Treasury Proposal: Phink Fuzzer Development

Proponent: 15aPQ8NRfgTHhMV436JZeVGQTCAQrCak7mGCDP3inJjRpV57 (**SRLabs**)
Beneficiary: 15aPQ8NRfgTHhMV436JZeVGQTCAQrCak7mGCDP3inJjRpV57 (**SRLabs**)
Date: [02.07.2024]
Requested USDT: 198,100
Short description: Creating an ink! smart contract fuzzer
Project Category/Type: Software development
Previous treasury proposals: NA

1. Context of the proposal

1.1. Background and Motivation

The security of ink! is fundamental for the integrity and reliability of smart contracts developed within the Polkadot ecosystem. ink! is a domain-specific language designed for writing smart contracts that run on the Substrate framework. Given its role in enabling decentralized finance (DeFi) applications, supply chain management solutions, and other critical blockchain-based services, ensuring the security and robustness of ink! smart contracts are paramount.

As part of our ongoing commitment to enhancing the security and reliability of the Polkadot ecosystem, Security Research Labs (SRLabs) proposes the development of Phink, a next-generation fuzzing tool for ink! smart contracts. This initiative stems from our extensive experience in blockchain security and our recognition of the limitations in current smart contract testing methodologies.

Previous efforts, such as the Inkscope-fuzzer, have laid a foundation but fall short in providing comprehensive code coverage and systematic exploration of smart contract code paths. Our goal with Phink is to fill these gaps by using a coverage-guided fuzzing backend and implementing a robust set of smart contract invariants that can be tested. Phink aims to address these gaps by incorporating advanced coverage-guided fuzzing techniques and smart mutations to increase the effectiveness of fuzzing campaigns. By leveraging established fuzzers like AFL++, Phink will facilitate effective discovery of potential smart contract vulnerabilities. This tool will support both developer-specific and common smart contract invariant, allowing for testing of native code conditions and critical inputs.

Value for the Community:

The development of Phink represents a significant advancement in the security infrastructure of the Polkadot smart contract landscape. By providing a more comprehensive and systematic approach to ink! contract fuzz testing, Phink will help developers identify and mitigate vulnerabilities earlier in the development process, thereby enhancing the overall security and reliability of smart contracts. This will be particularly beneficial for DeFi applications and supply chain management solutions, which require the highest level of security. Additionally, Phink will be an open-source tool, encouraging contributions and usage from the community. We will create detailed documentation and tutorials to help smart contract developers easily and effectively test their code.

1.2. Team Background and Credentials

The Phink project is led by an SRLabs team of experienced engineers and security experts with roots in the blockchain industry. The team has years of experience in Polkadot blockchain security, with a specific focus on security auditing.

Our security engineers have extensive backgrounds in fuzzing and Rust programming, having developed numerous fuzzers during audit engagements, including the prominent open-source substrate-runtime-fuzzer project within the Polkadot ecosystem. The maintainer of AFL++, the currently most effective fuzzer on Google's fuzzbench and one of the most widely used fuzzing tools in the industry is also a notable addition to the SRLabs Phink engineering team. His expertise in fuzzing technology and overall contributions to the security industry bring invaluable insights and capabilities to the Phink project.

SRLabs has extensive experience in end-to-end security assurance, with a specialized focus on Substrate components. We focus on the security assessment of all parachain and relay chain components, including runtime (business logic), outer node services (networking, XCM, bridges, WASM, EVM, Precompiles), consensus mechanisms, and smart contract audits (ink! and Solidity), as well as their configuration.

Since 2019, our team has provided blockchain assurance and collaborated with various teams within the Polkadot ecosystem through the Substrate Builders Program Security Assurance Program. We have conducted in-depth audits for approximately 20 ecosystem teams, including Polkadot, complemented by continuous security assurance. Our auditing process involves thorough manual code reviews supported by in-house dynamic and static analysis tools. Over the past few years, we have developed and maintained ziggy, a fuzzer manager and orchestration framework for Rust projects, in addition to the substrate-runtime-fuzzer.

Our next goal is to combine our expertise in the Substrate framework and fuzzing knowledge by developing our in-house fuzzer for ink! smart contracts. Our tool will address improvement needs identified in ink! fuzzers currently under development, namely by employing coverage-based fuzzing techniques that develop tested mutation strategies and test at a fast rate. Through this proposal, we seek funding for Phink, a fuzzer that will make bug detection more comprehensive and efficient while being easy to implement at an early stage.

