Introduction to TestMachine

Smart Contract Security Driven by Reinforcement Learning

24 July 2024

1. / Predator Overview

Predator is TestMachine's flagship technology. Predator is a standalone AI platform that leverages reinforcement learning (RL) algorithms and an Ethereum Virtual Machine (EVM) simulation environment to create a powerful smart contract penetration testing system. This system goes beyond the capabilities of passive code audits performed by humans, large language models (LLMs), or static code analyses by intelligently exploiting smart contract vulnerabilities in a simulated EVM environment.

At the core of TestMachine's vulnerability detection capabilities is an advanced deep reinforcement learning (RL) system, called Predator. RL allows an AI agent to learn and adapt its strategies through continuous interaction with the target contracts in a simulated blockchain environment. By exploring the vast space of possible interactions and receiving feedback in the form of rewards or penalties, the RL agent learns to identify and exploit vulnerabilities that may be missed by traditional static analysis or manual testing methods. A general outline of Predator's RL problem is illustrated in Figure 1.

Predator leverages RL algorithms to create a dynamic testing environment that can uncover subtle and complex vulnerabilities in smart contracts. The AI agent learns to recognize patterns and behaviors that are indicative of potential security risks, such as reentrancy, integer overflow, or access control issues — an also logic errors that are specific to a given contract or protocol. Through repeated interactions with the contract, the agent learns to craft increasingly targeted and effective exploit strategies, adapting its approach based on the specific characteristics and constraints of the contract. This iterative learning process allows Predator to identify vulnerabilities that may be difficult or impossible to detect through other means, providing clients with a highly effective and efficient tool for assessing the security of their smart contracts.

Predator uses the Jungle[1] framework to simulate realistic and diverse scenarios that may not be feasible or practical to test through manual means. Predator's AI agent can generate and execute a wide range of test cases, covering various edge cases, market conditions, and user behaviors that could potentially trigger vulnerabilities in the contract. This approach helps to ensure that the contract is robust and secure against a broad spectrum of potential threats, reducing the risk of costly and damaging exploits in real-world deployments.

Another benefit of Predator's RL-based approach is its ability to adapt and evolve alongside the rapidly changing landscape of smart contract security. As new vulnerabilities and attack vectors emerge, Predator's AI agent automatically learn to recognize and exploit these novel threats. This adaptability is necessary in the constantly evolving world of web3 technology, where new security challenges arise daily. By leveraging the power of RL, Predator can provide clients with the most up-to-date and effective smart contract penetration testing capabilities available.

[1] **Jungle** is a critical element of TestMachine's Predator platform, serving as the underlying framework that enables the Predator AI agent to interact with and analyze smart contracts in a simulated blockchain environment. Jungle abstracts away the complexities of the Ethereum Virtual Machine (EVM) and handles the integration of various data sources and analytical tools, allowing Predator to focus on its core task of identifying and exploiting vulnerabilities through advanced reinforcement learning techniques. See more in Section 2. /

