Polkadot

Proposal: Patron Milestones 2-7 Development

Proponent: 14wVYf1pE3CSWxFamrVT6TTYbRqn68Nbic14sTo13si7YQvk
Date: 04.08.2023
Requested DOT: $187,560 paid in two installments, one on the initial approval date and the other upon the report delivery. The price for each installment will be calculated on those dates to DOT using the EMA7 rate on Subscan

---

Summary

Patron is a tool to verify & build smart-contracts in one step. Smart contract verification is crucial for ensuring the security, reliability, and trust-worthiness of dApps and blockchain platforms. With Patron, you can simplify the deployment flow and help make the Polkadot ecosystem more secure.

In this proposal we are describing the next Patron milestones.
In general, we'd like to unite all the work in this roadmap in next main points:

- Utilizing Parity's verification image on a server in an isolated manner to ensure server safety of both our and self-hosted environments
- Implementing a new subcommand, which automatically checks server code hash against the local build code hash using the local source code
- Implementing the Caller UI
- Implementing local build capabilities that don't use Docker, to improve user flow for contract testing cases, where quick builds are required
- ink! Code analyzer integration

The team behind this product is Brushfam, led by professionals such as Markian Ivanichok and Green Baneling who are behind PSP22, PSP34, and PSP37 standards creation, major impact on how ink! works, creation of such infrastructure products for ink! smart contract ecosystem as Sol2Ink, Typechain and OpenBrush itself.

Context

Brushfam's goal is to increase adoption of Polkadot's WASM technology. This can be achieved by providing a supportive ecosystem for new developers. To make this happen, we think there is the need for a robust development infrastructure, including easy-to-use and time-saving products like OpenBrush and educational resources.

Patron Team

Brushfam team, that develops Patron, is led by professionals such as Markian Ivanichok and Yurii Yazupol who were the initial creators of Patron at its beginning. Other team members that are going to work with this proposal:

Yurii Yazupol - Product Lead
Ivan Leshchenko - Blockchain Developer
Nameless Endless - Blockchain Developer
Dominik Krížo - Head of Engineering
Artem Lech - Blockchain Developer
Varex Silver - Blockchain Developer
Matviy Matsipura - Designer
Alina Antropova - Business Developer (Business & Community Developer)

Now sharing the same mission they are working on Brushfam and its infrastructure products.

Our team has a track record of successfully delivered milestones within grants:

Patron Grant (funded by Web3Foundation)

Typechain-Polkadot Grant (funded by Web3Foundation)

Typechain-Polkadot Grant Follow Up (funded by Web3Foundation)

Typechain-Polkadot Grant Follow Up 2 (funded by Web3Foundation)

Sol2ink Grant (funded by Web3Foundation)

Sol2ink Grant Follow Up (funded by Web3Foundation)

OpenBrush Grant (funded by Web3Foundation)

OpenBrush Grant Follow Up (funded by Web3Foundation)

OpenBrush Grant Follow Up 2 (funded by Web3Foundation)

OpenBrush Grant (funded by AlephZero) - covered 10th milestone

Our motivation for developing Patron

Our primary motivation is the mission we set when creating Patron and Brushfam. To make Patron an entry point for developers coming into the ecosystem and businesses building their projects in it. Nowadays, smart contracts are one of the main instruments of development in the blockchain world. As blockchain itself should be open and reliable, we want to create a registry of all available contracts, so everyone could understand the risks and check whether a specific contract is safe or not.

The other side of it - manager of smart contracts, so every developer could manage his own contracts just like using GitHub. It is very important for users to have connection with what developers do, so that will increase trust between them.

Teams that use Patron:

Astar Network

Aleph Zero

Phala Network

Problem statement

1. **Seamless verification**. An ability to seamless get your on-chain deployed contract logic verified and matched with existing source code, using the usual deployment flow without obscure actions.
2. **Cumbersome build/local deploy/debug flow**. No automatic build/deploy/debug flow results in multiple repetitive manual actions.

f.e. current flow:

1. Change is done to the code
2. You are going to the terminal, entering ‘cargo-contract build`
3. now you must locate your compiled wasm file in the UI filesystem
4. Now you must drag to contracts-ui UI, then press the button upload
5. Then initiate and manually pass all arguments
6. Now imagine you have to repeat this process 10x/20x etc pretty cumbersome.

