A Trillion-dollar infinite arms race

Thomas Biddlecombe
Liverpool Hope University
Artificial Intelligence
20th May 2022

Abstract

Multi smart contract exploits are becoming more prevalent, and the community does not have the tools it needs to be able to combat this. Searching for these multiple contract exploits through a combination of intelligently searching and brute forcing the hopefully reduced combinations can result in these exploits being found before a bad actor can. Building on Marwala, and Xing, (2018) which is more general, this paper focuses on ensuring the safety of the contract as its focus. This paper investigates using AI to ensure the safety of a smart contract within the eco system it is to be deployed, not doing this would be to design a plane and do all the testing apart from simulate it in a wind tunnel. There is a tool made by AnchainAi but this only focuses on individual contract code and is closer to the hard coded tool by Crytic called Slither than AI. Either tool does not have any sort of perspective and instead just treats the contract like it exists in a bubble.

Keywords: EVM, smart contracts, artificial intelligence, auditing

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.

Definition of the problem and its space

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).

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:

$$apply(S, TX) -> S' \text{ or } error$$

The reason why this distinction is important is because no failing transaction can be forced into the blockchain Crosby et al., (2015). Assuming the chain stays secure as it is vulnerable to the 51% attack (ethereum.org) and that the nodes running the chain don't have any bugs in them including the bitcoin inflation bug Sedgwick, (2019) and the Shanghai attacks, (2017). These are examples of the system that runs the blockchain having an issue not the base blockchain.

Focusing on the deviation from the intended behaviour of a contract there are a huge number of directly and even more indirectly interconnected contracts that exist on just Ethereum's EVM alone, see Figure 1

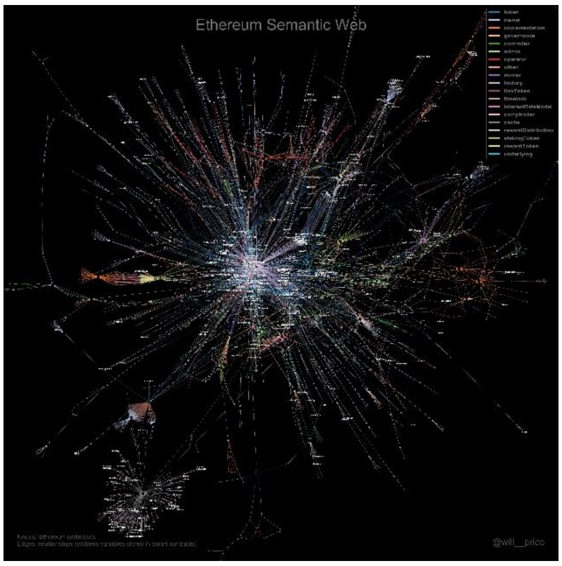
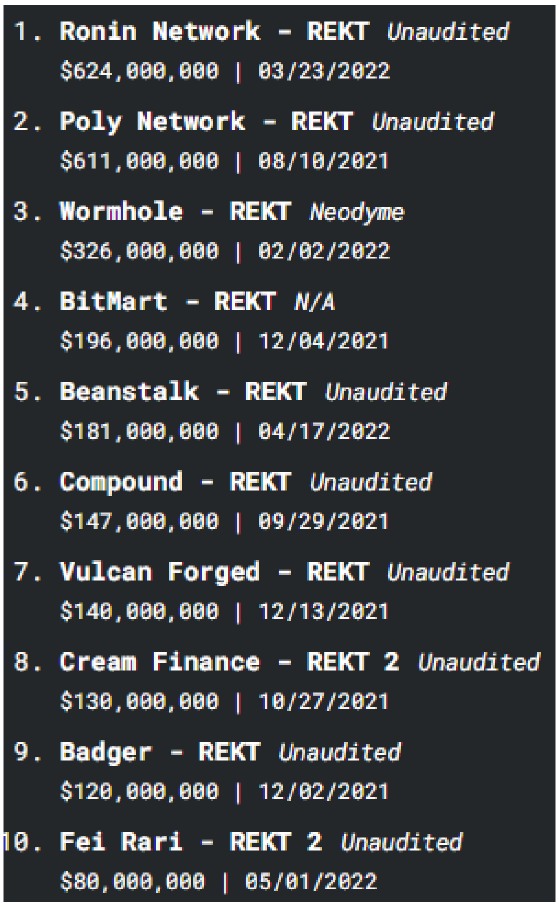