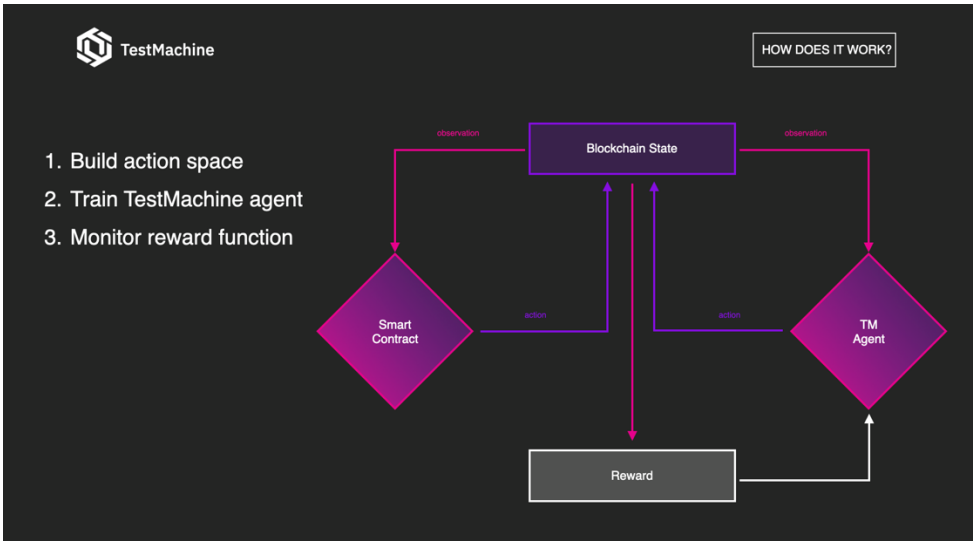

Figure 1: Outline of the `Predator` Reinforcement Learning (RL) problem. A `TestMachine` agent observes the state of the smart contract (and the blockchain in general), and then takes an action — e.g., calling a function on the smart contract. As a result, it receives a reward. The reward is tailored to incentivize the agent to do bad things to the contract — steal tokens, reveal private information, etc. — and thereby reveal critical vulnerabilities.

One of TestMachine's key innovations is how it translates developer-identified invariants into reward functions that guide the learning process. Invariants[2] are essentially properties or conditions that should always hold true for a smart contract to function correctly and securely. For example, an invariant might specify that the total supply of a token should never exceed a certain limit, or that a user's balance should never become negative. By encoding these invariants into a reward function of the `Predator` agent, we incentivize it to actively seek out and exploit vulnerabilities that violate these conditions. In the following sections, we present examples of how Predator uses invariants and their corresponding reward functions to identify and exploit subtle vulnerabilities in DeFi protocols and smart contracts.

1.1. Example: Draining Tokens from a DeFi Protocol

Of course, in practice, the invariants and reward functions used by `Predator` are much more complex and nuanced, considering a wide range of factors such as the specific logic of the contract, the interactions between different components, and the potential impact of external factors like market conditions or user behavior. However, the basic principle remains the same: by translating developer-identified invariants into well-crafted reward functions, we can shape the learning process of the `Predator` agent and enable it to discover and exploit vulnerabilities that might be missed by traditional security audits or static analysis tools.

During a `Predator` campaign, we track the progress of the agents by looking at their reward functions. An example of such a function is presented in Figure 2, which charts reward over time during a campaign against a popular DeFi lending protocol.

[2] In the context of smart contracts, an **invariant** refers to a specific condition or property that must hold true throughout the execution of the contract to ensure its intended behavior and security. Invariants are used to define the expected state of the contract at any given point and can be used to verify its correctness and identify potential vulnerabilities.

The reward function starts out negative — Predator agents are penalized, *e.g.*, for improperly calling smart contract functions. Over time, however, the agent learns how to effectively call functions and manipulate contract state. After a few hours, in this case, the Predator agent successfully steals tokens from the protocol, thereby breaking a critical invariant, and causing a spike in the reward function at the 1.5 million iteration mark.

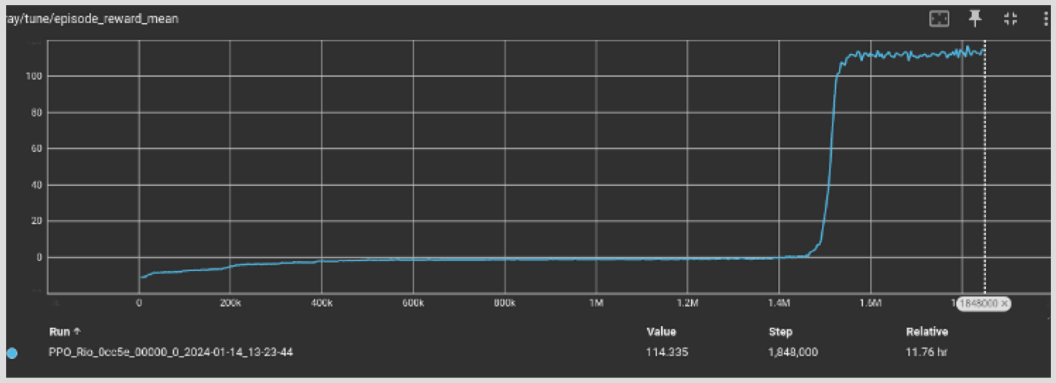

Figure 2: Sample Predator reward function during a successful DeFi protocol hack. Though the reward function starts out negative, after a few hours of learning, it jumps up dramatically — an event that here represents a successful token drain against a DeFi lending platform!