2. Problem statement and value proposition

The use of ink! in coding smart contracts for sensitive applications such as DeFi and supply chain management necessitates thorough code auditing to identify logic bugs and runtime errors. To enhance the accuracy and reliability of testing during the development phase of ink! smart contracts, the community has turned to developing fuzzing tools. Despite the progress made by projects like the Inkscope-fuzzer, certain limitations remain that our proposal aims to address.

Inkscope-fuzzer, as a property-based fuzzing tool, focuses on generating test inputs based on randomness to validate properties within a scope defined by the tester. While this approach allows for the verification of special input conditions, it lacks coverage guidance. Coverage-guided fuzzers have demonstrated significant success in various applications, including the fuzzer developed for Substrate, by systematically exploring new code paths and ensuring comprehensive code coverage. By incorporating similar, non-random input-based methodologies, we aim to enhance overall code coverage and facilitate the exploration of new code paths, this will effectively help to identify more security issues.

To further improve upon existing tools, we propose the development of a new fuzzing tool, Phink, which will address the need for more systematic and adaptive security testing approaches. Phink will generate smart mutations that build on previous iterations, periodically eliminating obsolete inputs, and minimizing the prompt fuzzing corpus to include only relevant inputs. This will allow for both developer-specific and common smart contract invariants, enabling the testing of native conditions in the code and critical inputs configurable by smart contract developers. Phink will be open-source, encouraging community contributions and improvements.

Phink will be implemented in Rust and will leverage fuzzers such as AFL++, libAFL and Honggfuzz, incorporating the latest coverage-based fuzzing methodologies. Our solution will be free from ink! version dependencies, allowing developers to use their own runtime, storage, and state, thus enhancing the tool's flexibility and applicability. Additionally, we plan to develop a user interface to extend the tool's ease of use, enabling visualization of code coverage on smart contracts.

By developing Phink, we want to empower the Polkadot community to automatically assess their smart contracts through comprehensive and adaptive fuzzing prompts based on smart mutations and including custom and default invariants. This will significantly improve the reliability and security of ink! smart contracts in critical applications.

3. Proposal objective and solution

The objective of this project is to develop a new fuzzing tool, Phink, for ink! smart contracts that addresses identified limitations in existing solutions and improves the detection of security vulnerabilities.

The primary objectives of the Phink project are the following:

1. **Enhanced security testing**: Develop a fuzzing tool that improves the detection of security vulnerabilities in ink! smart contracts through advanced coverage-guided techniques and smart mutations.
2. **Comprehensive invariant testing**: Testing and verifying system invariants are critical for creating reliable ink! smart contracts. Invariants are conditions that must always hold true within a protocol. Defining and rigorously testing these invariants helps developers avoid introducing bugs and enhances long-term code robustness. However, developing the internal knowledge and processes to create and maintain invariants is challenging, and only a few development teams have integrated invariants into their lifecycle. In this light, our proposal includes a dedicated focus on invariant development for ink! smart contracts.
3. **Broad compatibility**: Ensure compatibility with multiple ink! versions and custom runtimes, providing flexibility and future-proofing.
4. **User-friendly interface**: Create an intuitive user interface to facilitate ease of use and adoption among smart contract developers.
5. **Open source and easily accessible:** As an open-source tool, Phink encourages community contributions and continuous improvement. We create all resources necessary for developers to start fuzzing without the need for prior experience, including how-to tutorial and comprehensive documentation.
6. **Performance and reliability**: Validate the fuzzing tool's effectiveness through performance evaluation against well-known ink! smart contracts.

4. Use cases

The proposed fuzzer will have multiple application areas:

- **Development**: In smart contract development, it will assist developers in identifying and resolving logic bugs and vulnerabilities during the development phase. The fuzzer can be used as an additional means of testing smart contract assertions and properties, as well as offering a different security perspective.
- **Auditing**: For auditing, it will provide auditors with a powerful tool to uncover potential security issues in smart contracts, enhancing the auditing process by

automatically testing the smart contract with a variety of inputs and scenarios, making it easier to identify potential exploits.

- **Research**: Additionally, it will enable security researchers to explore and improve smart contract security methodologies.

5. Milestones and Deliverables

Our team has already developed an initial prototype for Phink. To achieve the objectives of this project, we will develop the full set of features for Phink. The deliverables will be organized into the following work packages.

M1. Project Setup

Establish the foundational infrastructure and define the project parameters to ensure smooth project execution.

Deliverables:

- **Project repository setup**: Create a centralized public repository for collaboration on GitHub. This will facilitate version tracking, release management, reviews and seamless collaboration among team members and the community.
- **Infrastructure setup**: Setup the necessary development and deployment infrastructure. This includes configuring pipelines for deployment, setting up a development environment and establishing community communication using Discord.

