latter is stored on the blockchain and consists of a *balance* and a *storage*. The balance represents the amount of ether (Ethereum's cryptocurrency) currently owned by the smart contract. The storage is organized as a key-value store and allows the smart contract to store values and keep state across executions. In summary, the EVM is a transaction-based state machine that updates a smart contract based on transaction input data and the smart contract's bytecode.

3 RELATED WORK

Ethereum smart contracts are programs that are executed across the Ethereum blockchain. Unfortunately, as with any program, they may contain bugs and can be vulnerable to exploitation. As discussed in [1], different types of vulnerabilities exist, often leading to financial gains for the attacker. The issue is made worse by the fact that smart contracts are immutable. Once deployed, they cannot be altered and vulnerabilities cannot be fixed. In addition to that, automated tools for launching attacks exist [4]. Several defense mechanisms have been proposed (e.g. [5, 11, 12]). However, while these tools identify vulnerabilities and scams, they cannot protect already deployed smart contracts from being exploited or neglectful users from falling for scams. Therefore, to deal with the issue of vulnerabilities in deployed smart contracts, [9] proposes a

[1]Consuming gas, without letting the transaction affect the state of the blockchain.
[2]For more details about the geth client: https://github.com/ethereum/go-ethereum.

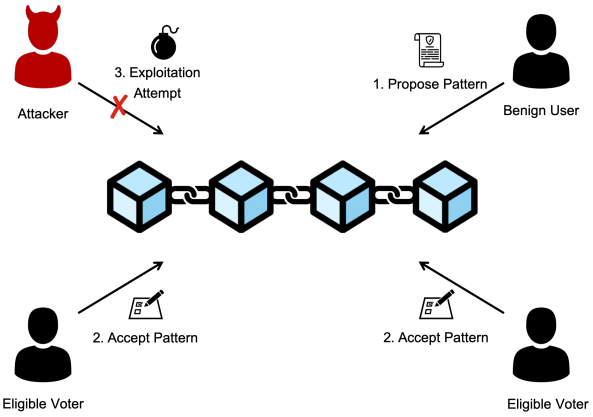

Figure 1: An illustrative example of ÆGIS's workflow: Step 1) A benign user proposes a pattern to the smart contract. Step 2) Eligible voters vote to either accept or reject the pattern. Step 3) An attacker fails to exploit a vulnerable smart contract due to the voted pattern matching the malicious transaction.

modification to the Ethereum client, geth. However, this approach only deals with one type of attacks, reentrancy, and requires all the clients in the network to be modified. The latter is an issue for the following reasons. On one hand side, every update of the vulnerability detection software requires an update of the different Ethereum client implementations. This is true for both, bug fixes and functionality upgrades, for example the detection of new vulnerabilities. On the other hand side, every modification of the clients needs to be adopted by all the nodes participating in the Ethereum blockchain. This can take time and breaks compatibility between updated and non-updated clients. In this work, we propose a generic solution that only requires clients to be modified once and that protects contracts and users from existing and future vulnerabilities, without modifying the clients every time a new scam or vulnerable smart contract is found.

4 METHODOLOGY

Our idea is to bundle every Ethereum client with a modified EVM capable of interpreting a DSL. The DSL is specifically tailored to the EVM instruction set and allows the description of malicious control and data flows in the form of patterns. The modified EVM can then revert transactions during execution based on pattern matching. For example, a malicious integer overflow could be described as the following pattern:

$$(\texttt{opcode} = \texttt{CALLDATALOAD}) \xrightarrow{\text{data}} (\texttt{opcode} = \texttt{ADD}) \wedge (\texttt{stack}[0] + \texttt{stack}[1] \neq \texttt{stack.result}) \xrightarrow{\text{data}} (\texttt{opcode in } [\texttt{SSTORE}, \texttt{CALL}])$$

This pattern evaluates to true if a transaction meets all of the following three conditions: 1) there is a data flow from a CALLDATALOAD instruction into an ADD instruction; 2) the result of the addition pushed by the EVM onto the stack is different from the sum of the two previous stack elements; and 3) there is a data flow of the result into either an SSTORE or a CALL instruction.

Table 1: Ethereum Top 10 Miners by Blocks*

Miner	# Blocks	% of all mined blocks
Spark Pool	24,261	24.42%
Ethermine	22,579	23.66%
F2Pool 2	11,373	11.92%
Nanopool	10,579	11.08%
MiningPoolHub	4,422	4.63%
zhizhu.top	3,377	3.54%
0xd224ca…b79f53	1,739	1.82%
PandaMiner	1,519	1.59%
0xaa5c42…acf05e	1,419	1.49%
xnpool	1,373	1.44%

*Source: Etherscan.io, August 2019

Patterns are governed via a smart contract that is deployed on-chain (see Figure 1). Whenever a new vulnerability or scam is discovered, anyone may write a new pattern using the DSL and propose it through the smart contract. The contract maintains a list of eligible voters that vote for either accepting or rejecting a new pattern. If the majority has voted with "yes", then the pattern is added to the list of active patterns. In that case, every client is automatically notified and retrieves the updated list of patterns from the smart contract (i.e. the blockchain). Thus, clients are updated independently. The only requirement is a one-time client-side update of the EVM to add the capability of processing patterns expressed in the DSL. In other words, if a pattern is accepted by the voting mechanism, it is updated instantaneously across all the clients through the existing consensus mechanism of the Ethereum blockchain.