Figure 1: Ethereum sematic web taken from https://twitter.com/will__price/status/1360676121835151360

for a small subset of them and their interconnected nature. To try to account for every possible combination of contract calls and values to be passed would be a monumental manual effort. There are not enough resources to properly audit all the contracts that could be deployed, and the space is too large for the auditor to be able to account for the myriad of attacks that may happen and cannot keep up with the number of new contracts being deployed. The solution needs to be able to scan the network for a wide variety of combinations of contract calls that when chained together would be able to create an exploit.

Exploits account for a huge amount of value lost inside the ecosystem, this leader board by Rekt:



1. **Ronin Network - REKT** *Unaudited*
   $624,000,000 | 03/23/2022
2. **Poly Network - REKT** *Unaudited*
   $611,000,000 | 08/10/2021
3. **Wormhole - REKT** *Neodyme*
   $326,000,000 | 02/02/2022
4. **BitMart - REKT** *N/A*
   $196,000,000 | 12/04/2021
5. **Beanstalk - REKT** *Unaudited*
   $181,000,000 | 04/17/2022
6. **Compound - REKT** *Unaudited*
   $147,000,000 | 09/29/2021
7. **Vulcan Forged - REKT** *Unaudited*
   $140,000,000 | 12/13/2021
8. **Cream Finance - REKT 2** *Unaudited*
   $130,000,000 | 10/27/2021
9. **Badger - REKT** *Unaudited*
   $120,000,000 | 12/02/2021
10. **Fei Rari - REKT 2** *Unaudited*
    $80,000,000 | 05/01/2022

Figure 2: Dashboard for Rekt, taken from https://rekt.news/leaderboard/

It shows multiple famous exploits that have occurred in recent memory, the full list has a recorded $3.8 billion in losses, 4.5% of the ecosystem lost. If these exploits were compared to when they happened and the subsequent amount of total value locked in the network, they would account for a higher proportion. Even though developers have been coding on the EVM for years it took until nearly 5 years for the main programming language to protect against under/overflows by default. Much like the early days of online computing, which ushered in a wave of a single individual being able to get into anything. Compare that to jailbreaking an iPhone which can earn you up to $2 million from Zerodium requiring teams of researchers. Nowadays with tools like slither and the general knowledge of the open-source community, hacks that used to get exploited like re-entrancy are now being closed off resulting in more attacks relying on a vast sum of wealth (economic exploitation), enabled by the advent of flash loans (Docs.aave.com) and better liquidity in decentralised exchanges like Uniswap.

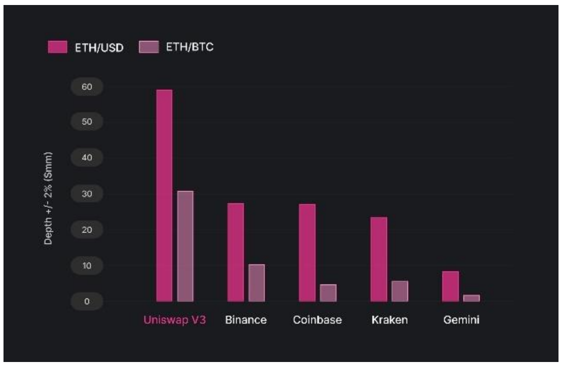


Figure 3: Graph showing the amount that can be made in a single trade without creating more than 2% slippage either way, taken from https://uniswap.org/blog/uniswap-v3-dominance

If there were more tools to ensure a base level of safety of the contracts, then they could be checked over more thoroughly ensuring a higher quality. Meaning that by the time they go to audit, they take less time to ensure correctness as the more rudimentary security problems have already been checked and amended. This means that the overall time it takes to complete an audit reduces, resulting in more and higher quality audits being completed as better teams have more time freed up. This helps accomplish the goal of making the whole network safer.