We believe this flow can be automated.

3. **Inefficient contract deployment.** CLI/script deployment usually are not sufficient for local/testnet development while existing UI instruments are rather complicated . Also, there is no common deploy flow for every stage - local/testnet/production, a tool that would combine both CLI automation and UI playground.
4. **Unified contract management.** Deployed smart contract management is currently done with the usage of hard-to-use CLI tools or UI instruments with just the basic features available. Also, developer contract management(UI used during development) and post-deployment contract management are different tools and interfaces(UIs).
5. **Vulnerability research** is done mostly ad-hoc, with no unified platform being available to assist users in covering common vulnerabilities. Ecosystem also lacks automatic vulnerability scanning platform, which will catch common mistakes in smart contracts that could lead to security issues.
6. **Interaction with external resources** (like HTTP APIs) based on events is obscure and has to be implemented manually.

Proposed solution

Database search
Users will have capabilities to perform database queries to retrieve scanned smart contract information, verified source code data, pre-built WASM blobs and JSON metadata values.

Smart contract verification module
As part of our deployment flow, the verification module will provide a reproducible verification environment to build and verify ink! smart contracts.
Users will be able to supply smart contract code and tooling versions, which will be invoked inside of an isolated environment.
After user supplied code build, smart contract explorer users can see verification details (similar to how it's implemented in EtherScan).
To start using the verification module, users will need to pay a fee. This will protect the verification module from abuse.

Smart contract manager
Manager that will allow users to register and deploy their contracts and invoke various actions on existing contracts available on the platform, such as periodic invocation and vulnerability scanning.

Unified ink! smart contract manager solution allows us to provide improved transparency (by building a contract ourselves and publishing it or by verifying an already published contract to match the provided source code), security (integrated vulnerability scanning, audit publishment capabilities) and versatility (periodic contract invocation, integrated scripting features.).
Smart contract manager should provide most (if not all) of its functionality while keeping user's private keys private, without delegating them to Patron.

Deployment tool

As part of our platform, we plan to provide users with a unified deployment tool that builds and publishes smart contracts for popular mainnets, testnets, and the user's local development node.
When used with mainnets, the contract deployment process is done on our platform side using isolated containers and contract verification workflow.
With testnets or local development nodes, we plan to simplify the onboarding process by automatically downloading required tooling and libraries for popular operating systems, ensuring that local builds work out of the box.
Eventually, we plan to unify our tools into Patron CLI, allowing developers to create new ink! contracts, use existing smart contract templates, transpile existing Solidity contracts, or generate Typechain bindings without leaving the same unified interface they will already be familiar with.

Build/compile flow improvement

Our unified deployment tool will also include code watching capabilities, allowing developers to quickly test application locally in an interactive fashion, without wasting time on manual smart contract deployment and instantiation.

Vulnerability scanning and bug bounty program

We plan to integrate security features into our platform by providing users with capabilities to review existing smart contract audits done by third-party companies and eventually provide a platform to audit smart contracts.
Vulnerability scanning can be invoked automatically to detect various common vulnerabilities via pre-configured intrinsics, while still allowing users to review contracts in more detail if necessary.

Scripting functionality

As part of the smart contract manager, we plan to provide a scripting functionality that will allow smart contract developers to access external APIs and implement complex workflows that depend on external data.

Patron will automatically (and in a verifiable way) request external APIs and call user's smart contract methods with data obtained from the response.
This workflow may be executed based on the contract's dispatched events or just by periodic contract calls.

Why Polkadot Network?

We believe that Polkadot is one of the most technological solutions on the market. Our platform can significantly improve the ink! ecosystem by covering transparency and security and providing versatile features, allowing developers and smart contract users to discover, discuss and improve.

We are also developing other infrastructural solutions that are aimed at this strategic goal and help the community of developers and businesses to solve their problems more effectively.

