HashEye

// SMART CONTRACT SECURITY AUDIT

Polkaswap

Security assessment by HashEye · prepared for Soramitsu

HASHEYE AUDITED

PROJECT	Polkaswap
CLIENT	Soramitsu
CATEGORY	Blockchain
PUBLISHED	August 1, 2021
REPORT ID	research-polkaswap-2021-08-01-tg4u2c

This report was produced under HashEye's layered review process — **automated detection**, **pattern correlation**, and **senior manual verification** — with every finding signed off by a human reviewer. Full findings detail and on-chain attestation are available on the report page at hasheye.io/audits/research-polkaswap-2021-08-01-tg4u2c.

Polkaswap Security Assessment August 2, 2021 Prepared For: Iurii Vinogradov | Soramitsu vinogradov@soramitsu.co.jp Pavel Golovkin | Soramitsu golovkin@soramitsu.co.jp Prepared By: Dominik Czarnota | HashEye dominik.czarnota@hasheye.io Artur Cygan | HashEye artur.cygan@hasheye.io

Executive Summary Project Dashboard Code Maturity Evaluation Engagement Goals Coverage Recommendations Summary Findings Summary 1. Ethereum bridge’s failure to check transferFrom return values could facilitate illicit transfers 2. Improper use of ecrecover weakens the bridge’s security 3. Users can register assets with empty name and ticker symbol fields 4. Use of ERC20 tokens that could become inflationary or defl ationary 5. Polkaswap blindly trusts upgradeable ERC20 proxy tokens 6. Peers are not punished for submitting invalid signatures in approve_request 7. Outdated Rust dependencies 8. Ethereum bridge cannot handle chain reorganizations 9. Ethereum bridge does not check transfer results 10. Potential reuse of peer signatures from, and in calls to, the prepareForMigration function 11. Risk of replay attacks across contract instances 12. ABI encodePacked collision 13. Inaccurate description of SwapSuccess event 14. Off-chain worker depends on a single Ethereum data source 15. Sorascan does not show asset IDs that are not present in the system 16. Peers’ secret keys are stored as plaintext in off-chain storage 17. LiquiditySourceType contains mock pools 18. A vector in the liquidity-proxy’s swap extrinsic can be used for network spamming 19. Zero-weight extrinsics can be used to spam the network 20. Unused create_swap extrinsic in technical pallet 21. Sorascan does not accurately display large initial supply values 22. eth-bridge Decoder.next_u8 method could panic if used 23. Non-mintable assets can be created with no initial supply 24. Off-chain worker can panic if the Ethereum API returns a null block_number A. Vulnerability Classifications B. Code Maturity Classifications

C. Code Quality Recommendations D. ERC20 Token Transfer Semantics E. Token Integration Checklist General Security Considerations ERC Conformity Contract Composition Owner Privileges Token Scarcity F. Proof of Concept for Creating a Non-Mintable Asset with an Initial Supply of Zero G. Fuzzing Polkaswap with test-fuzz

Executive Summary

From July 12 to July 30, 2021, Soramitsu engaged HashEye to review the security of the Polkaswap system. HashEye conducted this assessment over six person-weeks, with two engineers working from the sora2-substrate (9d72e6c), sora2-eth (f3cf040), and sora2-frame-pallets (021b1e5) repositories. During the first week of the assessment, we focused on gaining an understanding of the codebase, documentation, and Polkaswap system. We reviewed the Ethereum bridge smart contract, the off-chain components, and the permissions and assets pallets, among others. In the second week, we started reviewing the DEX components: the bonding curve, XYKPool , liquidity-proxy , MCBP , price-tools , pswap-distribution , trading-pair , xor-fee , vested-rewards , and r ewards pallets. In the last week, we also analyzed certain Substrate-related concerns, such as the defi nition of extrinsic weights and the atomicity of the extrinsics. In addition to manually reviewing the code, we leveraged the Polkaswap.io wallet interface, the Sorascan explorer, and custom Python scripts to conduct dynamic testing of various scenarios. We also tested certain parsing code using the test-fuzz fuzzing framework, which did not find any bugs. Our review resulted in 24 findings ranging from high to informational severity. Several high-severity issues result from the incorrect handling of certain operations in the Ethereum bridge contract, such as token transfers; specifically, a user could execute a transfer and a subsequent withdrawal without actually transferring any funds, draining value from the Polkaswap system. Since the system is live, we reported those issues to Soramitsu immediately upon discovering them, and the Soramitsu team disabled the problematic tokens in the off-chain workers, all of which are currently controlled by the team. Other high-severity issues involve bridge contract peer signatures, which in certain cases can be reused or replayed across contract instances. In addition, the off-chain bridge component has no mechanism for handling Ethereum blockchain reorganizations and depends on a single source of truth for Ethereum event data; as a result, an Ethereum reorganization involving numerous blocks or an attack against that source of data could lead to an incorrect chain state or a loss or theft of funds. As part of the audit, we identified non-security-related issues, which are detailed in Appendix C , and compiled a list of ERC20 token transfer semantics, provided in Appendix D , which helped us evaluate the risks stemming from the transfer-related findings. Appendix E provides guidance on interactions with arbitrary tokens, and Appendix G details the process of fuzzing the Polkaswap codebase and includes a patch that adds a fuzzing harness.

The Polkaswap system is complex, and its security is heavily dependent on the actions of trusted peers and the external API it uses as an Ethereum data source. Additionally, many aspects of the

Polkaswap system lack proper documentation, which makes it harder to reason about the system. HashEye recommends that Soramitsu take the following steps: • Fix the issues detailed in this report. • Expand the test suite. • Expand and centralize the documentation, ensuring that it details the system's algorithms and formulas. • Implement a process for ensuring that new supported ERC20 tokens have the expected semantics before they are used in the system. • Develop an automated mechanism for tracking upgradeable ERC20 tokens to ensure that their expected semantics do not change as new versions are introduced. After implementing the recommendations provided in this report, Soramitsu should perform additional security assessments to ensure that the expected security properties of the system hold. © 2021 HashEye Soramitsu Polkaswap Assessment | 4

Project Dashboard Application Summary Name Polkaswap Versions sora2-substrate (9d72e6c) sora2-eth (f3cf040) sora2-frame-pallets (021b1e5) Type Rust Platform Linux Engagement Summary Dates July 12–July 30, 2021 Method Full knowledge Consultants Engaged 2 Level of Effort 6 person-weeks Vulnerability Summary Total High-Severity Issues 7 □□□□□□□ Total Medium-Severity Issues 5 □□□□□ Total Low-Severity Issues 4 □□□□ Total Informational-Severity Issues 5 □□□□□ Total Undetermined-Severity Issues 3 □□□ Total 24 Category Breakdown Auditing and Logging 3 □□□ Data Exposure 1 □ Data Validation 16 □□□□□□□□□□ □□□□□□ Denial of Service 1 □ Patching 1 □ Undefi ned Behavior 2 □□ © 2021 HashEye Soramitsu Polkaswap Assessment | 5