When this occurs, the TestMachine platform immediately informs the client of the hack and delivers actionable feedback — about what happened and how to fix it. TestMachine provides this feedback in several forms, as illustrated in Figure 3. First, it can supply the Transaction Attack Sequence (TAS), which is a JSON file listing the specific transactions that lead to the exploit. Second, using LLMs it can provide a human readable explanation for how the attack occurred, including detailed information on how to fix the codebase. And finally, it can use the same LLMs to generate an executable unit test (here written in Brownie, which is a convenient Python interface to the EVM), which allows developers to easily recreate the attack locally on their own systems.

This detailed, automatically generated, immediately actionable feedback differentiates the TestMachine platform from any other security product available on the market today.

Predator also provides users real-time insight into its campaign via its interactive dashboard, as illustrated in Figure 4. The Predator dashboard is an essential tool for clients to gain real-time insights into the progress and

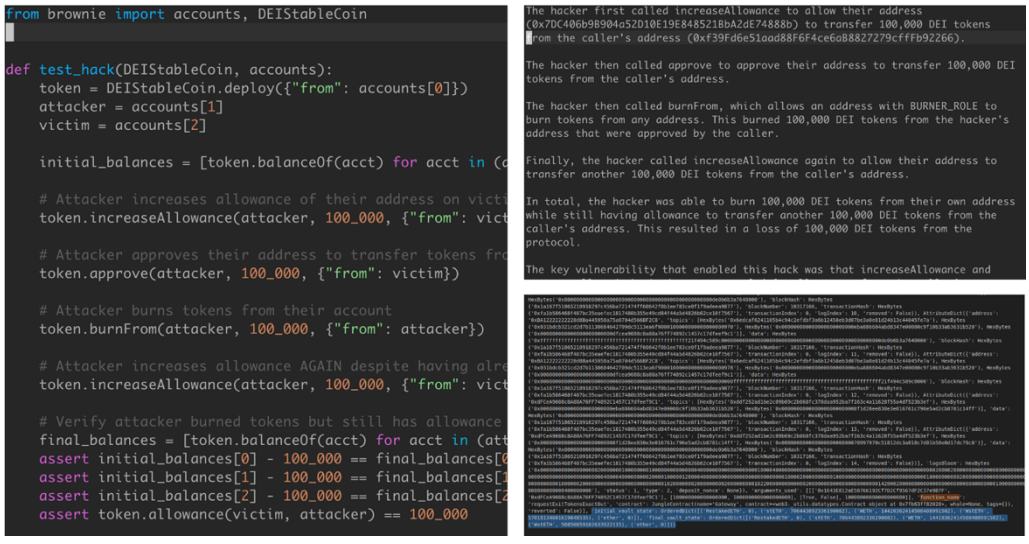

Figure 3: On the left, a unit test automatically generated by TestMachine that implements the exploit that Predator discovered. On the top right, a human-readable explanation of what happened during a successful hack. And, on the bottom-right, the Transaction Attack Sequence (TAS), a detailed list (in JSON format) of transactions that lead to the invariant-breaking hack.

effectiveness of the AI agent's smart contract penetration testing campaign. By providing a comprehensive and intuitive visual representation of the agent's activities and findings, the dashboard empowers clients to monitor the testing process closely, identify potential vulnerabilities quickly, and make informed decisions about the security of their smart contracts.

The Predator dashboard presents a stacked bar chart that displays the relative frequency of the RL agent's function calls. This chart offers a clear and concise overview of the agent's interactions with the target contract, allowing clients to understand the distribution of successful (red) and reverted (blue) function calls briefly, as illustrated on the left of Figure 4. By analyzing this data, clients can quickly identify patterns or anomalies in the agent's behavior, such as a high proportion of reverted calls, which may indicate the presence of access control issues or other vulnerabilities in the contract. This information can help clients prioritize their remediation efforts and focus on the most critical security risks first.