M2. Base ink! fuzzing harness development

Develop a baseline ink! smart contract fuzzing harness that can generate mutated contract inputs based on code coverage and execution data to maximize testing effectiveness.

Deliverables:

- **Fuzz harness implementation**: Set up the fuzzing environment and integrate the ink! smart contract interface with AFL++/libAFL/honggfuzz. This involves creating a common interface to arbitrary ink! smart contract functions, allowing mutated inputs from the AFL++, libAFL and Honggfuzz backend to be passed to smart contract functions. Additionally, implement mechanisms for capturing and analyzing outputs and state changes to identify potential vulnerabilities.
- **Custom WASM instrumentation**: Implement custom WASM ink! smart contract instrumentation to capture coverage information.

- **Invariant interface**: Develop an interface within the harness to allow developers to define and execute custom invariants. This interface will also support the execution of invariants identified and developed in M4.
- **Comprehensive bug detection**: Detect a wide range of bugs, including runtime errors. This ensures that Phink can handle a variety of potential issues beyond just invariants.

M3. Runtime support and compatibility

Ensure that Phink is compatible with a variety of runtime environments, enhancing its flexibility and future-proofing its use.

Deliverables:

- **Integration with default minimalistic runtime**: Develop compatibility with Substrate's default pallet_contract runtime module
- **Custom runtime compatibility module**: Create a module that allows Phink to be used with custom runtime environments. This will involve developing interfaces and configuration options to adapt Phink to different runtime setups.
- **Wide ink! version compatibility**: Ensure Phink's compatibility with multiple versions of ink!, providing support for previous and current versions.

M4. Invariant development

Enable systematic identification, development and testing of core ink! smart invariants to ensure robust ink! smart contracts.

Deliverables:

- **Invariant identification**: Identify potential common invariants at both the ink! function level and system level. Define pre-conditions for these invariants and document them.
- **Invariant implementation**: Implement the identified invariants within the fuzzing harness invariant interface. Develop specific test cases that incorporate these invariants ensuring that the fuzzing process checks for these conditions.
- **Invariant testing and refinement:** Run the invariants through the fuzzing process on a base set of ink! smart contracts. Analyze the results and refine the invariants and test cases based on the findings.

M5. Blockchain state tracking

Implement state tracking mechanism to enhance fuzzing accuracy and coverage by maintaining precise blockchain states.

Deliverables:

- **Snapshot-based fuzzing module**: Develop a module that uses snapshotting to allow for efficient state resets for fuzzing. This will increase the speed at which the fuzzer is able to test the contract under test.
- **State management interface**: Design and implement an interface to manage ink! smart contract states within the fuzzing harness. This interface will provide developers with tools to control the state transition during the fuzzing process.

M6. Performance evaluation

Assess the performance and effectiveness of the Phink fuzzing tool through benchmarking and analysis.

Deliverables:

- **Benchmarking amongst publicly available Ink! smart contracts**: Select a series of well-known ink! smart contracts and use them to benchmark the performance of Phink. This will involve setting up test environments and running the fuzzer against these contracts to collect coverage and general performance data
- **Performance metrics collection**: Collect key performance metrics, such as code coverage, number of unique bugs found and the execution speed. These metrics will provide insights into the effectiveness and efficiency of Phink.
- **Performance evaluation report**: Compile the benchmarking results and performance metrics into a report

M7. User interface development

Develop a user-friendly interface to facilitate ease of use and accessibility for smart contract developers

Deliverables:

- **User interface design**: Design an intuitive console-based user interface that allows developers to interact with Phink easily.
- **User Interface Implementation:** Develop the designed interface, integrating it with the backend functionalities of Phink. This will involve coding the front-end

console-based UI components, setting up data bindings and ensuring a responsive design.

- **Visualization component:** Incorporate code coverage visualization, state transitions and invariant testing results within the interface. This component will provide developers with insights into the fuzzing process and help them identify optimization potential for their fuzzing campaigns.