Total 24 © 2021 HashEye Soramitsu Polkaswap Assessment | 6

Code Maturity Evaluation Category Name Description Access Controls Weak. We found several issues involving the peer signature hashing schemes, which could enable the reuse of signatures or allow an attacker to add the bridge contract itself as a trusted peer. The system also lacks documentation on its use of validators and peers and its permission system, which enforce its access controls. Arithmetic Moderate. We did not find any arithmetic-related issues. However, there are many places in the code that use saturating arithmetic rather than checked. The arithmetic will require further investigation, as additional issues (such as rounding or economic issues) might be present. Assembly Use Not applicable. The project does not use assembly. Centralization Weak. In principle, the SORA Network ensures decentralization by using Substrate's built-in nominated proof-of-stake consensus algorithm. However, the Ethereum bridge contract is managed by trusted peers that are fully controlled by the Soramitsu team, which also controls all SORA Network validators. Additionally, the system depends on a single Ethereum data source and would therefore be significantly affected by a hack of that source. Code Stability Moderate. The code was undergoing changes during the audit and might continue to evolve before reaching its final version. Upgradeability Satisfactory. We did not find any issues directly related to upgradeability. Function Composition Satisfactory. The codebase's functions are narrow and have clear purposes. However, the number of Rust modules and the lack of architecture-related documentation make it difficult to track the components' interactions. Front-Running Satisfactory. We did not find any front-running issues. Monitoring Not considered. Specification Moderate. The system is very complex. While it has some documentation in the form of Medium posts, certain aspects of the system, such as the complex swap process, the peer system, © 2021 HashEye Soramitsu Polkaswap Assessment | 7

and the eth-bridge and technical pallets, are not well documented. The documentation should also be aggregated into one source. Testing & Verification Moderate. The unit tests are not exhaustive, and many check only happy paths. The unit tests should ensure that expected states are triggered when errors occur. © 2021 HashEye Soramitsu Polkaswap Assessment | 8

Engagement Goals The engagement was scoped to provide a security assessment of the Polkaswap system. Specifically, we sought to answer the following questions: • Could malicious users steal funds or manipulate asset prices in unexpected ways? • Does the Ethereum bridge contract handle supported tokens appropriately? • Can peer signatures be bypassed or manipulated? • Does the Ethereum bridge handle blockchain reorganizations properly? • Is it possible for users to perform swaps in unintended ways? • Are the arithmetic operations correct and prevented from overflowing? • Are storage changes applied appropriately and reverted in case of a failure? • Is it possible to spam or abuse the network? • Can third-party integrators rely on the system to emit events where appropriate? Coverage General use of Substrate. We performed a manual code review to check whether the Polkaswap system uses the Substrate features properly. We analyzed the runtime parameters it uses, the emitted events, the configuration of extrinsics such as weights and dispatches (including the means of verifying their origins), and its validation of unsigned transactions. We also examined the use of cryptographic primitives and looked for transaction atomicity issues. Ethereum bridge. We performed a manual code review of the Ethereum bridge smart contract and the off-chain components and ran the static analysis tool Slither on the Solidity code. We also analyzed the transfer and upgradeability semantics of the tokens supported by the bridge, which helped us assess the risks to the system and identify issues that required an immediate fix from the Soramitsu team.

DEX system. We performed a manual review of the liquidity sources in the bonding-curve-pool , multicollateral-bonding-curve-pool , and XYKPool pallets and analyzed most of the pallets used by the pools, namely the dex-api , dex-manager , assets , liquidity-proxy , price-tools , pswap-distribution , technical , rewards , and xor-fees pallets. We also used the Polkaswap.io wallet interface to execute dynamic testing of various swap processes; we then used the Sorascan explorer to check the results of that testing. Because of the time constraints of the audit, we reviewed the complex swap functionality only briefl y; it will therefore be necessary to conduct additional reviews to ensure that all swap implications work as expected.

Permissions system. We did not find any issues in the permissions system. However, we identified an unused enum case, the Mode::Forbid , which the Soramitsu team indicated will be deleted. We recommend ensuring that related changes do not enable bypassing of the system’s permission checks. Key management. We performed a manual review of the peer key management code.

Recommendations Summary This section aggregates all the recommendations made during the engagement. Short-term recommendations address the immediate causes of issues. Long-term recommendations pertain to the development process and long-term design goals. Short Term □ In the SORA Network, disable the four ERC20 tokens (BAT, HT, CHSB, and cUSDC) with transfer or transferFrom functions that return false upon a failure but still cause the Ethereum bridge to emit a Deposit or a Withdrawal event. Re-enable those tokens after the issue has been fixed on the Ethereum bridge’s side. TOB-PSWAP-001 , TOB-PSWAP-009 □ Prevent the Ethereum bridge contract from using invalid signatures in signature counts. Additionally, ensure that it properly validates the signatures of new peers to prevent the 0x0 address from being added as a peer. TOB-PSWAP-002 □ Add an empty-string check to the is_symbol_valid and is_name_valid functions to prevent the creation of assets with an empty ticker symbol or name field. TOB-PSWAP-003 □ Clearly document the bridge semantics to inform users that the bridge is not designed to use inflationary or deflationary ERC20 tokens. TOB-PSWAP-004 □ Disable support for upgradeable (proxy) tokens in the Polkaswap system. (See Appendix D for a list of these tokens.) Then implement an automated upgrade-tracking mechanism and have the system pause interactions with upgraded tokens until they have received approval by consensus. TOB-PSWAP-005 □ Add a mechanism (such as slashing) for punishing peers who submit invalid signatures when approving requests in the Ethereum bridge component. TOB-PSWAP-006 □ Update the vulnerable and unmaintained dependencies in the sora2-substrate codebase. This will prevent the exploitation of a known vulnerability against the SORA Network. TOB-PSWAP-007 □ Add code to eth-bridge to detect Ethereum reorganizations by comparing block hashes. Ensure that the code fetches blocks starting from the last processed block and compares that block’s hash to the hash of the corresponding block (the block at the same

height) returned by the API. If the hashes differ, a reorganization has occurred. Finally, design and document a reorganization-handling strategy. TOB-PSWAP-008 □ Upgrade the Ethereum bridge smart contract and fix the signature replay issue stemming from the prepareForMigration function. Add used mapping checks to prepareForMigration and shutDownAndMigrate , as well as all other Ethereum bridge functions, to protect them from signature reuse/replay. Additionally, until the signature hashing issues have been fixed, ensure that calls to the prepareForMigration function pass in a salt of a txHash value already present in the contract's used mapping. This will prevent users of the system, which is already live, from reusing signatures and adding the bridge contract as a peer, which could render the contract inoperable. TOB-PSWAP-010 □ Use the contract’s address and the chainID of the current transaction in all signature schemes to ensure that signatures cannot be reused across contract instances or after a chain fork. TOB-PSWAP-011 □ Use abi.encode instead of abi.encodePacked to encode data for signature checking, especially in the addNewSidechainToken and addEthNativeToken functions. This will prevent hash collisions caused by the hashing of multiple dynamic types. While this issue is currently mitigated by the used[txHash] check, we still recommend fixing it in case the code is reused. TOB-PSWAP-012 □ Update the description of the technical pallet's SwapSuccess event to indicate that it takes only one argument. TOB-PSWAP-013 □ Enable the off-chain worker to fetch Ethereum events from multiple sources, and cross-validate the values it returns. Then require validators to use at least two Ethereum sources, including one self-hosted Ethereum node. Finally, document the process of adding additional sources. These steps will mitigate the risk of system outages or theft in the event that the Ethereum source to which the off-chain worker connects becomes malicious or unavailable. TOB-PSWAP-014 □ Fix the Sorascan blockchain explorer so that it displays unknown asset IDs, includes their hex values, and marks them as nonexistent. TOB-PSWAP-015 □ Avoid storing peers’ secret key material in plaintext on the drive. Store private keys (encrypted via symmetric encryption) with a password, and decrypt each key in memory when the corresponding peer is initialized. Ensure that these passwords are delivered in a safe way, such as through standard input. To automate the deployment process, store passwords on another server in the internal network and configure a deployment bot to use the secure shell