Another important aspect of the Predator dashboard is the tracking of the agent's exploration of novel contract states through the enumeration of unique storage key sets mutated (see the right side of Figure 4). This feature provides valuable insights into the breadth and depth of the agent's testing coverage, ensuring that the AI is thoroughly exploring the contract's state space and uncovering potential vulnerabilities across a wide range of scenarios. By monitoring the growth of unique storage key sets over time, clients can assess the effectiveness of the testing campaign and determine whether the agent is successfully identifying new and relevant states that could impact the contract's security. This information can also help clients gauge the completeness of the testing process and decide when sufficient coverage has been achieved.

The real-time nature of the Predator dashboard allows clients to actively monitor the progress of the testing campaign and respond quickly to any identified vulnerabilities. As the agent explores the contract and uncovers potential issues, the dashboard updates dynamically, providing clients with immediate feedback on the status and findings of the testing process. This real-time feedback loop enables clients to collaborate closely with the TestMachine team, discussing the implications of the discovered vulnerabilities and developing targeted remediation strategies to address them promptly. By fostering a transparent, responsive, and automated penetration testing environment, the Predator dashboard helps clients stay agile and proactive in their smart contract security efforts.

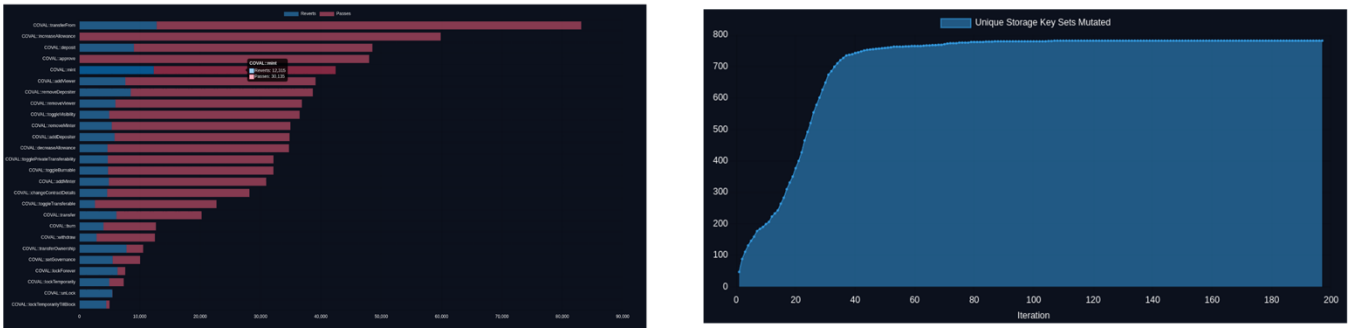

Figure 4: On the left, a chart from the Predator dashboard that specifies the distribution of smart contract function calls during a Predator campaign. The red bars indicate the fraction of function calls that passed versus the blue, which indicate the fraction that reverted. The chart allows users to easily see which functions Predator is focusing on as it attempts to uncover possible vulnerabilities. On the right, another dashboard chart — this tracks the RL agent's exploration of novel states based on the number of mutated storage key sets encountered.

1.2. Example: Flagging Abusable Functionality

TestMachine's latest updates to the Predator platform have significantly expanded its capabilities to rapidly identify and flag abusable functionality and scam tokens, providing clients with a powerful tool to enhance their token and protocol onboarding processes. By simulating the execution of smart contracts with an AI agent, Predator can now detect a wide range of potential vulnerabilities and malicious behaviors that could compromise the security and integrity of the underlying assets. These are reported in real-time to the users on the Predator dashboard, as depicted in Figure 5. Below we summarize some of the invariants that Predator automatically targets during each campaign.

One of the key features of this enhanced functionality is the ability to identify and flag **blacklisting mechanisms** within smart contracts. Blacklisting allows contract owners to arbitrarily restrict or prohibit certain addresses from interacting with the token, which can be abused to manipulate token holdings and transactions. Predator's AI agent can now simulate a range of scenarios and detect the presence of blacklisting functionality, alerting clients to potential risks associated with the token.

