SynergyAI: automated blockchain intelligence provider

Antoni Kiszka

Abstract

Blockchain technologies have given rise to new types of markets, which are vulnerable to abuse due to a lack of traditional regulatory guarantees. We propose a self-adjusting system which allows for assessing the safety of a potential investment, based on an ensemble of algorithms. This ensemble can analyze tokens within seconds from deployment and continuously update its predictions. To prevent drift from current market conditions, the algorithms are constantly tuned based on a retroactive evaluation of their performance. The result is a human-readable and reliable way to estimate the risk associated with a token.

Introduction

Problem

The emergence of cryptoassets, such as ERC-20 tokens, has created numerous novel challenges to investment safety. Seeking high returns, investors purchase tokens in spite of being unable to fully assess the risk of such an investment, making themselves vulnerable to ICO fraud.

The Solidity programming language, being the most popular smart contract development language is to this day at an early stage of development. This results in emerging ways to covey malicious functions and design patterns into the smart contract. The problem is so apparent, that the community of developers started organizing contests to hide[1] ill-intentioned code in seemingly secure contracts.

Around 4.6 million smart contracts were deployed on the Ethereum mainnet in Q4 2022 alone[2], a fair share of them being ERC-20[3] compliant.

[1] Underhanded Solidity Contest, https://underhanded.soliditylang.org/
[2] Alchemy, "Web3 Development Report (Q4 2022)"
[3] F. Vogelsteller, V. Buterin, "ERC-20: Token Standard" (2015)

Research[4] shows that 97.7% of Uniswap[5] listed tokens are considered *rug pulls*. If one were to choose a random token listed on Uniswap, there would be approximately a 97.7% chance that they would choose a fraud.

Decentralized exchanges (DEXs) are the most popular market where tokens can be obtained. Unlike traditional exchanges, such platforms are often open-source and have extensive public security audits. As a result, we assume counterparty risk to be minimal and instead focus on risks related to the token's holders and the token itself.

We find that the majority of ICO fraud (*rug pulls*) is performed by means of:

- the token creators retaining a significant amount of the initial supply that is obfuscated from potential investors;
- or, purposefully placing illicit mechanisms in a smart contract, e.g. allowing the creators to mint any amount of the token.

Solutions

Many solutions exist that try to tackle this problem. The most trustworthy way of ensuring that an asset is not fraudulent is auditing. However, the existence of audits does not decrease the risk of investing in new or low-marketcap tokens. They present a catch-22 situation: only companies that are not conducting fraud are willing to pay to have an audit conducted. Thus, they are of little value to investors seeking to support early-stage projects.

The goal of ensuring the safety of all token investments requires tools that are quick to use and do not require significant monetary commitment.

Recently, solutions trying to fulfil those criteria have become publically available. Notably, TokenSniffer[6] and honeypot detectors help detect anti-features[7] of smart contracts. While they can be a good starting point for investment research, their assessments are often changed by simply altering the names of functions and events, as those tools often look only for specific techniques employed by fraudsters instead of performing a full analysis.

In accordance with the above, our literature review has found that the amount of ICO fraud has not decreased since the introduction of those solutions[8]. We postulate that this is a direct result of the inadequacy of currently available analysis tools.

We believe that reliable fraud detection should not be based on searching for well-known techniques. Instead, full-spectrum code analysis and machine learning

[4] X. Pengcheng et al., "Trade of Trick? Detecting and Characterizing Scam Tokens on Uniswap Decentralized Exchange" (2021), URL: https://export.arxiv.org/pdf/2109.00229

[5] Uniswap Decentralized Exchange, https://uniswap.org

[6] https://tokensniffer.com

[7] An *anti-feature* is a functionality intentionally introduced by a developer whose existence can harm the user of the smart contract.

[8] S. Shobhit, "$9 Million Lost Each Day In Cryptocurrency Scams" (2019)

provide a way to detect both known and novel attempts at subterfuge.

The results of such a full-spectrum analysis can be used by investors and on-chain software through specialized oracles.