(SSH) protocol to start or restart peers. TOB-PSWAP-016 © 2021 HashEye Soramitsu Polkaswap Assessment | 12

□ Change the LiquiditySourceType so that mock pools are compiled only in a test environment. Use the #[cfg(test)] Rust annotation to make that change. TOB-PSWAP-017 □ Implement appropriate data validation for the selected_source_types vector in the liquidity-proxy pallet's swap extrinsic. Ensure that the network rejects calls if the vector is too long or if it contains duplicates. Additionally, consider accounting for the vector's length in calculations of the extrinsic's base weight. TOB-PSWAP-018 □ Benchmark the extrinsics that have a base weight set to zero. Then, based on the results of that benchmarking, change the weights so that they cannot be used to spam the network. TOB-PSWAP-019 □ Remove the technical pallet's create_swap extrinsic, which is unused and redundant. This will decrease the attack surface of the pallet. TOB-PSWAP-020 □ Use an arbitrary-precision decimal library to handle amount types. This will ensure that the Sorascan explorer displays the correct initial supply value even when the value is very large. TOB-PSWAP-021 □ Either remove the Decoder.next_u8 method from the eth-bridge pallet or change it so that it uses the u8::try_from method for decoding, as shown in figure 22.3. That change will prevent the method from panicking (and possibly causing a denial of service) if it is called with untrusted input. TOB-PSWAP-022 □ Add a minimum initial supply requirement to the assets pallet's register extrinsic. Additionally, consider adding a garbage collection method to handle assets that have no trading pairs and have not been used for a long time. TOB-PSWAP-023 □ Change the off-chain worker so that it no longer expects block_number values returned by the Ethereum API to be non-null. Additionally, ensure that the system always checks the values returned by external APIs. TOB-PSWAP-024 Long Term □ Migrate the Ethereum bridge smart contract to a new version and add a check for the return values of token.transferFrom to its sendERC20ToSidechain function. Ensure that the bridge emits a Deposit event only when the transfer succeeds (when its return value is true). TOB-PSWAP-001 □ Use Echidna or Manticore to ensure that invalid signatures cannot be used in the Ethereum bridge smart contract. TOB-PSWAP-002 © 2021 HashEye Soramitsu Polkaswap Assessment | 13

□ Perform extensive negative testing of all data validation code. TOB-PSWAP-003 □ Review the Token Integration Checklist and implement its recommendations to make sure that ERC20 tokens used by the bridge behave as expected. TOB-PSWAP-004 □ To disincentivize trusted actors from destabilizing the network, ensure that there are always mechanisms in place for punishing trusted actors who misbehave. TOB-PSWAP-006 □ Use the cargo - audit tool to ensure that the dependency versions used by the sora2-substrate project are free of known vulnerabilities, and integrate the tool into the CI/CD pipeline. TOB-PSWAP-007 □ Implement a strategy for automated reorganization handling. After detecting a reorganization, the algorithm should calculate which events need to be unapplied and which new events are missing from the blockchain. Additionally, design a strategy for handling negative balances, which can occur when events are unapplied. TOB-PSWAP-008 □ Migrate the Ethereum bridge smart contract to a new version and add a check for the return value of transfer calls to its receiveByEthereumAssetAddress and shutDownAndMigrate functions. TOB-PSWAP-009 □ Thoroughly document and test the signature hashing schemes. Consider using EIP-712: Ethereum typed structured data hashing and signing as a hashing structure. TOB-PSWAP-010 , TOB-PSWAP-011 □ Use the static analyzer Slither to find code in which abi.encodePacked is used with more than one dynamic type. TOB-PSWAP-012 □ Avoid relying on a single third-party source when fetching data that influences the chain state. TOB-PSWAP-014 □ Use a secret management solution such as HashiCorp Vault or AWS' Key Management Service. Note that currently, HashiCorp Vault does not support secp256k1 signatures; however, there are ongoing eff orts to add that support, tracked in issue hashicorp/vault#459 4 and pull request hashicorp/vault#11469 . We recommend tracking those eff orts and using the service once secp256k1 support has been implemented. TOB-PSWAP-016 □ Ensure that the staging and production environments do not use any code that should live only in tests. TOB-PSWAP-017 © 2021 HashEye Soramitsu Polkaswap Assessment | 14

□ Add tests to the liquidity-proxy pallet's swap extrinsic to ensure that it properly validates the selected_source_types vector and disallows duplicate sources and excessive vector lengths. TOB-PSWAP-018 □ Deploy only those extrinsics that will be used. TOB-PSWAP-020 □ Test the Sorascan explorer against edge-case values such as very small and very large values. The Sorascan explorer likely uses the Number type, which would explain why it displays rounded values. Testing the explorer against edge cases will help determine whether the values it displays are accurate. TOB-PSWAP-021 □ Use a technique such as fuzzing to test all encoding and decoding functions against edge-case inputs. TOB-PSWAP-022 □ Add unit tests to the off-chain worker codebase to ensure that it correctly handles all invalid and edge-case values fetched from the Ethereum API. TOB-PSWAP-024 © 2021 HashEye Soramitsu Polkaswap Assessment | 15

Findings Summary # Title Type Severity 1 Ethereum bridge's failure to check transferFrom return values could facilitate illicit transfers Data Validation High 2 Improper use of ecrecover weakens the bridge's security Data Validation High 3 Users can register assets with empty name and ticker

symbol fields Data Validation Informational 4 Use of ERC20 tokens that could become inflationary or defl ationary Data Validation Medium 5 Polkaswap blindly trusts upgradeable ERC20 proxy tokens Data Validation Medium 6 Peers are not punished for submitting invalid signatures in approve_request Data Validation Undetermined 7 Outdated Rust dependencies Patching Undetermined 8 Ethereum bridge cannot handle chain reorganizations Data Validation High 9 Ethereum bridge does not check transfer results Data Validation High 10 Potential reuse of peer signatures from, and in calls to, the prepareForMigration function Data Validation High 11 Risk of replay attacks across contract instances Data Validation High 12 ABI encodePacked collision Data Validation Informational 13 Inaccurate description of SwapSuccess event Auditing and Logging Informational 14 Off-chain worker depends on a single Ethereum data source Data Validation High 15 Sorascan does not show asset IDs that are not present in the system Auditing and Logging Informational