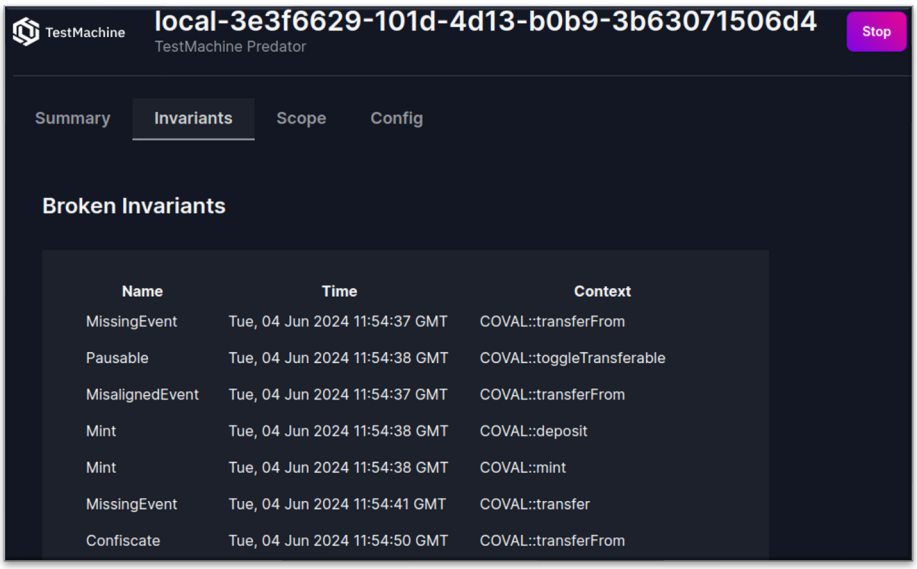

Figure 5: A tab on the Predator dashboard showing invariants broken during a campaign.

Similarly, Predator can now identify and flag the presence of **transaction fees** (TxFees) within smart contracts. While transaction fees can serve legitimate purposes, such as supporting the development and maintenance of the token ecosystem, they can also be abused to extract value from token holders or discourage certain types of transactions. By simulating the execution of the contract and analyzing the fee structure, Predator empowers clients to make informed decisions about the fairness and reasonableness of the token's transaction fees.

Pausing functionality is another critical aspect of smart contract security that Predator can now effectively address. Malicious actors may attempt to exploit pausing mechanisms to freeze token transfers or disrupt the normal operation of the contract. Predator's AI agent can simulate various pausing scenarios and identify potential vulnerabilities that could allow unauthorized parties to control the contract's state, enabling clients to assess the risks associated with pausing functionality before onboarding a new token.

Minting and external calls are two additional areas where Predator's enhanced capabilities can provide significant value to clients. Unauthorized minting of tokens can lead to inflation and devaluation, while external calls to untrusted contracts can introduce security vulnerabilities and compromise the integrity of the token. By simulating the execution of the contract and analyzing the minting and external call behaviors, Predator can flag potential issues and help clients ensure that the token adheres to best practices and security standards.

Predator's ability to detect **missing or fake transfer events** is another crucial advancement in its token security analysis capabilities. Transfer events are essential for accurately tracking token balances and transactions, and their absence or manipulation can lead to significant discrepancies and vulnerabilities. Predator's AI agent can now simulate token transfers and verify the presence and accuracy of the corresponding events, providing clients with greater assurance in the integrity of the token's accounting and transaction history.

Predator can also now identify and flag **arbitrary confiscation and upgradeability risks** within smart contracts. Arbitrary confiscation allows contract owners to seize or freeze token balances without the consent of the token holders, while upgradeability mechanisms can be abused to introduce malicious code or alter the contract's behavior without proper oversight. By simulating these scenarios and analyzing the contract's code, Predator empowers clients to identify and mitigate these risks before onboarding a new token or protocol.

These capabilities of TestMachine's Predator platform represent a significant leap forward in the automated detection of abusable functionality and scam tokens. By leveraging advanced AI and simulation techniques, Predator can now provide clients with a comprehensive and efficient means of assessing the security and integrity of new tokens and protocols. This empowers clients to make informed decisions about which assets to onboard, reducing the risk of introducing vulnerabilities or malicious behaviors into their ecosystems. As a result, clients can more rapidly and confidently expand their token offerings, while maintaining the highest standards of security and trust for their users.

