TestMachine

Whitepaper

Andrew Kilbride - Co-Founder CEO
ajk@testmachine.ai

Matthew Lewis - Co-Founder CTO
mjl@testmachine.ai

testmachine.ai

Abstract

Blockchain projects face several challenges that can lead to failure, including lack of time due to repetitive code reviews, lack of speed due to manual contract testing, and lack of security due to endless tool integration. These challenges can cause significant delays and financial losses, ultimately leading to the failure of the project. However, TestMachine provides a comprehensive solution that addresses these challenges and ensures the success and security of blockchain projects. Our real-time testing and auditing platform powered by AI eliminates manual code reviews with an on-demand audit, which allows developers to code while checking for bugs on the fly, saving time and improving productivity. We leverage industry-standard security tools and proprietary machine learning algorithms to get rid of manual testing, thus removing expensive delays for product launches. Additionally, we provide a zero-configuration ecosystem with an intuitive web interface, GitHub integration, and powerful command-line tools that remove the hassle of installing dependencies that can break the code and make it unsafe. By combining industry-standard tools with a novel reinforcement learning approach to smart contract penetration testing and on-demand audit, we help blockchain projects overcome the challenges and achieve success in a highly competitive industry.

1 Introduction

TestMachine is a real-time testing and auditing platform for blockchain developers. Our goal is to provide companies and developers with the most secure code possible on the blockchain. Writing secure code for blockchain applications is an increasingly urgent problem and becoming increasingly difficult to achieve. On the Ethereum chain alone, more than 46 million contracts have been deployed, comprising more than six billion lines of code. As the number of complex smart contracts and dApps deployed to blockchains grows, conventional approaches to code security — namely the manual review of code by expert human programmers — will become untenable within the next two years. TestMachine will be deployed in 2 main phases:

2 Phase I

leverages industry-standard open-source tools and proprietary machine learning algorithms to test, analyze, and dynamically expose known and unknown security vulnerabilities in smart contracts and dapps. TestMachine provides an intuitive web interface, tight GitHub integration for inclusion in existing CI/CD pipelines, and powerful command line tools that simplify and secure a developer's everyday workflow. Rather than forcing developers to commit to an expensive, time-consuming audit at the end of the development cycle, TestMachine will allow users to continuously audit blockchain code.

3 Phase II

brings the Surfactant Machine Learning Platform to bear on the problem of smart contract security. Surfactant uses novel reinforcement learning techniques to teach artificially intelligent agents to detect and exploit vulnerabilities in a smart contract. Powered by deep neural networks and trained on high-fidelity blockchain simulations, these agents learn powerful strategies that exploit data oracles, blockchain latency, and complex contract interactions. TestMachine analyzes these exploits to ensure developers can mitigate the vulnerabilities in their contracts and dApps. The TestMachine platform provides Phase I and Phase II capabilities to developers as part of a zero-configuration ecosystem. There is no need to search for new security tools in GitHub repositories; there is no need to provision one's own servers; and there is no need to worry about keeping up with best standards and practices. TestMachine will make blockchain code security as frictionless and as compelling as companies like Amazon Web Services have made deploying complex Web 2 applications. Below we summarize the current technical state of TestMachine, focusing on the system architecture and the details of the Surfactant machine learning platform.

4 System Architecture

We have developed the core system architecture for TestMachine and are currently building out the components for both the Phase I (open source) and Phase II (machine learning) components. In its current implementation, TestMachine is built on AWS, mainly using Kubernetes via the Elastic Kubernetes Service (EKS) for maximum efficiency and flexibility in both development and production. Our AI architecture can explained in 5 areas:

4.1 Interface

This is the user interface of the TestMachine application. It handles all user requests by interacting with the database and serving JSON, HTML, CSS, and JavaScript through HTTPS requests. Depending on the frontend framework that we end up using, an S3 bucket will be used to serve static files, while the API will be hosted in the EKS cluster. The API will sit behind a production-grade web server such as NGINX.

4.2 Auth

To maximize security, we will either use existing third-party identity providers such as Google, Twitter, or GitHub through OAuth, or use an internally hosted third-party identity provider such as Keycloak. Either way, TestMachine will leverage an internally hosted solution (possibly Keycloak) to operate as an identity broker and to authorize data access.

4.3 Language Tools

Language tools are the open-source tools that TestMachine will operationalize to streamlining the Web3 development experience. For solidity, these include tools such as OpenZeppelin and Slither, but for Vyper the toolbox looks a bit different. For each language TestMachine will have a scalable microservice that will parse the contract and report analysis results in both human- and machine-readable formats.

4.4 Simulation Network

TestMachine will house a variety of solutions to simulate real-world blockchain networks. These will range in complexity from single Ganache instances to larger, multi-node networks. There are two components of the simulation network: The simulation management layer will operate to collect metadata on the simulations, as well as facilitate the networking between the machine learning agents and the nodes hosting the simulation network. The simulation nodes themselves will capture as many real-world blockchain network features as possible, potentially spread across disparate environments.

4.5 Storage

The database will be hosted by Amazon RDS and guarded carefully within our security model. Any uploaded data or data artifacts too big or inappropriate for inclusion in the database will be stored in Amazon S3 buckets. TestMachine will be built using the Zero Trust security model, so all data will be encrypted over networks, regardless of whether they are internal or external. Additionally, permissions will be assigned according to the principle of least privilege.

