Wibx: Making Smart Contracts Even Smarter

Rafael Augusto Lopes Shigemura, Gildarcio Sousa Goncalves, Fábio Alves de Oliveira, Luiz Henrique Coura, Edizon Eduardo Basseto Júnior, Luiz Alberto Vieira Dias, Adilson Marques da Cunha, Paulo Marcelo Tasinaffo, Johnny Cardoso Marques

Computer Science Department
Brazilian Aeronautics Institute of Technology (Instituto Tecnológico de Aeronáutica – ITA)
São José dos Campos, São Paulo, Brazil

Abstract* – This paper describes the main results of a research effort involving techniques for automatic detection of security vulnerabilities on an Ethereum-based Smart Contract. During 16 weeks, at the Brazilian Aeronautics Institute of Technology (Instituto Tecnológico de Aeronáutica - ITA), a research-oriented version of Scrum agile method and its best practices took place. The Project, named Technological Solutions Applicable to Media and Social Products (in Portuguese Soluções Tecnológicas Aplicáveis a Mídias e Produtos Sociais) is being driven by a partnership between ITA and Ecossistema enterprise, in order to generate knowledge and expertise in blockchain related disciplines, as well to ground a brand new utility token named Wibx. The main contribution of this research branch (blockchain security) was the enhancement of the original Oyente tool, renaming it as Oyente-NG (New Generation), including the detection of 5 new relevant vulnerabilities beyond the 7 already previously implemented ones. Lastly, a Proof of Concept applying the Oyente-NG tool over a set of real contracts developed for Wibx is provided.

***Keywords* - Blockchain; Smart Contracts, Scrum Method; Intelligent Systems; Blockchain Security.**

I. INTRODUCTION

Since its introduction in 2008, Blockchain technology [1] has been promising in several aspects: removal of intermediaries, cost reduction of values transmissions, reliable and decentralized storage of transactions, alternative currencies, among others. These possibilities have generated global interest, but when it comes to value management, several issues emerge. Unlike other centralized forms of currency control such as credit cards, Blockchains is still in its beginning and there is plenty of room for potential fraudsters. Thus, it is essential to investigate appropriate strategies for overall vulnerability mitigation and security improvement.

Besides the Blockchain itself, a hot topic is the smart contract technology [2]. Basically, a smart contract is an agreement between mutually distrusting participant automatically enforced by the consensus mechanism of the blockchain, without relying on a trusted authority [3].

This attractive potential of automatic, decentralized, and trustworthy contract enforcement, along with standard blockchain capabilities, leveraged the third largest blockchain platform to date: Ethereum [4], whose capitalization has reached 132 billion dollars in January 2018, as shown by Figure 1.

Figure 1. The Ethereum Market Cap in 2018 [5].

Such a huge market cap, the related high volume of interested business investing resources on it, and the relative immaturity of underlying technology created a green field for hackers aiming to get financial or technological advantages.

One remarkable event was The DAO (Decentralized Autonomous Organization) Attack [6], where a crowdfunding contract, which raised ~150 million dollars, was hacked on June 18, 2016, and the attacker managed to take control over ~60 million dollars until the hard-fork of Ethereum main blockchain nullified the effects of the involved transactions.

Amidst this scenario, the Ecossistema enterprise has decided to develop and launch its own cryptocurrency, based on the Ethereum platform along with a private blockchain. This crypto, named Wibx [7][8], was planned to be a utility coin for mass usability, demanding high-level security and reliability.

The Ecossistema enterprise and the Brazilian Aeronautics Institute of Technology (Instituto Tecnologico de Aeronáutica - ITA) are undertaking a research-oriented version of Scrum agile method and its best practices [9][10]. The Project, named Technological Solutions Applicable to Media and Social Products (in Portuguese, Soluções Tecnológicas Aplicáveis a Mídias e Produtos Sociais) was conceived, in order to generate knowledge and expertise in blockchain related disciplines, as well to ground the Ecossistema`s Wibx cryptocurrency.