Milestones

Milestone 2

- **Estimated Duration: 3.5 weeks**
- **FTE: 2.6**
- **Costs: 43 680 USD**
- **Covered by Aleph Zero: 18 900 USD**
- **Covered by Polkadot: 24 780 USD**

Summary:

- Server image update, ability to verify code locally against the remote server

Number	Deliverable	Specification
0a.	License	MIT
0b.	Documentation	Extend Patron documentation with newly added features. We will provide API documentation for contributors to get along with the codebase.
1.	Server image update	Utilize Parity's verification image on a

		server in an isolated manner to ensure server safety of both our and self-hosted environments. By utilizing Parity's image we can ensure that `cargo-contract` 's verifiable builds have the same code hash as our own remote builds.
1.1	CLI `patron verify` command	We will implement a new subcommand, which automatically checks server code hash against the local build code hash using the local source code. This will ensure that developer can trust our remote build server, because code hash of the remote build is the same as local machine's local builds.
2.	UI updates	Improve user interface by including helpful guides into the web UI itself.
2.1	Front-end part	Implement new design
2.2	UI/UX design	The UI design can be previewed here.
3.	Delivery	Delivery of completed tasks

Milestone 3

- **Estimated Duration: 4.5 weeks**
- **FTE: 1.8**
- **Costs: 38 880 USD**
- **Covered by Aleph Zero: 13 500 USD**
- **Covered by Polkadot: 25 380 USD**

Summary:

- Contract call interface

Number	**Deliverable**	**Specification**
0a.	License	MIT

0b.	Documentation	Extend Patron documentation with newly added features. We will provide API documentation for contributors to get along with the codebase.
1.	Smart contract method call interface	We will provide users with functionality to perform calls to smart contracts from our UI.
1.1	UI/UX design	Front-end development includes UI/UX design, which incorporates our own style guidelines to streamline user experience while interacting with the contract call interface.
1.2	Method discovery	Parsing of contract metadata will be implemented on the client-side.
1.3	Dynamic generation of call inputs	We will implement a dynamically generated front-end for smart contract invocation purposes.
2.	Delivery	Delivery of completed tasks

Milestone 4

- **Estimated Duration: 3.5 weeks**
- **FTE: 2.6**
- **Costs: 43 680 USD**
- **Covered by Aleph Zero: 18 900 USD**
- **Covered by Polkadot: 24 780 USD**

Summary:

- Build developer environment (Docker-less local build CLI capabilities)

Number	**Deliverable**	**Specification**
0a.	License	MIT
0b.	Documentation	Extend documentation with a description of newly added features

1.	Local build capabilities	We will implement local build capabilities that don't use Docker, to improve user flow for contract testing cases, where quick builds are required. To improve the developer experience itself, we are planning to implement the `watch` command, which will handle the automatic upload, instantiation and debug UI loading.
1.1	Tooling installation guidance	Automatic or guided tool installation will be implemented where possible, simplifying user interaction with CLI.
1.2	CLI and website interface UI/UX	We will design CLI in a way reduces any usage friction as much as possible. Website UI will be extended with local node information.
1.3	CLI `watch` capabilities	Integrate filesystem watch capabilities that will automatically trigger `cargo-contract` build and open contract method call UI.
1.4	Multicontract projects support	We will support projects where multiple crates are providing multiple contracts
1.5	Contract caller local node support front-end	Local Substrate nodes will be supported by our contract caller interface.
2.	Delivery	Delivery of completed tasks

Milestone 5

- **Estimated Duration: 3.5 weeks**
- **FTE: 2.6**
- **Costs: 43 680 USD**
- **Covered by Aleph Zero: 18 900 USD**
- **Covered by Polkadot: 24 780 USD**

Summary:

- ink! Code analyzer integration

Number	Deliverable	Specification
0a.	License	MIT
0b.	Documentation	Extend Patron documentation with newly added features. We will provide API documentation for contributors to get along with the codebase.
1.	ink! Analyzer integration	We will integrate ink! Analyzer tool into Patron for automatic code scanning in remote builds. Using ink! Analyzer, we plan to scan contracts for known vulnerabilities and mistakes, enabling developers to identify errors in contracts more quickly. While the tool itself is new, it's extremely extensible and can be utilized as a simple Rust library. This feature introduced to allow for a more "in-depth" look into the diagnostics themselves. Ink! Analyzer's API is providing an expressive diagnostics API for that task (https://docs.rs/ink-analyzer/latest/ink_analyzer/struct.Analysis.html), which will allow us to integrate it more deeply with our UI and CLI without relying on using unsandboxed tooling or stdout parsing. (demonstration output)
1.1	Builder integration	Build processes will invoke ink! Analyzer tool automatically and collect diagnostics into the database. This feature will assist developers in automatically catching common mistakes without the need for any manual tool installation.