16 Peers’ secret keys are stored as plaintext in off-chain storage Data Exposure Medium 17 LiquiditySourceType contains mock pools Undefi ned Behavior Low 18 A vector in the liquidity-proxy’s swap extrinsic can be used for network spamming Data Validation Medium 19 Zero-weight extrinsics can be used to spam the network Denial of Service Medium 20 Unused create_swap extrinsic in technical pallet Undefi ned Behavior Undetermined 21 Sorascan does not accurately display large initial supply values Auditing and Logging Low 22 eth-bridge Decoder.next_u8 method could panic if used Data Validation Low 23 Non-mintable assets can be created with no initial supply Data Validation Informational 24 Off-chain worker can panic if the Ethereum API returns a null block_number Data Validation Low

1. Ethereum bridge’s failure to check transferFrom return values could facilitate illicit transfers Severity: High Difficulty: Low Type: Data Validation Finding ID: TOB-PSWAP-001 Target: sora2-eth/contracts/Bridge.sol Description The Ethereum bridge’s sendERC20ToSidechain function (figure 1.1) does not check the values returned by certain ERC20 tokens’ token.transferFrom function, which may return false (rather than reverting) when transfers of those tokens fail. In such cases, the bridge would still emit a Deposit event, which would then be processed by the SORA Network off-chain worker and logged by the Polkaswap system. As such, an attacker could execute the sendERC20ToSidechain function by calling token.transferFrom with an ERC20 token that does not revert upon a failure; the attacker, without having sent any funds, could then use the funds deposited into the SORA Network. As part of the audit, we analyzed 55 ERC20 tokens owned by the Ethereum bridge and found 4 that could be used to exploit this issue: BAT, HT, CHSB, and cUSDC. However, it is also possible that tokens implemented through a proxy contract could be migrated to a version with different semantics. Appendix D includes a list of these 55 tokens and details their transfer return value semantics. /** * Send ERC-20 token to sidechain. * * @param to destination address on the sidechain * @param amount amount to sendERC20ToSidechain * @param tokenAddress contract address of token to send */ function sendERC20ToSidechain (bytes32 to , uint amount , address tokenAddress) external shouldBeInitialized shouldNotBePreparedForMigration { IERC20 token = IERC20(tokenAddress); require (token.allowance(msg.sender , address (this)) ≥ amount, "NOT ENOUGH DELEGATED TOKENS ON SENDER BALANCE"); bytes32 sidechainAssetId = _sidechainTokensByAddress[tokenAddress]; if (sidechainAssetId ≠ "" || _addressVAL == tokenAddress || _addressXOR == tokenAddress) { ERC20Burnable mtoken = ERC20Burnable(tokenAddress); mtoken.burnFrom(msg.sender , amount); } else { require (acceptedEthTokens[tokenAddress], "The Token is not accepted for transfer to sidechain"); token.transferFrom(msg.sender , address (this), amount); } emit Deposit(to, amount, tokenAddress, sidechainAssetId);

} Figure 1.1: The sendERC20ToSidechain function (sora2-eth/contracts/Bridge.sol#L265-L291) Exploit Scenario An attacker approves a transfer of BAT, an ERC20 token, from his Ethereum account to the SORA Network Ethereum bridge, deliberately setting a transfer allowance that exceeds the balance of his account. The attacker then sends a transaction to the bridge, calling the sendERC20ToSidechain function to transfer BAT tokens to an account in the SORA Network; the amount of the transfer exceeds his balance but not the allowance amount. The bridge calls the BAT token’s transferFrom function ; the execution fails, but the bridge emits a Deposit event, which is processed by the SORA Network’s off-chain worker. The attacker can then use the funds on the SORA Network without having actually made a transfer. Recommendations Short term, in the SORA Network, disable the four ERC20 tokens (BAT, HT, CHSB, and cUSDC) with transfer functions that return false upon a failure but still cause the Ethereum bridge to emit a Deposit event. Re-enable those tokens after the issue has been fixed on the Ethereum bridge’s side. Long term, migrate the Ethereum bridge smart contract to a new version and add a check for the return value of token.transferFrom to its sendERC20ToSidechain function. Ensure that the bridge emits a Deposit event only when the transfer succeeds (when its return value is true).

2. Improper use of ecrecover weakens the bridge’s security Severity: High Difficulty: High Type: Data Validation Finding ID: TOB-PSWAP-002 Target: sora2-eth/contracts/Bridge.sol Description The bridge contract fails to check whether the address returned by the ecrecover function is the null address. An attacker could leverage this defi ciency to reduce the number of valid signatures required to execute administrative operations. The Ethereum bridge contract allows a set of peers to perform administrative operations by submitting ECDSA signatures. These operations include adding new peers or ERC20 tokens to (or removing them from) the bridge. Signatures are validated by the recoverAddress function: function recoverAddress (bytes32 hash , uint8 v , bytes32 r , bytes32 s) private pure returns (address) { bytes32 simple_hash = keccak256 (abi.encodePacked("\x19Ethereum Signed Message:\n32" , hash)); address res = ecrecover(simple_hash, v, r, s); return res; } Figure 2.1: The recoverAddress function (sora2-eth/contracts/Bridge.sol#L265-L291) However, the code does not properly check the return value of ecrecover , which can be 0x0 if the signature provided to the function is invalid. As stated in the Solidity documentation , ecrecover “recover[s] the address associated with the public key from [an] elliptic curve signature or return[s] zero on [an] error.” The addPeerByPeer function allows peers to add a new address, even the 0x0 address, as a peer:

function addPeerByPeer (address newPeerAddress , bytes32 txHash , uint8 [] memory v, bytes32 [] memory r, bytes32 [] memory s) public shouldBeInitialized returns (bool) { require (used[txHash] == false); require (checkSignatures(keccak256 (abi.encodePacked(newPeerAddress, txHash, _networkId)), v, r, s), "Peer signatures are invalid"); addPeer(newPeerAddress); used[txHash] = true ; emit ChangePeers(newPeerAddress, false); return true ; } Figure 2.2: The addPeerByPeer function (sora2-eth/contracts/Bridge.sol#L302-L324) If the 0x0 address is added as a peer, an invalid signature can be used to execute any administrative operation, weakening the security of the bridge. Exploit Scenario The 0x0 address is accidentally added as a peer. Eve, a malicious peer, exploits this mistake to include one invalid signature that will be accepted as valid, making it easier for her to reach the number of signatures required to execute administrative functions. Recommendations Short term, prevent the Ethereum bridge contract from using invalid signatures in signature counts. Additionally, ensure that it properly validates the signatures of new peers to prevent the 0x0 address from being added as a peer. Long term, use Echidna or Manticore to ensure that invalid signatures cannot be used in the Ethereum bridge smart contract.