M8. Documentation and reporting

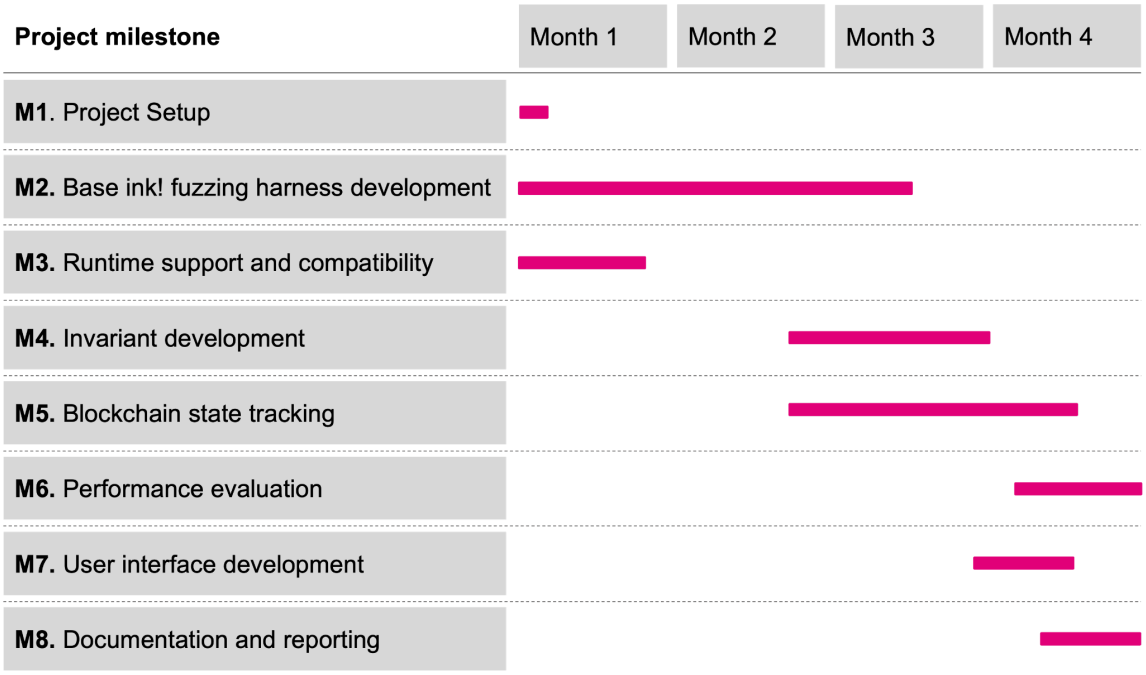
Provide documentation and reporting to support the use and maintenance of Phink

Deliverables:

- **Complete user manual**: Develop a user manual that provides detailed instructions on how to install, configure and use Phink. This manual will include step-by-step guides, troubleshooting tips and best practices
- **Technical documentation**: Create technical documentation that details the internal workings of Phink, including its architecture, modules and interface. This documentation will be aimed at developers who wish to use the tool or contribute to it.
- **Project report and final presentation:** Compile the findings, progress and results of the project into a report. This report will summarize the project's achievements, challenges and future directions, providing an overview of the project's outcomes. The final outcome will be presented to the community (e.g. Polkadot Decoded, Sub0, etc.). We will ensure that all documentation and reports are made publicly available, and the tool will remain open-source to encourage community contributions and transparency.

6. Timeline

This section provides a visual representation of the timeline detailing when each of the project milestones will be achieved. Please note that this timeline is based on our current estimates and is subject to change as circumstances evolve.



7. Budget

The on-chain submission will request funds in USDT.

Tasks	Hourly rate	Hours	Effort
M1. Project setup	$140	15	$2,100
M2. Base ink! fuzzing harness development		400	$56,000
M3. Runtime support and compatibility		150	$21,000
M4. Invariant development		200	$28,000
M5. Blockchain state tracking		300	$42,000
M6. Performance evaluation		150	$21,000
M7. User interface development		100	$14,000
M8. Documentation and reporting		100	$14,000
Project total		1,415	**$198,100**

8. Payment conditions

This section outlines the specific conditions related to the payment of this proposal to fund the development of the Phink fuzzer.

Total Amount in USDT:

- Total Amount Requested: The total amount requested for the complete development process is 198,100 USDT.

Payment Installments and Milestones:

- **Full payment upfront**: The full payment of the total amount will be made upfront upon approval of the proposal. This approach facilitates the immediate commencement of the development activities by SRLabs, ensuring there are no delays in the project's timeline.
- **Commitment to deliver milestones:** While the payment is made upfront, SRLabs commits to delivering all outlined milestones and reports as per the schedule described in section 5. This ensures that all deliverables are met according to the agreed timeline and quality standards.

Treasury Proposal and Reporting:

- **Reporting on Milestones:** SRLabs will submit a report summarizing the milestone achievements upon the completion of the development. This report will provide detailed insights into the progress and accomplishments of the project.
- **Public Availability:** The report will be made publicly available to ensure transparency and accountability to the Polkadot community. This openness aligns with our principle that, as a community-funded project, the outcomes belong to the community, ensuring that everyone has access to the results and can benefit from this collective knowledge.