1.3. Example: Economic Modeling

As TestMachine continues to evolve and expand its capabilities, we recognize that ensuring the security and integrity of smart contracts and DeFi protocols requires more than just identifying vulnerabilities in smart contract code. While our current focus on detecting and mitigating code-level issues has proven highly effective, we understand that the complex and dynamic nature of the DeFi ecosystem demands a more comprehensive approach to security. To that end, TestMachine is actively developing new tools and methodologies to address the growing threat of "second order" vulnerabilities, which can arise from the interplay of economic incentives, market conditions, and user behavior.

One of the key areas we are exploring is the modeling and simulation of different economic scenarios and asset price movements. By leveraging advanced machine learning techniques, such as deep reinforcement learning and generative adversarial networks (GANs), we aim to create realistic and adaptive models that can predict how changes in market conditions, liquidity, or user sentiment might impact the stability and security of DeFi protocols. These models will allow us to identify potential arbitrage opportunities, liquidation cascades, price manipulation vectors, and other economic vulnerabilities that could be exploited by malicious actors to drain funds, disrupt operations, or undermine the integrity and stability of the protocol. **Predator has already — without explicit instruction on our part — found examples of economic vulnerabilities in DeFi protocols** — *e.g.*, it was able to engineer an unanticipated arbitrage opportunity on a popular DeFi lending platform and drain value from its token vault.

In addition to economic modeling, we are also developing new tools to analyze and simulate the governance mechanisms of DeFi protocols. Many protocols rely on token-based voting systems to make critical decisions about protocol upgrades, parameter changes, or fund allocation. However, these governance systems can be vulnerable to various forms of manipulation, such as vote-buying, collusion, or bribery. By building sophisticated models of governance dynamics and

incentive structures, TestMachine will be able to identify potential weaknesses in these systems and recommend strategies for mitigating them, such as implementing quadratic voting, reputation-based voting, or other mechanisms to ensure fair and secure decision-making. The continued development of these “second order” vulnerability detection systems, which we believe will broadly expand our market base, is contingent on further funding of TestMachine, in one form or another.

2. / Jungle Overview

The "Jungle" is a critical element of TestMachine's architecture, serving as the underlying infrastructure that enables the Predator AI to focus on its core task of identifying and exploiting vulnerabilities in smart contracts. By abstracting away the complexities of the Ethereum Virtual Machine (EVM), data endpoints, and other low-level details, Jungle provides a simplified and standardized environment for Predator to operate in, greatly enhancing its efficiency and effectiveness.

One of the key benefits of Jungle is that it allows our AI engineers to concentrate on developing and refining the machine learning algorithms that power Predator, without having to worry about the intricacies of the blockchain infrastructure. This separation of concerns is crucial, as it enables our team to leverage their expertise in artificial intelligence and reinforcement learning to create more sophisticated and adaptive models for detecting and exploiting vulnerabilities, while relying on Jungle to handle the underlying simulation and execution details.

By encapsulating the EVM and other blockchain components within Jungle, TestMachine creates a more controlled, deterministic, and higher-fidelity environment for testing and experimentation. This is particularly important when dealing with complex, multi-agent systems like DeFi protocols, where the behavior of the system can be highly sensitive to small changes in the underlying conditions. By providing a stable and reproducible simulation environment, Jungle enables rigorous and systematic testing of Predator's strategies, without being confounded by the variability and unpredictability of the live blockchain.

Another key advantage of Jungle is that it facilitates the integration of additional data sources and analytical tools into the Predator AI pipeline. As shown in Figure 6, Jungle includes components like Alchemy and Chainstack, which provide access to real-time blockchain data and enable more sophisticated, realistic analyses of smart contract behavior. By leveraging these data sources within the Jungle environment, our AI engineers can create more informed and context-aware models for detecting vulnerabilities, considering factors like transaction history, token flows, and market conditions.

Finally, the modular and extensible architecture of Jungle allows us to easily incorporate new simulation capabilities and testing scenarios as our understanding of the DeFi ecosystem evolves. For example, as TestMachine develops new tools for modeling economic incentives, governance mechanisms, and user behavior, we can seamlessly integrate these components into the Jungle infrastructure, providing a richer and more realistic environment for Predator to operate in. This flexibility is

critical for ensuring that `TestMachine` remains at the forefront of smart contract security, as the DeFi landscape continues to grow and change.