Design of the system

The design composes of two main parts: the proposer and the fuzzy searcher, the proposer searches for "routes" which are passed through to the fuzzy searcher. The fuzzy searcher takes a list of contracts and the functions to call and then adjusts the parameters that are available to it so that it can try to find an exploit out of it and once found optimise it.

The complexity of the system can be compared to a chess AI, looking for a list of moves that achieves some goal. The proposer can be compared to a search algorithm used in chess, given a state and the previous choices it can choose what contract and function to call to maximise the expected profit. It becomes a task of throwing out the obviously bad moves and refining down to the ones which it expects to result in a successful exploit.

For the refinement of the search space there are a few things that can be used to focus on the search, looking at what other users call, looking at addresses stored in the byte code and certain structures that would allow for known exploits.

For a more complex system that could be launched later, it could utilise Etherscan which has a large repository for these contracts with their code, this means that an NLP model could be used on this data to find other exploits or help filter down the search space for the next contract to call. This NLP system could be used to identify potential risks in the code and then the base AI can look at how to manipulate that single variable, the process is similar to static analysis, working from node to node, building a computational graph.

Analysis of the individual externally owned accounts (EOAs) (not contracts) can be investigated to determine what contracts they interact with then build a dependency graph to show how each bundle relates to each other as a weighted average across all users. Focusing on metrics such as frequency and amount moved relative to the normal sizes being moved by the user. The reason why this may be insightful is that even though the contracts may appear to have nothing to do with one another, they may still be related as part of an ecosystem.

A large benefit of the proposer searcher separation is the possibility for decentralisation, defining two rings of participants in this network, proposers, and searchers. If when a proposer finds a new route that it thinks would be optimal, it compiles the route into just the base byte code, removing anything identifiable. Broadcasting that out to the outer ring, searchers randomly “walk” about this space running the bytecode on the different possible inputs. They then report the data back to the searcher who combines the new state with the network as they look to see if any exploit has occurred. This approach allows for a trust minimised network so if you’re a developer of a new decentralised application (a collection of contracts) you can become a proposer and pay the networks searchers while obfuscating the contents of your contract. The EVM byte code can be de-compiled meaning that it might only be useful for scanning public code and actively scanning against new exploits. For projects pre-launch running the code locally would still be possible as auditing firms could require a certain number of iterations searched as a base requirement.

Computational requirements

Ethereum’s EVM runs at about 0.35 MIPS which is slower than the TK-80, a computer from 1976

Figure 4 TK-80 computer from 1976, taken from http://museum.ipsj.or.jp/en/computer/personal/0002.html

(other EVMs may run at different speeds). The head state is about 400MBs, but this includes data not needed for this problem like other users' balances which aren't necessarily needed. Stripping that back you could assume that the contracts on eth take up about 20-50Mbs at most. Assuming that there are 10,000 contracts of relevancy (any value to be captured) each contract having up to 20 ways to interact with it, the search space is now:

$$200{,}000^{number\ of\ steps}$$

Where number of steps defines how many contracts to string together. Meaning that for an exploit that takes 5 steps to discover, by brute force alone (excluding any parameter optimisation) it would take $3.2e26$ steps. If each step took 1 millisecond to process it would take over ten quadrillion years. This highlights the importance of intelligently filtering options down the search space a lot, by making smarter decisions, throwing away the bad ones and focusing in on the important ones.

Future Work

The application for this is larger than just securing contracts, it could be used to discover new types of maximal extracted value (MEV), this includes on chain activities such as liquidating and market making between different DEXs e.g. front running or sandwich attacking transactions. See this dashboard by FlashBots which shows how much MEV gets extracted.