Thus, a research branch for blockchain security has emerged, and its main goal was to find novel techniques for vulnerability mitigation in Ethereum blockchains, in general, and Ethereum-based smart contracts, in special.

The main contribution of this research was the enhancement of the original Oyente [11] tool, named Oyente-NG (Oyente - New Generation), including the detection of 5 new relevant vulnerabilities, beyond the 7 already previously implemented ones. Lastly,a Proof of Concept applying the Oyente-NG tool over a set of real contracts developed for the Wibx utility coin is provided.

II. BACKGROUND

This section describes the following key concepts, methods, and techniques used for the development of the Technological Solutions Applicable to Media and Social Products project, named in Portuguese *Soluções Tecnológicas Aplicáveis a Mídias e Produtos Sociais* - STAMPS: the Blockchain technology; the Ethereum platform; the Smart Contract technology; the vulnerabilities in Ethereum-based Smart Contracts; and the Oyente tool.

A. The Blockchain Technology

Blockchain is a growing list of records, named blocks, which are linked by cryptography. Each block contains a cryptographic hash of the previous block, a timestamp, and transaction data (generally represented as a Merkle tree root hash) [1].

By design, a blockchain is resistant to modification of data. It is an open, distributed ledger that can efficiently record transactions between two parties in a verifiable and permanent way, as shown in Figure 2. For use as a distributed ledger, a blockchain is typically managed by a peer-to-peer network collectively adhering to a protocol for inter-node communication and validating new blocks. Once recorded, data in any given block cannot be altered retroactively without alteration of all subsequent blocks, which requires a consensus of the network majority. Although blockchain records are not unalterable, blockchains may be considered secure by design.

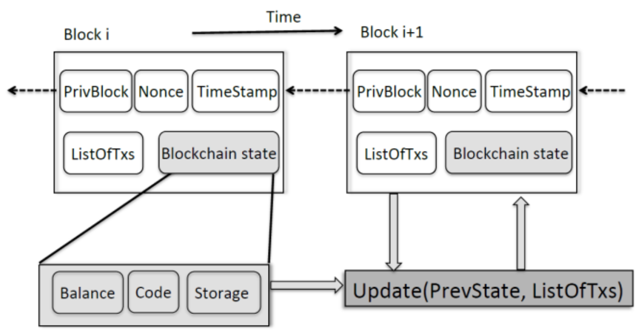

Figure 2. The most popular design of a blockchain, used by Bitcoin and Ethereum. Block with several elements [11].

Blockchain was invented by Satoshi Nakamoto in 2008 to serve as the public transaction ledger of the cryptocurrency bitcoin [1]. The invention of the blockchain for bitcoin made it the first digital currency to solve the double-spending problem without the need of a trusted authority or central server. The bitcoin design has inspired other applications [4], and blockchains, which are readable by the public are widely used by cryptocurrencies. Private blockchains have also been proposed for business use [12].

B. The Ethereum Platform

Ethereum is an open-source, public, blockchain-based distributed computing platform and operating system featuring smart contract (scripting) functionality. It supports a modified version of the Nakamoto consensus [1] via transaction-based state transitions [4].

The platform was initially described in a white paper by Vitalik Buterin [4], with a goal of building decentralized applications. Buterin had argued that bitcoin needed a scripting language for application development. Failing to gain agreement, he then proposed the development of a new platform with a more general scripting language.

Ethereum provides a decentralized virtual machine, the Ethereum Virtual Machine (EVM), which can execute scripts by using an international network of public nodes. The virtual machine's instruction set, in contrast to others like Bitcoin Script, is thought to be Turing-complete. The platform also provides an special concept called "Gas" for internal transaction pricing mechanism and is used to mitigate spam, remunerate miners, and allocate resources on the network [13].

C. The Smart Contract Technology

Basically, Smart Contracts are computer programs that can be correctly executed by a network of mutually distrusting nodes, without the need of an external trusted authority [3].

Conceptually, as explained by Szabo on its seminal paper [17]: "Smart contracts combine protocols with user interfaces to formalize and secure relationships over computer networks. Objectives and principles for the design of these systems are derived from legal principles, economic theory, and theories of reliable and secure protocols."