Methodology

SynergyAI analyzes a specific token from three specific *in situ*[9] perspectives:

1. an AI-based static analysis
2. dynamic analysis
3. token holder graph analysis

Given a token T, those three perspectives can be modelled as functions of the form:

$$A\colon \mathbf{T} \longrightarrow [0, 1]$$

That take as input some token, including its smart contract creation code and yield a risk score associated with this contract. We define the risk associated with a particular token as the percentage of the supply that a single entity can control. For example:

- a token whose contract includes a mint function callable by the owner has an associated risk of $r = 100\%$
- a token which has 50% of its supply in a locked liquidity pool, 30% bought by unconnected EOAs[10], and 20% in a premint unvested pool has a risk of $r = 20\%$.

When aggregated, the risks returned by all three analytic algorithms provide a way to accurately assess the trustworthiness of a smart contract[11].

This approach is akin to a Mixture of Experts ensemble used in AI. However, unlike MoE ensembles, SynergyAI does not utilize deep learning algorithms for analysis of the entire problem space. Instead, we chose a mixture of deep learning, dynamic programming, and classical graph analysis as we found those approaches to better fit the diverse problem space of token risk assessment.

AI-based static analysis

Static analysis is the process of analyzing a program without running it. Such an approach allows to detect flaws and intentionally created anti-features.

[9] In this context *in situ* means that the analysis is performed completely off-chain and does not involve any interaction with the deployed token.

[10] An *externally owned account* (EOA) is an account not controlled by a smart contract, but rather by a regular user.

[11] We assume that every token is associated with a single main smart contract.

The publically available contract creation code, published whenever a smart contract is deployed, is notoriously difficult to decompile and understand. Unlike regular decompilation tools, we preprocess and structure parts of the smart contract code; this prepares the code for later analysis by means of an AI model.

Dataset

For usage with the model, a smart contract bytecode must be preprocessed using the following steps:

1. the storage location for token balances is identified and saved;
2. code paths that only read the state of the contract are discarded, leaving only write functions;
3. functions directly affecting the token balances are identified;
4. functions affecting values that the afore-mentioned functions depend on are identified - this dependency resolution is repeated until completion;
5. code paths that do not affect balances directly or indirectly are discarded.

This process results in a set of paths that lead from a set of initial conditions and constraints to a change in token balances. Such a path can be modelled as a directed acyclic graph (DAG) and regarded as a single way to change one's token balance.

In this form, paths can be tokenized and ingested into the AI model. We have obtained satisfactory preliminary results by representing the graph as a text file containing the list of vertexes, but future research is needed to evaluate the best format for the tokenization of data in such a format.

Training

The model is initially trained by creating smart contracts manually and introducing variations in their features (i.e. changing the names of functions, modifying the list of possible features, merging and splitting code paths, altering constants etc.) The variation is random but remains predictable enough to allow the risk factor to be calculated manually. The pairs of compiled bytecodes and calculated risk factors are used to train the model.

The aforementioned process provides an essential part of model training. Nevertheless, it is not feasible to generate training data that is representative enough to cover most real-world applications. To ensure that the model improves continuously and can adapt to changing market conditions, a retroactive assessment and tuning pipeline should be implemented. The reasoning behind such a process has been described in Accuracy improvements.

Interpretation of results

The static analysis model returns a single piece of information: the risk value. This value does not reflect the overall risk related to a token - it captures only

one of its parts, i.e. the possibility of the smart contract containing dangerous anti-features.

Dynamic analysis

Dynamic analysis is the process of inspecting software by executing it in a controlled environment. By executing certain functions within a smart contract, followed by simulating transfers and sales. this algorithm is able to detect anti-features such as hidden fees and honeypots without the need to understand the underlying code.

The analysis process begins by identifying all public and external functions within a contract that modify its state. Then, we identify functions defined by EIP-20[12] and assert their compliance with the standard using a standardized test suite. Upon detection of a standards violation (such as failing to return the correct balance of a holder), the token is automatically issued a risk score of 1 and flagged for manual review.

