Introduction

The EVM ecosystem faces two big problems which are caused by there being too many contracts and not enough auditors. Firstly, too many contracts are attempting to get audited, meaning they wait longer or apply for auditing from a worse firm. Secondly, there are too many contracts in the networks that exist out there for an auditor to account for the myriad of ways that it might be interacted with. The ecosystem needs a better tool that ensures a higher level of quality. In this paper the current issues facing the ecosystem and a proposed solution to help tackle this issue will be discussed.

Literature Review

Chess AI

“Deep Blue was intelligent the way your programmable alarm clock is intelligent. Not that losing to a $10 million alarm clock made me feel any better.” — Garry Kasparov (1). The paper Deep Blue, written by Murray Campbell, A. Joseph Hoane Jr., and Feng-hsiung Hsu, overviews a chess-playing AI developed by IBM, which famously defeated the world champion Garry Kasparov in a six-game match in 1997. The system used custom hardware, allowing it to evaluate up to 2.5 million positions per second. The system relied on a heuristic evaluation function, which assigned values to different board positions based on factors such as the material balance, pawn structure, and king safety. Deep Blue's algorithms also incorporated a database of grandmaster-level games, which allowed the system to recognize common patterns and strategic motifs. By combining these techniques with a brute-force search of the game tree, Deep Blue was able to analyse many moves ahead and make highly sophisticated decisions. While Deep Blue's victory over Kasparov was a major milestone in the history of AI, the system was highly specialized and could not be easily adapted to other domains.

Smart contracts

A smart contract (Dr Gavin Wood Ethereum 2022) is a self-executing program that automatically enforces the terms of an agreement between two or more parties. It is built on blockchain technology and allows for secure and transparent transactions without the need for intermediaries. Smart contracts are used in a wide range of applications, from financial services and real estate to supply chain management and voting systems. They can execute automatically and autonomously once the predefined conditions are met, making them a powerful tool for automating processes and reducing costs. Smart contracts have the potential to revolutionize many industries by providing a secure, efficient, and transparent way to conduct business transactions.

EVM

This links into the behaviour of the Ethereum Virtual Machine (EVM)(ethereum.org), the EVM is not like most virtual machines where if you had access to it, you could change anything that you want. The EVM is a system of state transition, given a state S and a transaction TX we can represent the state change as such:

$$apply(S, TX) \rightarrow S' or error$$

The reason why this distinction is important is because no failing transaction can be forced into the blockchain Crosby et al., (2015). The Ethereum Virtual Machine (EVM) is a software environment that allows developers to write and deploy smart contracts on the Ethereum blockchain. It is a powerful, decentralized computing platform that enables the execution of code in a secure and deterministic manner. The EVM uses a bytecode language that is designed to be executed on a the virtual machine, allowing developers to write programs that are independent of any specific operating system or hardware. It also has a sophisticated gas system that helps to prevent malicious attacks by ensuring that executing code consumes a finite and limited amount of resources. The EVM is a critical component of the Ethereum ecosystem, enabling developers to build decentralized applications and services that can interact with each other in a trust less and transparent way.

Transactions

Transactions on the Ethereum blockchain are the means by which value is transferred between addresses. Each transaction contains information such as the sender and recipient addresses, the amount of Ether being transferred, and an optional data field. Transactions are broadcast to the network, verified by nodes through a consensus mechanism, and added to the blockchain as part of a new block once validated. To execute a transaction, the sender must have a sufficient balance of Ether and must pay a fee in the form of "gas" to cover the cost of computation and storage on the network. Transactions on Ethereum are irreversible once they have been confirmed and added to the blockchain, making it a secure and reliable way to transfer value. In addition to simple Ether transfers, transactions can also be used to execute smart contracts and trigger automated actions on the blockchain.

Exploits

What is an exploit?

The definition of an exploit is a difficult subject, there are two schools of thought, one is about intention and the second is “if the transaction doesn’t revert, I’ve done nothing wrong”. The first one states that an exploit can occur when a system is not used in the way that was intended. The second looks at the whole network that is being run, the contracts on it and the ways that you can interact with it and says that if they can do something they should be allowed to do it. This behaviour can be observed from the original DAO exploiter, Gazi Güçlütürk (2018): “I am disappointed by those who are characterizing the use of this intentional feature as "theft". I am making use of this explicitly coded feature as per the smart contract terms” Anonymous, (2022).

Network Exploit

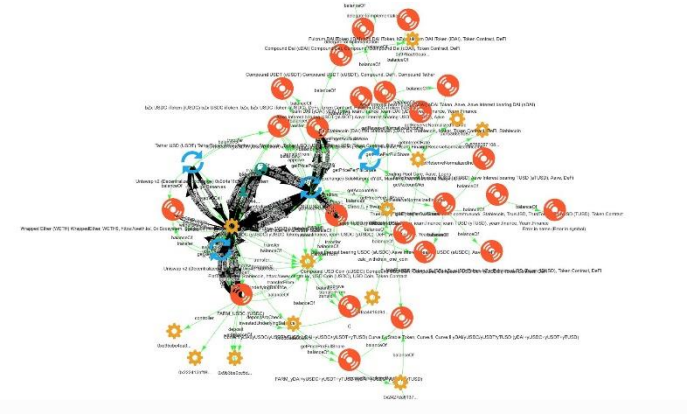
A network exploit in this context is something that sits outside of the EVM that this paper focuses on, for example the Bitcoin inflation bug or the Shanghai DOS attacks. These are things that even though are exploits, they are a deviation of the expected behaviour of the network which is different from a contract that runs on it. This is not an exploit of a contract and thus exploits of contracts that use an exploit in the networks implementation are not counted.