It makes clear that, since its principle, the main goal was to implement full-fledged, real-life contracts using software, but following some principles as confidence, impartiality, and automation.

Using cryptographic technology and other security mechanisms, Smart Contracts can secure many algorithmically specifiable relationships from breach or malicious interference by third parties, up to considerations of time, user interface, and completeness of the algorithmic specification [17].

Its potential application in important contracting areas, including credit, content rights management, payment systems, and contracts with bearer were perceived far before the Blockchain era. Ethereum leveraged the concept, implementing it natively throughout solidity, a Turing-complete language capable of generating programs running beside its blockchain, thus taking advantage of all its capabilities, as in Figure 3.

```

import "remix_tests.sol";
import "./ballot.sol";

contract test3 {

    Ballot ballotToTest;
    function beforeAll () public {
       ballotToTest = new Ballot(2);
    }

    function checkWinningProposal () public {
        ballotToTest.vote(1);
        Assert.equal(ballotToTest.winningProposal(), uint(1), "Pro1 won!");
    }

    function checkWinninProposalWithReturnValue () public view returns (bool) {
        return ballotToTest.winningProposal() == 1;
    }
}

```

Figure 3. An Ethereum smart contract that illustrates a simple voting system.

D. Vulnerabilities in Ethereum-based Smart Contracts

Despite being designed, in principle, for secure specifiable relationships from breach or malicious interference by third parties, this is not totally true at practice. In fact, even when implemented on top of another secure technology as blockchain, a considerable volume of vulnerabilities was discovered, putting at risk the assets and, therefore, the businesses relying on it.

In the past two years, several research papers were published about potential security vulnerabilities in Ethereum-based Smart Contracts [14][15][16]. Especially in Luu [11], it is found a useful taxonomy which, despite not been exhaustive, reflects the most prominent breaches. That is shown in Table 1, with one more vulnerability included: the Integer Underflow & Overflow.

One of the previous-mentioned vulnerabilities, The DAO Attack, was alone responsible for ~60 million dollars in losses [6]. Furthermore, a complicating factor is the immutability of smart contracts: there is yet no means to fix a buggy contract (like the DAO contract), and once it is published on the network, there is no way back.

Thus, it is evident the ROI in R&D of strategies and techniques for risk mitigation related to Smart Contracts, especially for companies aiming to allocate resources on it. To the Blockchain Security branch of the project it was given the task of researching, developing, and applying techniques, methods, and tools that would mitigate the risk that any of these vulnerabilities could be exploited on the Wibx crypto coin.

Table 1. Taxonomy of vulnerabilities in Ethereum-based Smart contracts, and known related attacks.

Cause of Vulnerability	Known Attacks
Call to the unknown	The DAO Attack
Gasless send	King of The Ether Throne (KoET)
Exception disorders	GovernMental, KoET
Type casts	-
Reentrancy	The DAO Attack
Keeping secrets	Multiplayer Games
Immutable bugs	Rubixi, GovernMental
Ether lost in transfer	-
Stack size limit	GovernMental
Unpredictable state	GovernMental, Dynamic Libraries
Generating randomness	-
Integer Underflow & Overflow	The Beauty Chain Attack
Parity Multisig Bug	The Parity Attack
Time constraints	GovernMental

E. The Oyente Tool

As demonstrated in [18], some of the vulnerabilities discussed in the previous section could be addressed, at their root cause, by improvements to the operational semantics of Ethereum. However, it would require analysis and approval of community and, thereafter, all clients in a network to upgrade.

As that option is virtually impracticable, there were provided a pre-deployment mitigation tool called Oyente [18] to help: 1. developers to write better contracts; and 2. users to avoid invoking problematic contracts. Importantly, other analyses can also be implemented as independent plugins, without interfering with the existing features.

Based upon symbolic execution [19], Oyente manages to represent program's concrete states as symbolic states. That symbolic state forms symbolic paths having path conditions, which can be proved satisfiable or unsatisfiable, thus confirming the path (and, consequently, the conditions) feasibility.