Compliant tokens progress to the next stage where the algorithm simulates calls to functions outside EIP-20 from different addresses and notes the changes. The purpose of this step is to identify functions that set fees, change reflection settings or block certain addresses. Having identified such functions, the algorithm performs a search in the parameter space, trying to find the configuration, which funnels the most funds to the deployer of the token.

Such a search is easy to perform within the space of a single function, such as `setFee(uint256 nominator, uint256 denominator)`; however, searching larger parameter spaces thoroughly necessarily involves a non-constant speed penalty to the algorithm. While we have had promising results with using classical methods of reducing search spaces, we are also considering the introduction of an AI model able to look for patterns without human intervention.

We include this algorithm in the ensemble to ensure that every contract analysis is complete. Having two complementary approaches to contract analysis helps safeguard against model hallucination and significantly increases the cost of developing ways to subvert the fraud detection system.

Holder graph analysis

A significant of ICO fraud is conducted even without the introduction of contract anti-features. If Eva deploys a new ERC-20 smart contract, she can create the token supply herself. Then, she can only add a part of the said supply to a liquidity pool and scatter the rest throughout a range of seemingly unaffiliated addresses. Such a process can be mediated by exchanges, bridges and other tools, often making the transactions untraceable to the average investor.

[12] F. Vogelsteller, V. Buterin, "ERC-20: Token Standard" (2015)

By analyzing the network of token holders at TGE[13] and monitoring how it changes over time, we are able to identify cliques of cooperating addresses. Then, instead of considering their holdings as separate, we can lump them together and identify how large are the holdings of contract deployers and other groups.

The risk returned by this analysis is the per cent of the total supply held by the largest clique, excluding tokens held in approved lock contracts with a release time of more than 2 days in the future. Such a value is important to potential investors, as cliques operate as singular entities

- they are able to coordinate the sale of their holdings as well as other operations that could significantly alter the price of an asset.

Aggregation and accuracy

To counteract potential variance in accuracy between ensemble algorithms we introduce an aggregation system. Similarly to a gating model in an MoE setting, the aggregation system can be thought of as a one-layer neural network. The weights within the aggregator are defined as follows:

$$\text{Let } a, b, c \in [0, 1] \text{ such that } a + b + c = 1$$

Those weights are associated with modes of token analysis. Consequently, given a token T, we define an aggregate risk function $R(T)$ such that:

$$\forall T : R(T) = a * A(T) + b * D(T) + c * G(T)$$

Where $A(T)$ is the result of the static analysis, $D(T)$ is the result of the dynamic analysis, and $G(T)$ is the result of the holder graph assessment.

$R(T)$ can be regarded as the final risk value calculated by the ensemble.

Accuracy improvements

Initially, the aggregator weights are set at $a = b = c = 1/3$. As more tokens are assessed by the algorithms, SynergyAI will be able to gather more data about the accuracy of the algorithms.

Improving the accuracy of the aggregator requires monitoring past risk scores for all tokens. After a certain time, we expect the deployers of a portion of those assets to sell their holdings. We assume that a rational actor always eventually sells all of their liquid holdings, i.e. after we have identified a token as fraudulent it can be assumed that its creators will not sell less than they are able to - thus

[13]Token generation event, here understood as the moment when the token supply is first created.

maximizing their profits. Operating under that assumption, we can calculate the actual risk:

$$R_a = \text{tokens sold/total supply}$$

Having gathered an appropriate amount of data on past performance, comparing R_a to $A(T)$, $D(T)$, and $G(T)$ makes it possible to identify the loss of the algorithm and change the aggregator weights to promote more accurate algorithms.

Conclusion

We have proposed a comprehensive system for estimating the risk of a token being used to conduct ICO fraud. Unlike preexisting tools, the SynergyAI algorithm ensemble can detect fraud by first-principles analysis, instead of simply comparing against a predefined list of illicit techniques. Additionally, the SynergyAI suite is able to evolve without human intervention, ensuring the quality of results even as blockchains change.

This allows SynergyAI to provide a service that accurately detects threats while keeping the cost of a singular evaluation low.