Towards Usable Protection Against Honeypots

Christof Ferreira Torres[1], Mathis Baden[2], Radu State[1]
[1]*SnT, University of Luxembourg, Luxembourg*
{christof.torres,radu.state}@uni.lu
[2]*Telindus, Luxembourg*
mathis.baden@telindus.lu

***Abstract*—The Ethereum blockchain enables the execution of so-called smart contracts. These are programs that facilitate the automated transfer of funds according to a given business logic without the participants requiring to trust one another. However, recently attackers started using smart contracts to lure users into traps by deploying contracts that pretend to give away funds but in fact contain hidden traps. This new type of scam is commonly referred to as *honeypots*. In this paper, we propose a system that aims to protect users from falling into these traps. The system consists of a plugin for MetaMask and a back-end service that continuously scans the Ethereum blockchain for honeypots. Whenever a user is about to perform a transaction through MetaMask, our plugin sends a request to the back-end and warns the user if the target contract is a honeypot.**

***Index Terms*—Ethereum, Smart Contracts, Honeypots, Scams**

I. Introduction

Smart contracts are programs that are executed across a decentralized network of distrusting nodes. Ethereum [4] is currently the most prominent blockchain implementation enabling the execution of Turing-complete smart contracts. Smart contracts are usually written using a dedicated high-level smart contract programming language (e.g. Solidity) and afterwards compiled into bytecode. The bytecode is then stored on the blockchain and dictates the execution of transactions that are submitted to the contract.

The major advantage of smart contracts compared to traditional programs, is that they allow to carry out complex transactions among anonymous distrusting parties without the need for a central authority. However, recently attackers started using smart contracts to lure Ethereum users into traps by deploying contracts that pretend to give away funds but in fact contain hidden traps. The community began referring to this type of contracts as *honeypots*.

Honeypots are smart contracts that appear to have a flaw in their design, which allows anyone to drain the funds contained in the contract. However, the ability to retrieve the funds is always bound to a condition that requires a user to transfer a minimum amount of funds in advance to the contract. Then, when a user tries to retrieve the funds, a yet unknown trapdoor unfolds, which prevents the draining of funds to succeed.

In our previous work [3], we proposed HoneyBadger, a tool that leverages symbolic execution in order to detect smart contract honeypots. A large-scale analysis on more than 2 million smart contracts revealed that more than 600 honeypots are currently deployed on the Ethereum blockchain. Moreover, a thorough analysis on a subset of the identified honeypots, showed that 240 users already became victims of honeypots and that attackers already made more than $90,000 profit.

In this work, we introduce a system that makes use of our previous work in order to automatically scan the Ethereum blockchain for newly deployed honeypots and use this information to warn users when they are about to send their funds to a honeypot. The remainder of this paper is organized as follows. Section II discusses preliminary background on honeypots. The design and architecture of our system is explained in Section III. The implementations details are described in Section IV. Finally, we conclude our work in Section V.

II. Background

Honeypots generally operate in three phases. In the first phase, the honeypot creator deploys a seemingly vulnerable contract and places a bait into the contract in the form of funds. In the second phase, a victim finds the vulnerable contract and attempts to claim the bait by transferring the required amount of funds. However, the subsequent attempt of claiming the bait fails and ultimately results in the victim's previously transferred funds being trapped. In the third phase, the honeypot creator withdraws his bait and the funds that the victim lost, through a backdoor in the contract.

In [3], we proposed a taxonomy of honeypots based on the level on which they operate: execution environment (e.g. Ethereum Virtual Machine), compiler (e.g. Solidity compiler[1]) and third-party content provider (e.g. Etherscan[2]). The first class of honeypots deceives users by making use of unknown or unusual behaviour of the smart contract execution environment. The second class relates to honeypots that deceives users by making use of bugs that are introduced by faulty compilers. The final and third class takes advantage of the improper display of information by third-party content providers such as blockchain explorers.

III. System Architecture

Figure 1 illustrates our system architecture, which consists of two main parts: a *front-end* and a *back-end*. The front-end embodies the part through which the user interacts with our system. Whenever a user wants to interact with a smart contract, he or she does this thorough the graphical user interface of a wallet. The front-end consists of a modified