The main advantage of symbolic execution over traditional test approaches (like dynamic testing) is the capacity of reasoning about a program path-by-path (which is often a finite set), instead of reasoning input-by-input (which is often an infinite set). Symbolic execution can also be viewed as abstract interpretation [20], as shown in Figure 4.

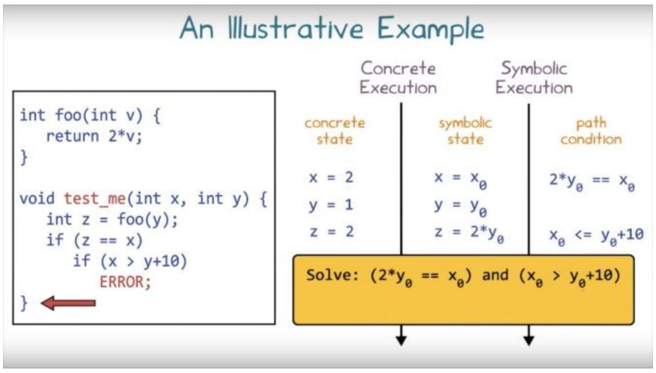

Figure 4. Illustrative example of a symbolic execution [21] .

Table 2 presents the open and addressed vulnerabilities, according the above-mentioned taxonomy.

Table 2. Vulnerabilities in Ethereum-based Smart contracts, and its mitigation status.

Cause of Vulnerability	Status
Call to the unknown	Open
Gasless send	Open
Exception disorders	Addressed by Oyente
Type casts	Open
Reentrancy	Addressed by Oyente
Keeping secrets	Open
Immutable bugs	Open
Ether lost in transfer	Open
Stack size limit	Addressed by Oyente
Unpredictable state	Addressed by Oyente
Generating randomness	Open
Integer Underflow & Overflow	Addressed by Oyente
Parity Multisig Bug	Addressed by Oyente
Time constraints	Addressed by Oyente

As shown in Figure 5, the Oyente architecture is modular and well suited for scalability. Briefly, the system takes as inputs to the program, be it as bytecode or source file, as well (and optionally) the blockchain global state. Then, the CFG (*Control-flow Graph*) Builder generates the control flow graph of the contract and passes it to Explorer, which will execute the simulation. Thereafter, the CORE ANALYSIS seeks for potentially problematic paths and query the Z3 Solver [22] for path feasibility. Finally, the VALIDATOR checks the flagged 'problematic' paths for possible false positives and the Visualizer shows the results in textual mode.

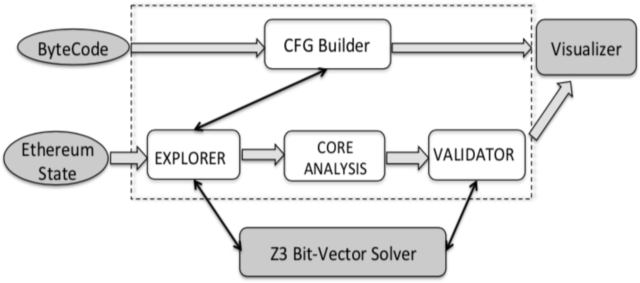

Figure 5. Oyente architecture and its main components inside the dotted rectangle. Shaded components publicly available [11].

Figure 6 shows an output from Oyente, running over a snippet purposely built with the reentrance vulnerability. There, we can see the results and its flags pointing to the presence or absence of 7 vulnerabilities originally detectable. The reentrance is flagged True, as expected.