1.2	Front-end display	Diagnostics related to a particular code hash or contract will be displayed on the associated pages.
1.3	Tool UX planning	This task is very important as developer flow starts from CLI. We plan to carefully evaluate user experience for interactions with CLI, optimizing user flow as much as possible. As part of the task, we plan to improve visual output of our commands, with experience of developers and designers in mind.
1.4	Diagnostic messages UI/UX	For diagnostics, an integrated into our current design system interface will be implemented. Diagnostic messages will contain information about the source code line where a warning was issued, severity of such a message and its contents. This information will help developers analyze smart contract problems quickly and easily.
2	Delivery	Delivery of completed tasks

Milestone 6

- **Estimated Duration: 3.5 weeks**
- **FTE: 1.8**
- **Costs: 30 240 USD**
- **Covered by Aleph Zero: 0 USD**
- **Covered by Polkadot: 30 240 USD**

Summary:

- CLI expansion

Number	**Deliverable**	**Specification**
0a.	License	MIT

0b.	Documentation	Extend Patron documentation with newly added features. We will provide API documentation for contributors to get along with the codebase.
1.	Testing guidelines	Core functionality will be covered by a comprehensive unit test suite.
2.	Deployment tool expansion	Deployment tool is to be expanded to support local development workflows, integration with sol2ink, OpenBrush and TypeChain.
2.1	OpenBrush and sol2ink and Typechain	We will build our own wrappers which will encapsulate the tools themselves into a unified CLI utility. For OpenBrush, we will provide pre-made templates for new projects, and for sol2ink - we will provide tools to create new projects from existing Solidity smart contracts.
2.2	Tool UX planning	We plan to carefully evaluate user experience for interactions with CLI, optimizing user flow as much as possible.
3	Delivery	Delivery of completed tasks

Milestone 7

- **Estimated Duration: 4 weeks**
- **FTE: 3**
- **Costs: 57 600 USD**
- **Covered by Aleph Zero: 0 USD**
- **Covered by Polkadot: 57 600 USD**

Summary:

- Scripting (oracle-like) function ability

Number	**Deliverable**	**Specification**

0a.	License	MIT
0b.	Documentation	Extend Patron documentation with newly added features. We will provide API documentation for contributors to get along with the codebase.
1.	Scripting functionality	We will implement scripting functionality for smart contract to interact with off-chain environment.
1.1	Editor	A script editor will be implemented, which will allow user to edit scripts on the website itself.
1.2	Runtime	We will implement a runtime for script to be executed in, which will isolate separate scripts from each other.
2	Delivery	Delivery of completed tasks

Payment conditions

$187,560 paid in two installments, one on the initial approval date and the other upon the report delivery. The price for each installment will be calculated on those dates to DOT using the EMA7 rate on Subscan.

Team members rates

Backend Rust Developer: Our Backend Rust developers are specialized in their field and have a deep understanding of the intricacies involved. Their hourly rate is **$100/hr**, which reflects their expertise and the market rate for professionals of their caliber.

Frontend Developer: The hourly rate for our Frontend developers, who ensure seamless functionality and user experience, is **$80/hr**.

Product Lead: Guiding the direction and strategy of the project, our Product Lead has an hourly rate of **$120/hr**, reflective of their experience and the responsibilities they shoulder.

UI/UX Designer: Crafting the visual and user interaction aspects of our product, our UI/UX designer have an hourly rate of **$75/hr**.

Tester: Ensuring the quality and functionality of our product, our Tester come in with an hourly rate of **$50/hr**.