3. Users can register assets with empty name and ticker symbol fields Severity: Informational Difficulty: Low Type: Data Validation Finding ID: TOB-PSWAP-003 Target: pallets/assets/src/lib.rs Description The SORA Network allows any user to register an asset by providing a name and ticker symbol for it. The code validates name and ticker symbol values (figure 3.1) but permits the use of empty strings. /// According to UTF-8 encoding, graphemes that start with byte 0b0XXXXXXX belong /// to ASCII range and are of single byte, therefore passing check in range 'A' to 'Z' /// and '0' to '9' guarantees that all graphemes are of length 1, therefore length check is valid. pub fn is_symbol_valid (symbol: & AssetSymbol) → bool { symbol. 0. len() ⩽ ASSET_SYMBOL_MAX_LENGTH && symbol . 0 .iter() .all(|byte| (b'A' ..= b'Z').contains(&byte) || (b'0' ..= b'9').contains(&byte)) } /// According to UTF-8 encoding, graphemes that start with byte 0b0XXXXXXX belong /// to ASCII range and are of single byte, therefore passing check in range 'A' to 'z' /// guarantees that all graphemes are of length 1, therefore length check is valid. pub fn is_name_valid (name: & AssetName) → bool { name. 0. len() ⩽ ASSET_NAME_MAX_LENGTH && name. 0. iter().all(|byte| { (b'A' ..= b'Z').contains(&byte) || (b'a' ..= b'z').contains(&byte) || (b'0' ..= b'9').contains(&byte) || byte == & b' ' }) } Figure 3.1: The is_symbol_valid and is_name_valid functions (pallets/assets/src/lib.rs#L744-L766) Recommendations Short term, add an empty-string check to the is_symbol_valid and is_name_valid functions to prevent the creation of assets with an empty ticker symbol or name field. Long term, perform extensive negative testing of all data validation code.

4. Use of ERC20 tokens that could become inflationary or deflationary Severity: Medium Difficulty: Low Type: Data Validation Finding ID: TOB-PSWAP-004 Target: sora2-eth/contracts/Bridge.sol Description If the bridge uses an ERC20 token that is inflationary, is defl ationary, or has any kind of dynamic supply or balance, its bookkeeping may be severely affected. The bridge contract was not designed for tokens that can suddenly change the value returned by balanceOf or can transfer an unexpected number of tokens. When a user makes a deposit, the balance of the contract and the number of tokens deposited are used in several operations: /** * Send ERC-20 token to sidechain. * * @param to destination address on the sidechain * @param amount amount to sendERC20ToSidechain * @param tokenAddress contract address of token to send */ function sendERC20ToSidechain (bytes32 to , uint amount , address tokenAddress) external shouldBeInitialized shouldNotBePreparedForMigration { IERC20 token = IERC20(tokenAddress); require (token.allowance(msg.sender , address (this)) ⩾ amount, "NOT ENOUGH DELEGATED TOKENS ON SENDER BALANCE"); bytes32 sidechainAssetId = _sidechainTokensByAddress[tokenAddress]; if

(sidechainAssetId ≠ "" || _addressVAL == tokenAddress || _addressXOR == tokenAddress) { ERC20Burnable mtoken = ERC20Burnable(tokenAddress); mtoken.burnFrom(msg.sender , amount); } else { require (acceptedEthTokens[tokenAddress], "The Token is not accepted for transfer to sidechain"); token.transferFrom(msg.sender , address (this), amount); } emit Deposit(to, amount, tokenAddress, sidechainAssetId); } Figure 4.1: The sendERC20ToSidechain function (sora2-eth/contracts/Bridge.sol#L265-L291) The bridge emits a Deposit event with the expected amount value; if inflationary tokens were being used, though, the value might not be correct. The following widely used ERC20 tokens can cause issues:

• Tether (as well as other tokens) can charge a fee for each transfer, meaning that users will receive fewer tokens than expected. Currently, this fee is set to zero, but if it is increased, Tether will become a deflationary token. • Ampleforth executes daily rebases on contracts or increases the total supply (and therefore the balance of each user) so that its expected price will be pegged to USD 1. Currently, it does not appear that any of the tokens supported by the Polkaswap system are defl ationary or inflationary or could be used to exploit this finding. However, certain of the tokens use a proxy contract and could therefore be upgraded, as detailed in TOB-PSWAP-005 . Exploit Scenario Alice deploys a bridge and initializes it to use a token that has a dynamic supply. Users perform deposits and withdrawals. At some point, there is an error in the bookkeeping, so users do not receive the expected number of tokens when executing withdrawals. Recommendations Short term, clearly document this behavior to inform users that the bridge is not designed to use inflationary or defl ationary ERC20 tokens. Long term, review the Token Integration Checklist and implement its recommendations to make sure that ERC20 tokens used by the bridge behave as expected.

5. Polkaswap blindly trusts upgradeable ERC20 proxy tokens Severity: Medium Difficulty: High Type: Data Validation Finding ID: TOB-PSWAP-005 Target: off-chain worker Description The Polkaswap system does not have an automated mechanism for tracking supported tokens implemented via a proxy contract, which can be upgraded; nor can it pause interactions with a token that has been upgraded pending the token’s approval. As part of an upgrade, a token’s semantics may change such that the token violates the Polkaswap system’s assumptions. This can lead to problems such as incorrect accounting or could leave the Polkaswap system vulnerable to theft or a loss of funds. Exploit Scenario A proxy token supported by the Polkaswap system performs an upgrade and changes its semantics, becoming defl ationary. As a result, in each transfer of this token, a fraction of the transferred amount is burned. An attacker finds that Polkaswap cannot handle defl ationary tokens and transfers his tokens to the SORA Network. The ERC20 balance of the attacker’s SORA Network account is then higher than the amount of his deposit, allowing the attacker to spend more tokens than he transferred. Recommendations Short term, disable support for upgradeable (proxy) tokens in the Polkaswap system. (See Appendix D for a list of these tokens.) Then implement an automated upgrade-tracking mechanism and have the system pause interactions with upgraded tokens until they have received approval by consensus.

6. Peers are not punished for submitting invalid signatures in approve_request Severity: Undetermined Difficulty: High Type: Data Validation Finding ID: TOB-PSWAP-006 Target: eth-bridge Description The approve_request extrinsic checks the validity of signatures submitted by peers; however, there is no punishment imposed on peers who submit invalid signatures (figure 6.1). if ! Self ::verify_message(request_encoded.as_raw(), &signature_params, &ocw_public, &author,) { // TODO: punish the peer. return Err (Error::<T>::InvalidSignature.into()); } Figure 6.1: pallets/eth-bridge/src/lib.rs#L1640-L1648 Recommendations Short term, add a mechanism (such as slashing) for punishing peers who submit invalid signatures when approving requests in the Ethereum bridge component. Long term, to disincentivize trusted actors from destabilizing the network, ensure that there are always mechanisms in place for punishing trusted actors who misbehave.