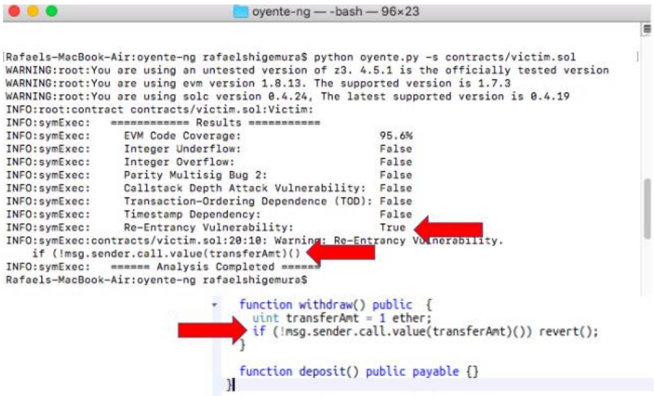

Figure 6. Oyente execution output over an example snippet containing the reentrance vulnerability.

To the best of our knowledge, this kind of approach for verification & validation of smart contracts is not implemented in any other blockchain platforms like Bitcoin [1], Corda [19], amongst others.

III. THE OYENTE-NG PROPOSED TOOL

This section describes the implementation of improvements in original Oyente, in order to detect a new set of vulnerabilities. It presents and briefs the targeted ones and, at the end, it explains the criteria adopted for the detection of each vulnerability.

A. The Targeted Vulnerabilities

Although the implemented analyses and their results [16] were quite relevant, they wouldn't be enough to achieve compliance with Wibx minimum security requirements. Thus, considering them and also following the proposed taxonomy, there were defined the need for 5 more automatic detections, the risks for which are presented in Table 3.

Table 3. Selected vulnerabilities for implementation in Oyente-NG.

Vulnerability	Known risks
Call to the unknown	Ether stealing
Gasless send	Ether overspending for transaction processing
Type casts	Unexpected contract behavior
Ether lost in transfer	Ether locking
Generating randomness	Ether stealing

The only vulnerability we were unable to implement was the Immutable Bug. It concerns to the impossibility of changing a contract once it is published onto Ethereum blockchain. The immutability has been exploited in various attacks [15] and, for all of then, the stolen ether was unrecoverable. Even having this important feature, there was not yet a definitive solution, though it is possible to obtain some insights in [23].

B. Implementation Overview

We describe how we have implemented our analyses as follows:

- *Call to the unknown detection* - We do analyse the symbolic trace of each called function, in order to infer possible mismatches between the called and the caller signatures. If some mismatch is found, the contract is flagged as potential call to unknown;
- *Gasless send detection* - Considering that each bytecode instruction has its gas consumption, we detect a potential high demanding gas contract by summing up the value of each instruction symbolically executed. If the amount of gas is greater than 2300, the contract is flagged as high demanding. This figure 2300 is the limit for gas units available to the calee and is considered a good estimator for gas-intensive contracts;
- *Type casts detection* - In analogue way to call to unknown, we detect dangerous type casts through called and caller function`s signature checking during symbolic execution;
- *Ether lost detection* - During the analysis stage, every address found in source file is checked according to Ethereum standards. A contract is flagged as potential Ether lost if some address is invalid; and
- *Randomness bug detection* - Detecting randomness bug is, at principle, straightforward. We only have to check if the contract executes some of random source instructions which, although well known, are all extremely dangerous for the assets involved with the contract.

IV. THE PROOF OF CONCEPT ON A REAL SCENARIO

This section describes the usage of the Oyente-NG on a real scenario. It shows the automatic detection of vulnerabilities unseen by the developers, even being experienced ones. At the end, it addresses the development and the main challenges faced by the research project.

A. Evaluation of the Wibx contract - Version 1

Although simple, the contract was found with 2 occurrences of Integer Overflow and a potential Out of Gas Send. Figure 7 shows the overall analysis and its results.

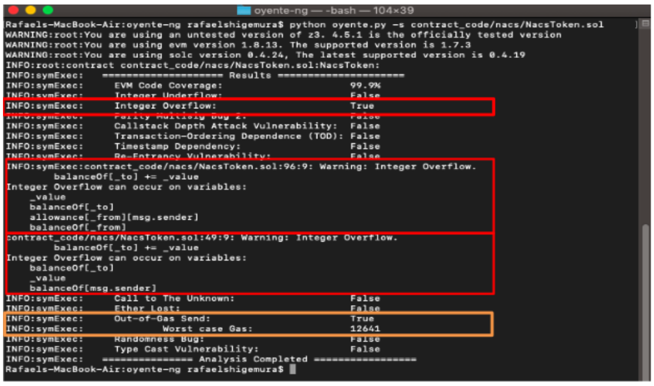

Figure 7. Output of Oyente-NG run over the first version of the Wibx contract, and detected vulnerabilities.