Singe contract

A single contract exploit is where only one contract is involved in the exploit (the exploited one) for example here is a list of common exploits https://github.com/crytic/slither/wiki/Detector-Documentation - slither, most of these are problems that can be exploited in a single transaction, not even needing to write a contract to perform the exploit. These can be things such as not using a NONCE when verifying signatures, or even more simply allowing the override of the “from” field instead of using “msg.sender” for an ERC-20 contract (see later). All these are things that can be easily checked for automatically and thus do not require the planned tool.

Multi contract

Multi contract exploits are more complex inherently as they have more moving parts, for example the Harvest finance exploit in 2020. In the graphic on the left a tool was used to visualise all the transactions and calls made to all of the involved contract. This visualisation shows the complexity of the exploit, which would be very difficult to find manually. When you audit the code before deploying it you are assuming that your team is smarter than the rest of the world and will find everything. There are thousands of people searching exploits every day, they can check through a lot more things that your small team.



Flash loans

Flash loans first introduced in 2019 by Marble allow for the instant zero risk access to Billions of dollars of funds without any collateral. It works because it's enforced by code, a call can be passed and the amount being asked for to the lending contract which sends the funds to the requester, then makes the call that it was given and finally asks for the funds back with a small amount of interest. This all done within a single transaction, the funds are never "left" in the borrower's account. Although research is being done on multi block flash loans which will provide even more opportunities to exploit contracts in the future.

MEV

MEV (Maximal Extractable Value) on Ethereum refers to the amount of value that can be extracted by miners or validators through various activities such as reordering, censorship, and transaction insertion. This value can come from sources such as transaction fees, arbitrage opportunities, and liquidations. MEV can have both positive and negative effects on the Ethereum ecosystem, as it can incentivize miners to act in ways that maximize their profits, but also create instability and inequality. To mitigate these effects, Ethereum developers are working on solutions such as MEV-Geth and Flashbots that aim to make MEV extraction more transparent and equitable.

Current Tools and their methodology

Slither - Static analysis

Slither is a tool that uses static analysis to identify security vulnerabilities in Solidity smart contracts written for the EVM. The tool employs a combination of both data flow and control flow analysis techniques to perform an examination of the contract's code. It first parses the source code into an abstract syntax tree (AST), then applies a variety of analysis techniques to detect potential vulnerabilities such as reentrancy attacks, integer overflow/underflow, and use of uninitialized storage. Slither provides detailed feedback to the developer about the identified vulnerabilities, along with suggestions for remediation. It also includes a number of plugins that can be used to extend its functionality.

The use of static analysis is a valuable approach for identifying security vulnerabilities in smart contracts. Tools like Slither can help developers find and address security issues early in the development process before the contract is deployed on the blockchain. However, there are some limitations to this approach. For example, static analysis tools may not be able to detect all types of vulnerabilities and may produce false positives or false negatives. Additionally, the complexity of smart contracts and the lack of formal specifications can make it difficult to ensure complete

coverage of all potential vulnerabilities. Despite these challenges, the use of static analysis tools like Slither represents an important step towards improving the security of smart contracts and ensuring the integrity of any contract.

Foundry - Unit tests

Foundry is a testing tool designed to aid developers to write comprehensive and efficient unit tests for EVM smart contracts. The tool enables developers to easily create test cases that cover a wide range of scenarios and edge cases. This allows them to identify and address potential vulnerabilities before deploying their contracts to the blockchain. Foundry provides a range of features to support unit testing, including support for a variety of test frameworks, automatic contract deployment, and the ability to simulate real-world conditions such as network latency and gas prices. Unit testing allows developers to test a large variety of features of a contract, this is done by simulating the smart contracts behaviour in a verity of potential environments. It also allows the testing of the contract for multiple network deployments with very little reconfiguration being required.

Foundry – Fuzzing

Foundry provides a powerful fuzz testing feature that can help developers identify vulnerabilities in their smart contracts. Fuzz testing involves generating random inputs and sending them to the smart contract to see how it responds. Foundry's fuzz testing module can automatically generate a large number of inputs and run them through the contract, recording the responses and identifying any potential issues. The tool can also generate inputs that are specifically designed to trigger edge cases or uncommon scenarios, allowing developers to test their contracts under a wide range of conditions. By using fuzz testing, developers can identify potential vulnerabilities that may not be detected by other testing methods, such as unit testing or manual code review. This can help to improve the overall security and reliability of smart contracts, ensuring that they are able to perform as intended in a variety of real-world scenarios.

AI smart contract analysis

Current attempts at the use of ML techniques are mainly focused on the detection and classification of known exploits within a single function. This is inherently limited as none of these systems focus on the wider environment that it will be deployed into. They are only detecting if the function will run as intended not if the design of the overall system is flawed. The two papers focus on the same problem, the classification of vulnerabilities, They use KNNs, SGD and Graph based methods, none of which are able to effectively evaluate the contracts vulnerability to external factors. The focus on the defence of the contract limits the capability of the use of ML methods. They have no goal, nothing to reach, without that they cannot achieve anything.

Non exhaustive list of references / list of useful links

[1] Deep Thinking: Where Machine Intelligence Ends and Human Creativity Begins., Deep Thinking: Where Machine Intelligence Ends and Human Creativity Begins.
[2] https://arxiv.org/pdf/2302.07347.pdf
[3] https://core.ac.uk/download/pdf/82416379.pdf
[4] https://ethereum.github.io/yellowpaper/paper.pdf