7. Outdated Rust dependencies Severity: Undetermined Difficulty: High Type: Patching Finding ID: TOB-PSWAP-007 Target: sora2-substrate dependencies Description The sora2-substrate project depends on six outdated package versions that contain known vulnerabilities. These vulnerabilities, which we identified using the cargo - audit tool , are listed below with their RUSTSEC advisory numbers. Dependency Version Advisory Description Libsecp256k1 0.3.5 RUSTSEC-2021-0076 (“libsecp256k1 allows overflowing signatures”) This issue allows signature overflows in the Signature::parse and Signature::parse_slice functions, which are used in the off-chain worker and the rewards pallet . Nalgebra 0.19.0 RUSTSEC-2021-0070 (“VecStorage Deserialize Allows Violation of Length Invariant”) This is a memory safety issue involving the deserialization of VecStorage from untrusted input. The sora2-substrate code does not directly use the crate or its VecStorage type. Nalgebra is a dependency of linregress, which is used by the frame-benchmarking crate. Since the crate should be used only for benchmarking, it does not appear that this issue affects the Polkaswap system. Prost-

types 0.7.0 RUSTSEC-2021-0073 (“Conversion from prost_types::Time stamp to SystemTime can cause an overflow and panic”) The current sora2-substrate code does not appear to use this conversion. © 2021 HashEye Soramitsu Polkaswap Assessment | 27

raw-cpuid 8.1.2 RUSTSEC-2021-0013 (“Soundness issues in raw-cpuid ”) There are soundness issues in this crate’s cpuid_count function and as_string method. These issues are present in cranelift-native , an indirect dependency of Substrate. We notified the Substrate developers (through this thread) that the issues should be fixed, so they will hopefully be addressed in a newer release. Cranelift-codegen 0.69.0 RUSTSEC-2021-0067 (CVE-2021-32629) (“ Memory access due to code generation flaw in Cranelift module”) This bug may allow a sandbox escape from a WebAssembly module in an x64 Cranelift back end. We have not investigated this issue thoroughly enough to defi nitively state whether it affects the Polkaswap system; however, we notified Substrate about it in the aforementioned thread . Additionally, nine of the packages used in the project have “allowed warnings,” which means they were either pulled from Cargo packages (for reasons such as vulnerabilities) or are unmaintained. Recommendations Short term, update the vulnerable and unmaintained dependencies in the sora2-substrate codebase. This will prevent the exploitation of a known vulnerability against the SORA Network. Long term, use the cargo - audit tool to ensure that the dependency versions used by the sora2-substrate project are free of known vulnerabilities, and integrate the tool into the CI/CD pipeline. © 2021 HashEye Soramitsu Polkaswap Assessment | 28

8. Ethereum bridge cannot handle chain reorganizations Severity: High Difficulty: High Type: Data Validation Finding ID: TOB-PSWAP-008 Target: eth-bridge Description The SORA Network integrates with Ethereum through the eth-bridge pallet, which monitors events emitted by the bridge contract. The eth-bridge code does not detect Ethereum chain reorganizations or include any strategy for handling them. The code processes all blocks through the highest block except for the newest blocks (specifically the 30 newest blocks, since the CONFIRMATION_INTERVAL constant is set to 30). This reduces the likelihood that the SORA Network will observe a reorganization. Exploit Scenario An attacker deposits ETH by calling the sendEthToSidechain function, and eth-bridge registers the event and adds funds to the attacker’s account on the SORA Network. An Ethereum chain reorganization occurs, and the Deposit event does not exist in the reorganized chain. The funds added to the attacker’s account are not reclaimed by the chain. Recommendations Short term, add code to eth-bridge to detect Ethereum reorganizations by comparing block hashes. Ensure that the code fetches blocks starting from the last processed block and compares that block’s hash to the hash of the corresponding block (the block at the same height) returned by the API. If the hashes differ, a reorganization has occurred. Finally, design and document a reorganization-handling strategy. Long term, implement a strategy for automated reorganization handling. After detecting a reorganization, the algorithm should calculate which events need to be unapplied and which new events are missing from the blockchain. Additionally, design a strategy for handling negative balances, which can occur when events are unapplied. References ● Ethereum Classic Suffers Reorganization That Resembles 51% Attack Amid Miner Complications © 2021 HashEye Soramitsu Polkaswap Assessment | 29

9. Ethereum bridge does not check transfer results Severity: High Difficulty: High Type: Data Validation Finding ID: TOB-PSWAP-009 Target: eth-bridge Description The shutDownAndMigrate and receiveByEthereumAssetAddress functions (figures 9.1–2) use the ERC20 standard transfer function to transfer tokens when migrating a contract or performing a token withdrawal. However, the functions do not check the value returned by the transfer function, which may be false if a transfer fails. Without this check, funds may become locked in the bridge contract, which could result in a contract shutdown; alternatively, the contract could emit a Withdrawal event despite the lockup. As part of the audit, we analyzed 55 ERC20 tokens owned by the Ethereum bridge and found 4 that could be used to exploit this issue: BAT, HT, CHSB, and cUSDC. However, it is also possible that tokens implemented through a proxy contract could be migrated to a version with different semantics. Appendix D includes a list of these 55 tokens and details their transfer return value semantics. function shutDownAndMigrate (/* (...) */) public shouldBeInitialized shouldBePreparedForMigration { // (...) for (uint i = 0 ; i < erc20nativeTokens.length; i++) { IERC20 token = IERC20(erc20nativeTokens[i]); token.transfer(newContractAddress, token.balanceOf(address (this))); } Bridge(newContractAddress).receivePayment{value: address (this).balance}(); initialized_ = false ; emit Migrated(newContractAddress); } Figure 9.1: The shutDownAndMigrate function (sora2-eth/contracts/Bridge.sol#L175-L203) function receiveByEthereumAssetAddress (/* (...) */) public shouldBeInitialized { // (...) if (tokenAddress == address (0)) { used[txHash] = true ; // untrusted transfer, relies on provided cryptographic proof to.transfer(amount); } else { IERC20 coin = IERC20(tokenAddress); used[txHash] = true ; // untrusted call, relies on provided cryptographic proof coin.transfer(to, amount); } emit Withdrawal(txHash); } Figure 9.2: The receiveByEthereumAssetAddress function (sora2-eth/contracts/Bridge.sol#L371-L402) © 2021 HashEye Soramitsu Polkaswap Assessment | 30

Exploit Scenario A user calls receiveByEthereumAssetAddress to request his funds. Because the transfer semantics of the requested ERC20 token violate the bridge contract’s assumptions, the transfer fails. The bridge does not send the funds to the user’s Ethereum account but still emits a Withdrawal event, which is incorrectly accounted for in the SORA Network. Recommendations Short term, in the SORA Network, disable the four ERC20 tokens (BAT, HT, CHSB, and cUSDC) with transferFrom functions that return false upon a failure but still cause the Ethereum bridge to emit a Withdrawal event. Re-enable those tokens after the issue has been fixed on the Ethereum bridge’s side. Long term, migrate the Ethereum bridge smart contract to a new version and add a check for the return value of transfer calls to its receiveByEthereumAssetAddress and shutDownAndMigrate functions.