We can see on the above figure how the Oyente-NG flags the vulnerabilities found. On the first red box, there is a boolean declaring the presence or absence of each. Second and third red boxes show some details, pointing, in this case, the program variables susceptible to integer overflow. Finally, the orange box shows estimates for the worst case gas consumption

Figure 8 presents the detailed report of variables. For instance, it points that, on the line 96, column 9 of the source code, the variable *balanceOf[_to]* is prone to overflow. It makes easier and faster for the developer to validate and patch the issues.

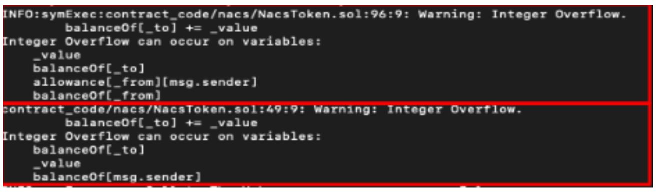

Figure 8. Integer overflow vulnerability detected, and respective places of occurence.

Figure 9 presents, in case of potential gasless send (also called out-of-gas send), an estimate for the worst case. With the intention of assuming transaction costs, it is essential for Wibx to minimize the gas usage of its contracts, in order to reduce operational costs and increase efficiency.

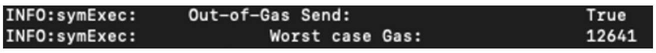

Figure 9. The worst case Gas estimates for the contract.

B. Evaluation of the Wibx contract - Version 2

The second version of Wibx contract is far more elaborated and complex, gathering the contract itself and a set of dependencies. It is essential to ensure the security of both, contract and dependencies, so will be shown the outputs of each file. Figure 10 shows the analysis of the dependency BCCHandled.sol, which was found with high gas demand.

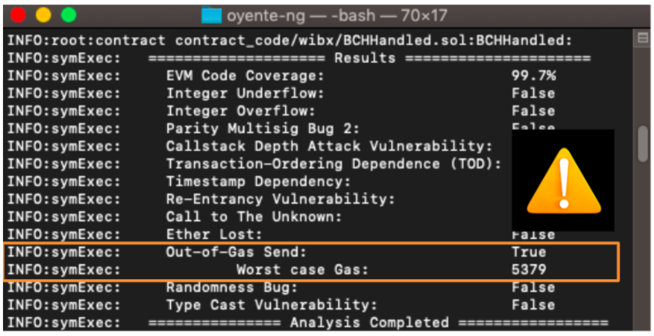

Figure 10. Evaluation of first dependence: BCCHandled.sol.

Figure 11 shows the analysis of the dependency ERC20.sol, which was also found with high gas demand.

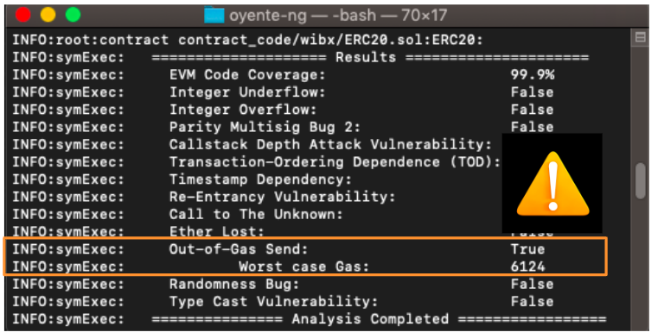

Figure 11. Evaluation of second dependence: ERC20.sol.

Figure 12 shows the analysis of the dependency SafeMath.sol, an almost ubiquitous library for safe arithmetic operations. It is possible to see that this file achieved the best possible evaluation, with no vulnerabilities found.