5 DISCUSSION

Setting up the list of eligible voters is crucial, as these will have the power to decide on the result of transactions. One option is to choose miners as eligible voters, as these already carry a powerful role in deciding which transactions are to be included into blocks. However, miners may lack incentives to stop specific vulnerabilities or scams, as well as the expertise to decide on which patterns to be added or rejected. Moreover, as seen in Table 1, miners should not be given voting power according to their mining power. Otherwise, a small fraction of top miners could collude and together control more than 50% of the votes. Therefore, it may be better to select a group of independent security experts, such as the members of the Smart Contract Weakness Classification registry (SWC)[3].

In contrast to democratic elections, where privacy and verifiability are paramount, a voting system for vulnerabilities would require *accountability*. In this light, it is more akin to 'boardroom' e-voting systems than 'parliamentary' e-voting systems. Such systems allow for tracing an unencrypted vote back to a voter, which, in turn, enables a constituency (e.g., shareholders) to hold the voter accountable for their voting choices. Another important requirement is that of *fairness*: the requirement that each voter should have equal power to affect the outcome, regardless of when they vote. One aspect of this is to prevent leaking of intermediate results (which would give later voters more information on the effect of their vote). This can be achieved through the use of cryptographic commitments [2, 6]. Of course, this still lets later voters see the number of votes cast, which reveals whether or not the result could potentially be a tie. This violates fairness: the last voter can choose to vote to create or break a potential tie. To mitigate these concerns, voting should be done off-chain with a strong protocol. Moreover, this protocol should result in a a proof of correct execution of the voting process, which is then stored on-chain to fulfill the accountability requirement.

[3]For more details see: https://smartcontractsecurity.github.io/SWC-registry/.

6 CONCLUSION

We propose ÆGIS, a modified EVM that uses patterns in order to protect users from vulnerable smart contracts and scams. Moreover, we present a novel mechanism that allows patterns to be updated quickly and transparently via the blockchain. Finally, we discuss the challenges faced when decentralizing the governance of these patterns.

ACKNOWLEDGMENTS

This work is partly supported by the Luxembourg National Research Fund (FNR) under grant 13192291.

REFERENCES

[1] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. 2017. A Survey of Attacks on Ethereum Smart Contracts (SoK). In *Proceedings of the 6th International Conference on Principles of Security and Trust - Volume 10204*. Springer-Verlag New York, Inc., 164–186. https://doi.org/10.1007/978-3-662-54455-6_8

[2] Gilles Brassard, David Chaum, and Claude Crépeau. 1988. Minimum disclosure proofs of knowledge. *Journal of computer and system sciences* 37, 2 (1988), 156–189.

[3] PeckShield Inc. 2018. ALERT: New batchOverflow Bug in Multiple ERC20 Smart Contracts (CVE-2018-10299). https://blog.peckshield.com/2018/04/22/batchOverflow/.

[4] Johannes Krupp and Christian Rossow. 2018. teether: Gnawing at ethereum to automatically exploit smart contracts. In *27th USENIX Security Symposium (USENIX Security 18)*. 1317–1333.

[5] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. 2016. Making Smart Contracts Smarter. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16)*. ACM, New York, NY, USA, 254–269. https://doi.org/10.1145/2976749.2978309

[6] T. Moran and M. Naor. 2007. Split-ballot voting: everlasting privacy with distributed trust. In *Proc. 14th ACM Conference on Computer and Communications Security (CCS)*. ACM, 246–255.

[7] Satoshi Nakamoto. 2009. Bitcoin: A Peer-to-Peer Electronic Cash System. *Cryptography Mailing list at https://metzdowd.com* (03 2009).

[8] Sergey Petrov. 2017. Another Parity Wallet hack explained. https://medium.com/@Pr0Ger/another-parity-wallet-hack-explained-847ca46a2e1c.

[9] Michael Rodler, Wenting Li, Ghassan O. Karame, and Lucas Davi. 2019. Sereum: Protecting Existing Smart Contracts Against Re-Entrancy Attacks. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27*.

[10] David Siegel. 2016. Understanding The DAO Attack. https://www.coindesk.com/understanding-dao-hack-journalists/.

[11] Christof Ferreira Torres, Julian Schütte, and Radu State. 2018. Osiris: Hunting for Integer Bugs in Ethereum Smart Contracts. In *Proceedings of the 34th Annual Computer Security Applications Conference (ACSAC '18)*. ACM, New York, NY, USA, 664–676. https://doi.org/10.1145/3274694.3274737

[12] Christof Ferreira Torres, Mathis Steichen, and Radu State. 2019. The Art of The Scam: Demystifying Honeypots in Ethereum Smart Contracts. In *28th USENIX Security Symposium (USENIX Security 19)*. USENIX Association, Santa Clara, CA, 1591–1607.

[13] Gavin Wood. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper* 151 (2014), 1–32.