5 Reinforcement Algorithms

As discussed above, the blockchain community has a suite of open-source tools for analyzing the quality of smart contracts and detecting known security vulnerabilities. These tools offer support against known vulnerabilities, they provide little help against unknown threats and more sophisticated attacks. To address this problem, TestMachine uses a novel machine learning approach based on a set of techniques known as reinforcement learning. An artificially intelligent agent observes its environment to find itself in a state sS. In the present context, the set of available observations, denoted S, includes the state of the blockchain, the properties of the smart contract in question, the value of relevant data oracles, and so on. The agent takes an action according to a policy, (s,a). These actions are selected from a set of actions, aA , and may include, e.g., calling publicly accessible methods on the smart contract, spoofing a data oracle, pausing for a certain amount of time, and so on. After an action is taken, the agent observes the state of the environment again. The agent subsequently receives a reward, R, based on the action taken and the new state in which it finds itself. The reward is designed to encourage specific behavior in the agent. For example, in the current context, the reward might be the total number of tokens extracted from the smart contract into the wallet of the adversarial RL agent. Training algorithms then update the agent's neural network based on the observed states, actions, and rewards. The update ensures that the agent learns to maximize the reward over time. The goal is to learn which action to take in each state in order to maximize the total reward received over time.

5.1 Training Agents

How do we train the agents — that is, how do we modify the agent's neural networks in order to produce a useful policy? RL is a so-called unsupervised machine learning technique, meaning that it does not require vast amounts of labeled data — that is, contracts that have been hand-labeled by humans as, say, "good smart contracts" or "bad smart contracts". Instead, we require only a reward function that specifies what we care about, and a high-fidelity simulation of the contract operating on the blockchain. We can then simulate the RL agent interacting with the contract and allow the RL training algorithms to intelligently modify the agent's neural network. A deep neural network consists of many layers of so-called neurons connected by a web of weights, as illustrated in Figure 3. These weights specify the connection strength between each neuron in one layer and the neu-

rons in layers above and below. Note the subscript on the policy, : represents the set of weights that characterize each agent's neural network. At each step in the training process, we update the weights tt+1 so that the agent "learns" sophisticated behaviors that will maximize the reward it receives. We must do this very carefully, however. The RL problem is mathematically straightforward, but training agents can be difficult. If the neural network parameters are changed too quickly, the training process may become unstable, and the policy may never converge to something useful. To this end, we use a method of training known as policy gradient algorithms, and have implemented a variant of policy gradient known as Proximal Policy Optimization (PPO) [Schulman, 2017], which is both data-efficient and highly stable. The technical details of the algorithm are beyond the scope of this discussion, but PPO provides well-motivated mathematical safeguards that allow us to update the neural network as rapidly as possible without inducing instability into the training process. This means that TestMachine can learn quickly — that is, with fewer steps of the simulation — while at the same time ensuring that the policy converges to something that reliably identifies security vulnerabilities in smart contracts.

5.2 The Surfactant System

This framework handles the details of automatic differentiation and allows us to rapidly design and implement deep neural networks with the flexible topologies required for the problem at hand. At a high level, Surfactant allows reinforcement learning algorithms to operate on simulated blockchains. Surfactant currently supports Solidity-based smart contracts operating on simulated Ethereum chains, which are implemented as detailed in the Simulation Network section. Future efforts will extend the platform to other languages and chains. The long-term goal of the Surfactant platform is to become largely agnostic to both blockchain and the underlying programming languages. The focus of our current implementation is two-fold. First, we have designed deep neural networks with the notion of transfer learning in mind. By transfer learning, we mean ensuring that the policy that has been learned in one instance — by breaking a specific smart contract, for example — can be leveraged to rapidly exploit new, previously unseen contracts. By allowing our agents to learn across contracts, we can vastly accelerate the process of vulnerability detection. Furthermore, consolidating exploitation strategies into a core neural network (or set of networks) allows us to improve the accuracy of our algorithms and differentiate TestMachine capability from competitors. Ensuring that learning transfers from one task to another requires, among other considerations, careful attention to the structure of the neural networks that implement the RL policy. Different smart contracts have different action and observation spaces, and we must ensure that the structure of our networks allow us to capture core exploitation strategies while adapting as necessary to the different interfaces provided by the other contracts. Second, we have developed novel approaches to representing observation and action spaces in the context of smart contracts. These methods allow us to model the methods and strongly typed arguments associated with Solidity-based smart contracts. We can exploit both discrete and continuous actions simultaneously when training RL agents and handle the data types intrinsic to the Solidity programming language. Proper handling of observation and action spaces can dramatically improve the performance of the RL algorithms on real-world problems. As the Surfactant platform continues to develop over the next several months, we will explore additional training algorithms and network architectures. We will perform systematic experiments to optimize the size and structure of the neural networks, statistically characterize the time required to identify vulnerabilities in contracts, and determine the minimally required fidelity of the underlying blockchain simulations.

6 Disclaimer

This paper is for general information purposes only. It does not constitute investment advice or a recommendation or solicitation to buy or sell any investment and should not be used in the evaluation of the merits of making any investment decision. The opinions reflected herein are subject to change without being updated.