Figure 13 shows the analysis of the dependency TaxLib.sol, which also achieved the best possible evaluation, with no vulnerabilities found.

Finally, Figure 14 shows the analysis of the main contract: WibxToken.sol. It was found, besides the potential excessive gas use, an occurrence of integer underflow, even using the SafeMath library. It was later discovered that a missing and (not so well documented) importing command made the code vulnerable.

Although none of the new implemented vulnerabilities were found at provided contracts, we are planning a benchmark for using the main Ethereum network, in order to gather statistics about the actual state of affairs concerning these vulnerabilities.

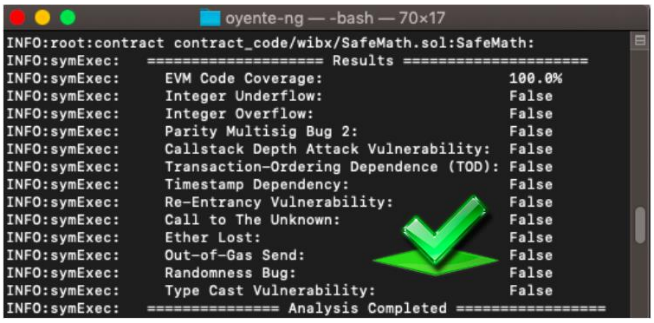

Figure 12. Evaluation of the third dependence: SafeMath.sol.

Figure 13. Evaluation of the fourth dependence: TaxLib.sol.

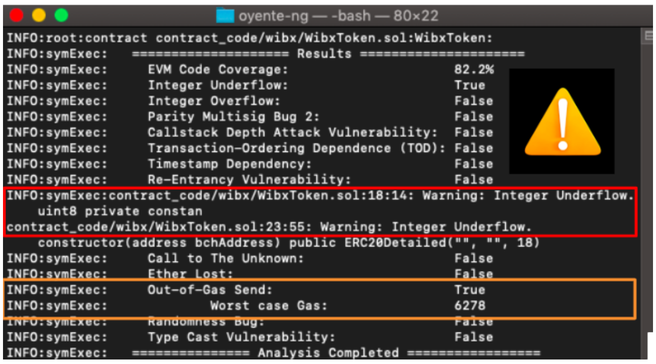

Figure 14. Evaluation of the main contract: WibxToken.sol.

V. CONCLUSION

The goal of this paper was to report the main results of a research effort involving automated reasoning techniques for the detection of security vulnerabilities in Ethereum-based Smart Contracts.

The implemented product, Oyente-NG, has allowed the detection of 5 additional vulnerabilities in complement to the 7 existing ones previously implemented in the original Oyente. Thus, we have shown that: it is possible to automatically analyze, detect, and flag vulnerabilities for Ethereum-based smart contracts; and also it is possible to state that this approach is able to be effectively applied to mitigate risks and/or increase smart contracts resilience.

The following challenges and requirements were successfully tackled on this research: vulnerability taxonomy, automatic detection using symbolic execution, agile development, and smart contracts assessment.

A Research-Based Scrum Agile Framework was adapted for managing the cohesive, productive, and collaborative development team of researchers remotely working. Finally, a Proof-of-Concept applied to a real set of smart contracts has shown the effectiveness of the proposed method.

The authors recommend that those implemented elements associated with different enterprise efforts be used to improve and speed up smart contract quality, thereby optimizing existing resources and contributing to better security.

VI. FUTURE WORKS

As a natural continuation of this research and due to its importance on the global context, the authors of this paper suggest the following works for further research, involving the expansion of the proposed concept:

- Its use for detecting the largest set of contracts (as much as possible) for benchmarking purposes;
- The implementation of new vulnerabilities, insofar as they are discovered, similarly to the updates of antivirus products; and
- Finally, an applicability study of the proposed concept for other languages used for smart contract development, e.g, typescript.