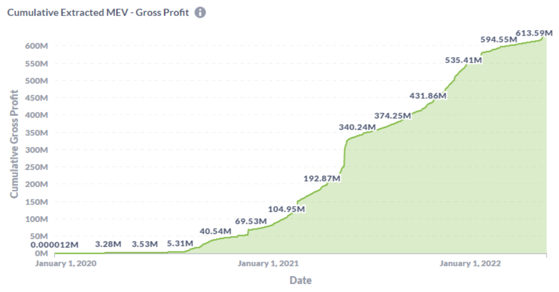


Figure 5: dashboard showing how much MEV is extracted, taken from https://explore.flashbots.net/

This is a large and growing industry, and new ways to capture MEV are being discovered all the time. With the use of this searcher and proposer system, new MEV opportunities can be found which when refined by a user can help stabilise the whole ecosystem.

This system could also be used with a GAN to write and automatically recommend fixes for problematic contracts, whilst still satisfying the test laid out by the developers. Accomplishing two things: train a tool that helps write safe code and recommend changes. Furthermore, this will help to expose issues in the testing scripts of a developer's contract, to ensure the one thing better than 100% code coverage, which is 100% logic coverage.

Conclusion

Security is an infinite battle, against an unrelenting enemy, in an infinite game. "There are no losers, only smaller winners" Rekt News, (2021). A tool that when used by the masses can discover exploits before a bad actor can is vital. Computers are not smart, but they can do roughly the same thing millions of times. By finding new exploits that are refined by experts and fed back into the system to be shared with the wider community, we can hard code these problems into tools like slither so that they no longer need to be checked for by the AI if it discovers it. By building on the work of the open-source community we can move faster by pulling in the same direction.

Citations

Marwala, T. and Xing, B., 2018. Blockchain and Artificial Intelligence. *Arxiv*, [online] Available at: https://arxiv.org/abs/1802.0445 [Accessed 17 May 2022].

AnChain.AI. 2022. Professional Services | AnChain.AI. [online] Available at: https://anchain.ai/services/ [Accessed 18 May 2022].

GitHub. 2022. *slither: Static Analyzer for Solidity.* [online] Available at: https://github.com/crytic/slither [Accessed 20 May 2022].

Docs.aave.com. 2022. *Flash Loans - FAQ*. [online] Available at: https://docs.aave.com/faq/flash-loans [Accessed 20 May 2022].

Gazi Güçlütürk, O., 2018. *The DAO Hack Explained: Unfortunate Take-off of Smart Contracts*. [online] Medium. Available at: https://ogucluturk.medium.com/the-dao-hack-explained-unfortunate-take-off-of-smart-contracts-2bd8c8db3562 [Accessed 18 May 2022].

Anonymous, 2016. *An Open Letter - Pastebin.com*. [online] Pastebin. Available at: https://pastebin.com/CcGUBgDG [Accessed 18 May 2022].

ethereum.org. 2022. *Ethereum Glossary | ethereum.org*. [online] Available at: https://ethereum.org/en/glossary/#51-attack [Accessed 18 May 2022].

ethereum.org. 2022. *Ethereum Virtual Machine (EVM) | ethereum.org*. [online] Available at: https://ethereum.org/en/developers/docs/evm/ [Accessed 20 May 2022].

Sedgwick, K., 2019. *Bitcoin History Part 10: The 184 Billion BTC Bug – Featured Bitcoin News*. [online] Bitcoin News. Available at: https://news.bitcoin.com/bitcoin-history-part-10-the-184-billion-btc-bug/ [Accessed 18 May 2022].

Weare, K., 2017. *Ethereum Security Alert Issued, Ethereum Foundation Responds with "From Shanghai, With Love"*. [online] InfoQ. Available at: https://www.infoq.com/news/2016/09/Ethereum-DOS-Attack/ [Accessed 17 May 2022].

Zerodium.com. 2022. *ZERODIUM - How to Sell Your Zero-Day (0day) Exploit to ZERODIUM*. [online] Available at: https://zerodium.com/program.html [Accessed 20 May 2022].

Uniswap Protocol. 2022. Home | Uniswap Protocol. [online] Available at: https://uniswap.org/ [Accessed 17 May 2022].

Rekt News, 2021. *Curve Wars*. [image] Available at: https://www.youtube.com/watch?v=K24riBPqjmA [Accessed 20 May 2022].