[1]https://github.com/ethereum/solidity
[2]https://etherscan.io/

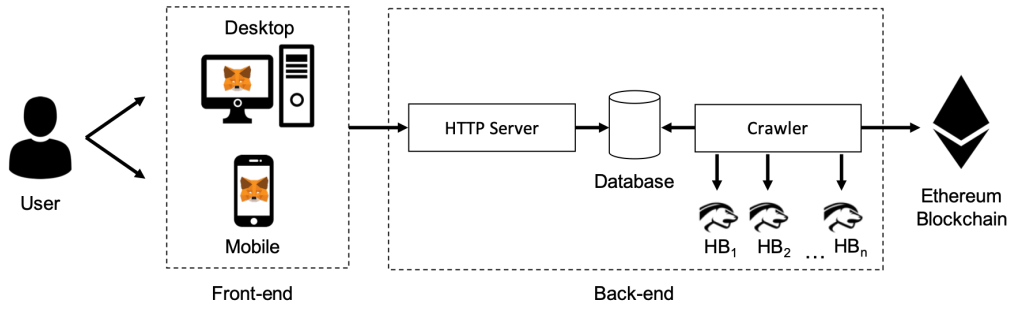


Fig. 1: Illustration of our system architecture.

wallet that sends a request to the back-end to check if the contract is a honeypot or not. Depending on the back-end's response, the wallet either displays a warning message or an approval message, prior to a user sending a transaction.

The back-end consists of two main components: a crawler and an HTTP server. The crawler continuously monitors the Ethereum blockchain for newly deployed smart contracts. For every newly deployed contract, the crawler retrieves the contract's bytecode and starts an instance of HONEYBADGER, which analyzes the bytecode and reports back if the contract is a honeypot or not. The crawler then stores the results of the analysis into a database. Finally, the HTTP server handles the requests coming from the front-end and responds with information about the contract, which is retrieved from the crawler's database.

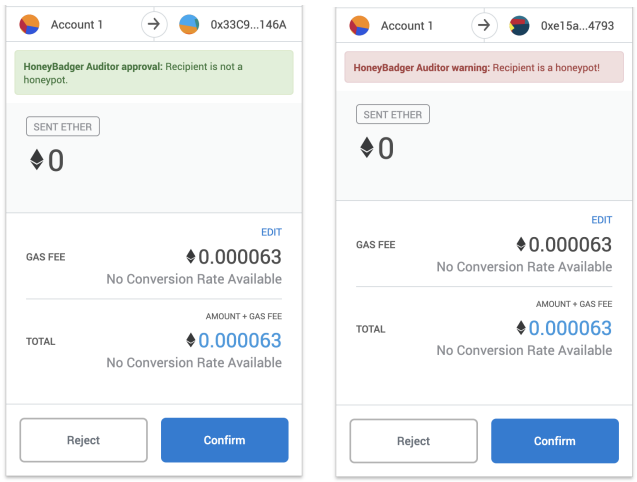
IV. IMPLEMENTATION

We use MetaMask for our front-end, since it is open source and a popular cryptocurrency wallet that runs on different browsers and mobile devices. Moreover, MetaMask recently launched a Beta version that permits to write plugins that extend the capabilities of MetaMask[3]. We created a fork of the original Beta version in order to improve the appearance of messages, by making them more appealing to the user (see Figure 2). We implemented our plugin in JavaScript and named it HONEYPOT AUDITOR. The plugin listens to events that are triggered when a user wants to send a transaction, and then sends a request to our HTTP server. Both, the HTTP server and the crawler have been implemented in Python. We used Flask, a popular out-of-the-box HTTP server implementation in Python and implemented a RESTful API that receives a contract address as input and returns details about the honeypot analysis as JSON. The crawler uses the web3 library to connect to a local Ethereum node and starts multiple instances of HONEYBADGER in parallel through processes. Finally, we used MongoDB as our database.

V. CONCLUSION

In this work, we present a novel solution to protect users from smart contract honeypots. We propose a plugin called HONEYPOT AUDITOR, which users can install in order to extend their MetaMask wallet with the capability of detecting honeypots. Whenever a user is about to send funds to a smart contract, the plugin makes a request to our back-end to check if the contract address has been flagged as a honeypot. Afterwards, the plugin displays a message that warns the user if he or she is about to send funds to a honeypot. Future work includes the extension of our solution towards the detection of new types of honeypots [1], [2], by making use of machine learning techniques.

[3]https://github.com/MetaMask/metamask-snaps-beta



(a) Approval (b) Warning

Fig. 2: HONEYPOT AUDITOR messages in MetaMask.

ACKNOWLEDGMENTS

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

REFERENCES

[1] R. Camino, C. F. Torres, and R. State, "A Data Science Approach for Honeypot Detection in Ethereum," *arXiv preprint:1910.01449*, 2019.
[2] B. Mueller and D. Luca, "The Ether Wars: Exploits, counter-exploits and honeypots on Ethereum," August 2019.
[3] C. F. Torres, M. Steichen, and R. State, "The Art of The Scam: Demystifying Honeypots in Ethereum Smart Contracts," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 1591–1607.
[4] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, pp. 1–32, 2014.