ACKNOWLEDGMENT

The authors would like to thank: the Brazilian Aeronautics Institute of Technology (Instituto Tecnológico de Aeronáutica - ITA); the Casimiro Montenegro Filho Foundation (Fundação Casimiro Montenegro Filho - FCMF); and the Ecossistema Digital Business enterprise. for their infrastructure and financial support to the development of this research project, allowing its PoC in a real environment.

REFERENCES

[1] Nakamoto,S., “Bitcoin : A Peer-to-Peer Electronic Cash System”, https://bitcoin.org/bitcoin.pdf, 2009.

[2] Clack, C.D., et. al..: "Smart contract templates: foundations, de-sign landscape and research directions", 2016.

[3] Atzei N et. al., "A survey of attacks on Ethereum smart contracts" In *Proceedings of the 6th International Conference on Principles of Security and Trust*, vol. 10204, pages 164-186, 2017.

[4] Buterin, V., “A Next Generation Smart Contract & Decentralized Application Platform”, https://cryptorating.eu/whitepapers/Ethereum/Ethereum_white_paper.pdf, 2013.

[5] CoinMarketCap.com. “Ethereum Capitalisation”. https://coinmarketcap.com/currencies/ethereum/, 2018.

[6] Prisco, G., "The DAO raises more than $117 million in world’s largest crowdfunding to date". https://bitcoinmagazine.com/articles/the-dao-raises-more-than-million-in-world-s-largest-crowdfunding-to-date-1463422191, 2016.

[7] Ecossistema Ltd., “Wibx Whitepaper”, https://static.wibx.io/whitepaper_en.pdf, 2018.

[8] Ecossistema Team, “Wibx Coin”, https://github.com/wibxcoin/Contracts, 2018.

[9] Sutherland J., “SCRUM Handbook”, Scrum Training Institute Press, 2017.

[10] Cohen D., et. al., “An Introduction to Agile Methods” In *Advances in Computers*, vol. 62, pages 1-66, 2004.

[11] Luu, L. et. al., “Making Smart Contracts Smarter” In *CCS ‘16, Vienna, Austria*, 2016.

[12] IBM, "IBM Blockchain based on Hyperledger Fabric from the Linux Foundation", https://www.ibm.com/blockchain/hyperledger.html, 2019.

[13] G. Wood, “Ethereum: A Secure Decentralised Generalised Transaction Ledger EIP-150 Revision”, https://gavwood.com/paper.pdf, 2014.

[14] Tosh D. K. et al, “Security Implications of Blockchain Cloud with Analysis of Block Withholding Attack” In *17th IEEE/ACM International Symposium on Cluster,* Cloud and Grid Computing, https://www.computer.org/csdl/proceedings/ccgrid/2017/6611/00/07973732.pdf, 2017.

[15] X. Li, et al., “A survey on the security of blockchain systems” *In Future Generation Computer Systems*, https://arxiv.org/pdf/1802.06993.pdf, 2017.

[16] Dika, A., “Ethereum Smart Contracts: Security Vulnerabilities and Security Tools”, https://brage.bibsys.no/xmlui/bitstream/handle/11250/2479191/18400_FULLTEXT.pdf?sequence=1, 2017.

[17] Szabo, N., "Formalizing and securing relationships on public networks", https://firstmonday.org/ojs/index.php/fm/article/view/548/469-publisher=First, 1997.

[18] King, J., “Symbolic execution and program testing” In *Commun. ACM, 19(7):385-394*, 1976.

[19] Hearn, M.., “Corda: A distributed ledger”, http://www.corda.net/content/corda-technical-whitepaper.pdf, 2016.

[20] Cousot, P., Cousot, R., “Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints”. In *Proceedings of the 4th ACM SIGACT-SIGPLAN*, 1977.

[21] Introduction to Software Analysis. “Chapter 11: Dynamic Symbolic Execution.” YouTube video, 15:52. Jan. 05, 2019. https://youtu.be/QrtGOrSrVPQ

[22] Microsoft Corporation. “The Z3 theorem prover”. https://github:com/Z3Prover/z3.

[23] Nadolinski, E. , “Proxy Patterns”, https://blog.zeppelinos.org/proxy-patterns/, 2018.