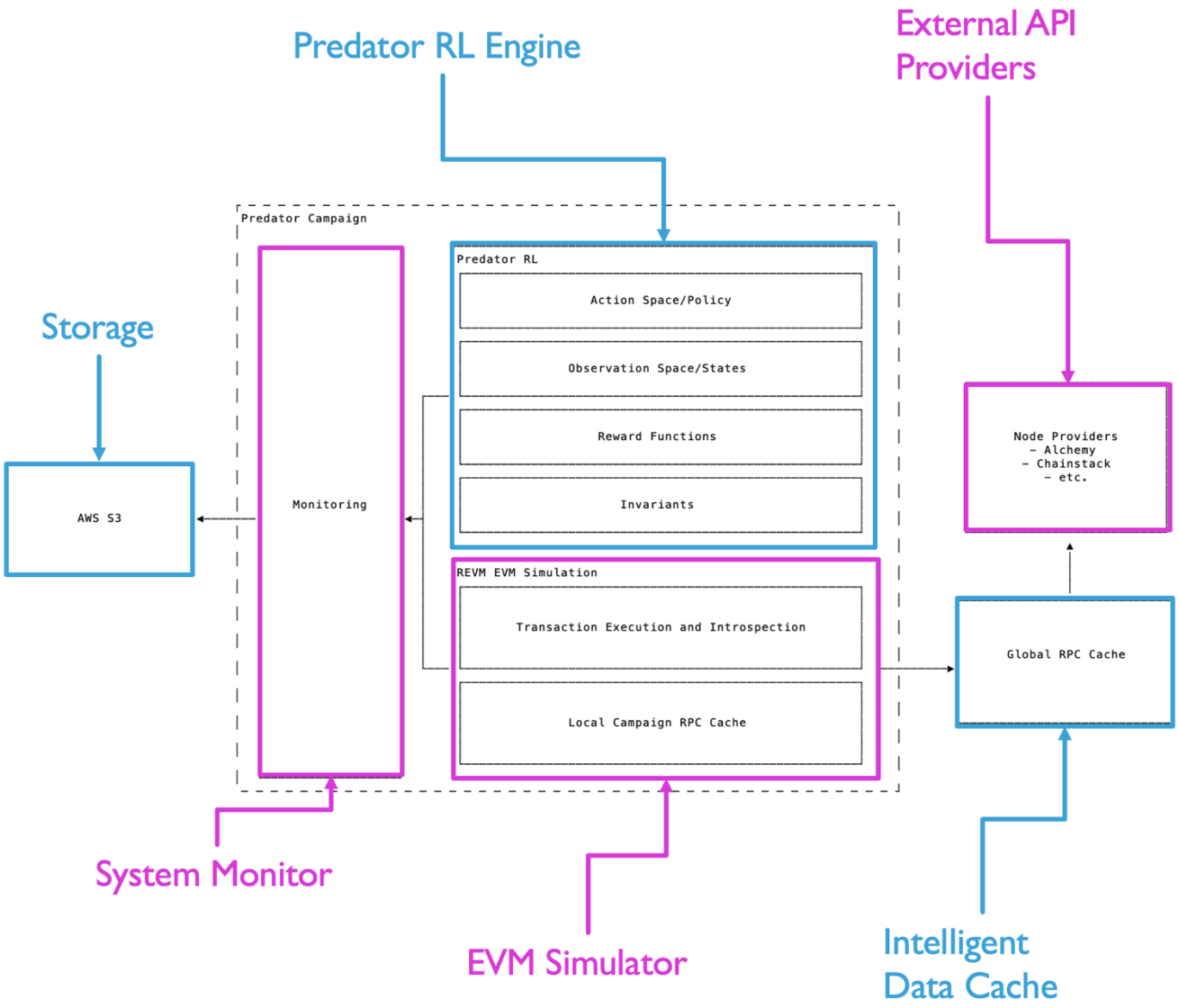

Figure 6: The `Jungle` infrastructure. `Predator` algorithms run within a `Jungle` environment, which abstracts away the details of the EVM simulations, data endpoint providers (such as Alchemy, Chainstack, etc.), monitoring and logging systems, etc. This allows our machine learning engineers to focus on developing RL algorithms without the need to handle the tedious details of the blockchain simulations.