10. Potential reuse of peer signatures from, and in calls to, the prepareForMigration function Severity: High Difficulty: High Type: Data Validation Finding ID: TOB-PSWAP-010 Target: eth-bridge Description The prepareForMigration function (figure 10.1) does not ensure that a salt has not been used before. It also uses the same data layout for signatures as the addPeerByPeer and removePeerByPeer functions (figure 10.2). This data layout is detailed in the table below. (Note, though, that the salt argument used by the prepareForMigration function corresponds to the txHash field used by addPeerByPeer and removePeerByPeer .) An attacker could therefore replay a valid signature from the prepareForMigration function to call the other functions; for example, an attacker could add the bridge contract address as a peer through the addPeerByPeer function. Because the bridge cannot sign any data, that would make it more difficult to perform peer-approved operations. It would also be possible for an attacker to replay a signature from addPeerByPeer or removePeerByPeer in prepareForMigration ; however, because of a check in the prepareForMigration function for the bridge contract’s address, the attacker would need to call those functions with the bridge address as the newPeerAddress or peerAddress (depending on the function). Function prepareForMigration addPeerByPeer removePeerByPeer First argument address thisContractAddress (must be the bridge address) address newPeerAddress address peerAddress Second argument bytes32 salt bytes32 txHash (verified as unique through the used[txHash] == false check) Third argument bytes32 _networkId (set when the contract is initialized)

function prepareForMigration (/* (...) */) public shouldBeInitialized shouldNotBePreparedForMigration { require (preparedForMigration_ == false); require (address (this) == thisContractAddress); require (checkSignatures(keccak256 (abi.encodePacked(thisContractAddress, salt, _networkId)), v, r, s), "Peer signatures are invalid"); preparedForMigration_ = true ; emit PreparedForMigration(); } Figure 10.1: The prepareForMigration function (sora2-eth/contracts/Bridge.sol#L144-L162) function addPeerByPeer (/* (...) */) public shouldBeInitialized returns (bool) { require (used[txHash] == false); require (checkSignatures(keccak256 (abi.encodePacked(newPeerAddress, txHash, _networkId)), v, r, s), "Peer signatures are invalid"); addPeer(newPeerAddress); used[txHash] = true ; emit ChangePeers(newPeerAddress, false); return true ; } function removePeerByPeer (/* (...) */) public shouldBeInitialized returns (bool) { require (used[txHash] == false); require (checkSignatures(keccak256 (abi.encodePacked(peerAddress, txHash, _networkId)), v, r, s), "Peer signatures are invalid"); removePeer(peerAddress); used[txHash] = true ; emit ChangePeers(peerAddress, true); return true ; } Figure 10.2: The addPeerByPeer and removePeerByPeer functions (sora2-eth/contracts/Bridge.sol#L302-L358) Exploit Scenario The Ethereum bridge is operating with 10 trusted peers, but 3 of them lose access to their accounts. Instead of adding new peers, the remaining peers prepare a bridge migration, calling the prepareForMigration function with 7 valid peer signatures. An attacker reuses those signatures and calls the addPeerByPeer function to add the bridge contract as a peer. After these operations, there are 11 peers, so the system requires 8 valid peer signatures for every action. However, because there are not enough valid peer signatures to finish the migration (through the shutDownAndMigrate function), the system will remain permanently inoperable.

Recommendations Short term, upgrade the Ethereum bridge smart contract and fix the signature replay issue stemming from the prepareForMigration function. Add used mapping checks to prepareForMigration and shutDownAndMigrate , as well as all other Ethereum bridge functions, to protect them from signature reuse/replay. Additionally, until the signature hashing issues have been fixed, ensure that calls to the prepareForMigration function pass in a salt of a txHash value already present in the contract's used mapping. This will prevent users of the system, which is already live, from reusing signatures and adding the bridge contract as a peer, which could render the contract inoperable. Long term, thoroughly document and test the signature hashing schemes. Consider using EIP-712: Ethereum typed structured data hashing and signing as a hashing structure.

11. Risk of replay attacks across contract instances Severity: High Difficulty: High Type: Data Validation Finding ID: TOB-PSWAP-011 Target: eth-bridge Description All of the Ethereum bridge's signature schemes lack mechanisms for preventing signature replay attacks across contract deployments or chain forks. None of the signature hashing schemes include any information about the contract's instances (figure 11.1). As a result, signatures can be reused across multiple contract deployments. keccak256 (abi.encodePacked(newToken, ticker, name, decimals, txHash, _networkId)) keccak256 (abi.encodePacked(thisContractAddress, salt, _networkId)) keccak256 (abi.encodePacked(thisContractAddress, newContractAddress, salt, erc20nativeTokens, _networkId)) keccak256 (abi.encodePacked(name, symbol, decimals, sidechainAssetId, txHash, _networkId)) keccak256 (abi.encodePacked(newPeerAddress, txHash, _networkId)) keccak256 (abi.encodePacked(peerAddress, txHash, _networkId)) keccak256 (abi.encodePacked(tokenAddress, amount, to, from, txHash, _networkId)) keccak256 (abi.encodePacked(sidechainAssetId, amount, to, from, txHash, _networkId)) keccak256 (abi.encodePacked("\x19Ethereum Signed Message:\n32" , hash)) Figure 11.1: The signature schemes used in the Ethereum bridge contract Moreover, while the signature hashing schemes use a _networkId variable meant to refl ect the current ID of the network, this variable is set only once, by the constructor, and will not change in the event of a fork after the contract deployment (figure 11.2). /** EVM netowrk ID */ bytes32 public _networkId ; // (...) constructor (address [] memory initialPeers, address addressVAL , address addressXOR , bytes32 networkId) { // (...) _networkId = networkId; } Figure 11.2: The bridge contract constructor sets the networkId once (sora2-eth/contracts/Bridge.sol#L44-L70). Exploit Scenario • Alice and Bob are the signers when a peer wants to add a new peer.

• Eve deploys a second version of the Ethereum bridge contract. • Eve convinces Bob and Alice to sign a transaction adding Eve as a trusted peer in the contract instance. • Eve reuses the signature in the original bridge contract and becomes a trusted peer in that contract as well. Recommendations Short term, use the contract's address and the chainID of the current transaction in all signature schemes to ensure that signatures cannot be reused across contract instances or after a chain fork. Long term, thoroughly document and test the signature hashing schemes. Consider using EIP-712: Ethereum typed structured data hashing and signing as a hashing structure.

12. ABI encodePacked collision Severity: Informational Difficulty: High Type: Data Validation Finding ID: TOB-PSWAP-012 Target: eth-bridge Description The addEthNativeToken and addNewSidechainToken functions use the abi.encodePacked Solidity function and pass in two consecutive strings, which could lead to a hash collision. We set the severity of this finding to informational because the used[txHash] check currently prevents exploitation of this issue. The addEthNativeToken function passes in ticker and name strings, and the addNewSidechainToken function passes in name and symbol strings (figure 12.1). function addEthNativeToken (address newToken , string memory ticker, string memory name, uint8 decimals , bytes32 txHash , uint8 [] memory v, bytes32 [] memory r, bytes32 [] memory s) public shouldBeInitialized { require (used[txHash] == false); // (...) require (checkSignatures(keccak256 (abi.encodePacked (newToken, ticker, name , decimals, txHash, _networkId)), // (...) } function addNewSidechainToken (string memory name, string memory symbol, uint8 decimals , bytes32 sidechainAssetId , bytes32 txHash , uint8 [] memory v, bytes32 [] memory r, bytes32 [] memory s) public shouldBeInitialized { require (used[txHash] == false); require (checkSignatures(keccak256 (abi.encodePacked (name, symbol, decimals, sidechainAssetId, txHash, _networkId)),

Figure 12.1: The problematic abi.encodePacked calls (sora2-eth/contracts/Bridge.sol#L113-L133 and L217-L239) This scheme leaves the code vulnerable to a hash collision, in which two calls to these two functions, each with different ticker and name (or name and symbol) strings, would result in the same hash. The Solidity documentation also includes a warning about this issue: Figure 12.2: A warning from the Solidity documentation Recommendations Short term, use abi.encode instead of abi.encodePacked to encode data for signature checking, especially in the addNewSidechainToken and addEthNativeToken functions. This will prevent hash collisions caused by the hashing of multiple dynamic types. While this issue is currently mitigated by the used[txHash] check, we still recommend fixing it in case the code is reused. Long term, use the static analyzer Slither to find code in which abi.encodePacked is used with more than one dynamic type.

13. Inaccurate description of SwapSuccess event Severity: Informational Difficulty: High Type: Auditing and Logging Finding ID: TOB-PSWAP-013 Target: sora2-substrate/pallets/technical/src/lib.rs Description The description of the SwapSuccess event in the technical pallet suggests that the event should have two arguments, an initiator and a finaliser , but the event takes only a single argument (figure 13.1). This event is currently triggered only by a perform_create_swap_unchecked function, which passes in a source argument (figure 13.2). This function is called by two other

functions: • The exchange function in the XYKPool pallet, which passes in a sender argument • A perform_create_swap function in the technical pallet, which passes in a source argument We set the severity of this issue to informational because it does not have a direct security impact on the Polkaswap system. However, it may pose problems for third-party developers who would like to track swap-related events. /// Swap operaction is finalised [initiator, finaliser] . /// TechAccountId is only pure TechAccountId. SwapSuccess(AccountIdOf<T>), Figure 13.1: The SwapSuccess event (sora2-substrate/pallets/technical/src/lib.rs#L364-L366) /// Perform creation of swap, version without validation pub fn perform_create_swap_unchecked (source: AccountIdOf <T>, action: & T ::SwapAction,) → DispatchResult { common::with_transaction(|| { action.reserve(&source)?; if action.is_able_to_claim() { // always true if action.instant_auto_claim_used() { // always true if action.claim(&source) { // always true Self ::deposit_event(Event::SwapSuccess(source)); Figure 13.2: The perform_create_swap_unchecked function (sora2-substrate/pallets/technical/src/lib.rs#L92-L102) Recommendations Short term, update the description of the technical pallet's SwapSuccess event to indicate that it takes only one argument. © 2021 HashEye Soramitsu Polkaswap Assessment | 39

14. Off-chain worker depends on a single Ethereum data source Severity: High Difficulty: High Type: Data Validation Finding ID: TOB-PSWAP-014 Target: sora2-substrate/pallets/technical/src/lib.rs Description The Polkaswap system uses a single source of truth to fetch Ethereum events from the Ethereum bridge contract. There is no mechanism for including additional Ethereum data sources. If this Ethereum data source experienced an outage, the Polkaswap system would also be at risk of an outage; if an attacker hacked the API, the attacker could manipulate the events provided to the system in an attempt to steal funds from it. Exploit Scenario An attacker finds a way to hack the Ethereum data source used by the Polkaswap system. The attacker then modifies the API to serve bogus events to the Polkaswap off-chain worker, changing the chain state to his benefi t. Recommendations Short term, enable the off-chain worker to fetch Ethereum events from multiple sources, and cross-validate the values it returns. Then require validators to use at least two Ethereum sources, including one self-hosted Ethereum node. Finally, document the process of adding additional sources. These steps will mitigate the risk of system outages or theft in the event that the Ethereum source to which the off-chain worker connects becomes malicious or unavailable. Long term, avoid relying on a single third-party source when fetching data that influences the chain state. © 2021 HashEye Soramitsu Polkaswap Assessment | 40

15. Sorascan does not show asset IDs that are not present in the system Severity: Informational Difficulty: High Type: Auditing and Logging Finding ID: TOB-PSWAP-015 Target: Sorascan Description When a transaction is sent with an asset ID that is not present in the system, the Sorascan blockchain explorer will not display the asset ID (figure 15.1). Figure 15.1: Sorascan does not show unknown asset IDs (https://test.sorascan.com/sora-staging/transaction/0x6c8ffa734f89d20862472786f74e27c46b0 3018b240ec75980e011b679a323f6). Recommendations Short term, fix the Sorascan blockchain explorer so that it displays unknown asset IDs, includes their hex values, and marks them as nonexistent. © 2021 HashEye Soramitsu Polkaswap Assessment | 41

16. Peers’ secret keys are stored as plaintext in off-chain storage Severity: Medium Difficulty: High Type: Data Exposure Finding ID: TOB-PSWAP-016 Target: Off-chain storage Description When a peer is initialized, its secret key is loaded as plaintext into Substrate’s off-chain storage (figure 16.1), from which it is fetched when the peer signs outgoing requests (figure 16.2). Because the off-chain storage is not encrypted, the key is eff ectively stored in a file as plaintext. As a result, an attacker who is able to read arbitrary files from the disk or has access to unencrypted backups could steal the key. if let Some (first_pk_raw) = SyncCryptoStore::keys(&*keystore_container.sync_keystore(), eth_bridge::KEY_TYPE) .unwrap() .first() .map(|x| x. 1. clone()) { let pk = eth_bridge::offchain::crypto::Public::from_slice(&first_pk_raw[..]); if let Some (keystore) = keystore_container.local_keystore() { if let Ok (Some (kep)) = keystore.key_pair:: <eth_bridge::offchain::crypto::Pair>(&pk) { let seed = kep.to_raw_vec(); bridge_peer_secret_key = Some (seed); } } } else { log::debug!("Ethereum bridge peer key not found.") } if let Some (sk) = bridge_peer_secret_key { let mut storage = backend.offchain_storage().unwrap(); storage.set(STORAGE_PREFIX, STORAGE_PEER_SECRET_KEY, &sk.encode()); Figure 16.1: The key material is loaded into the off-chain storage (node/src/service.rs#L102-L121) let secret_s = StorageValueRef::persistent(STORAGE_PEER_SECRET_KEY); let sk = secp256k1::SecretKey::parse_slice(&secret_s .get::< Vec < u8 >>() .flatten() .expect("Off-chain worker secret key is not specified."),) .expect("Invalid off-chain worker secret key."); // Signs `abi.encodePacked(tokenAddress, amount, to, txHash, from)`. let (signature, public) = Self ::sign_message(encoded_request.as_raw(), &sk); Figure 16.2: The key is used to sign outgoing requests (pallets/eth-bridge/src/offchain/handle.rs#L70-L79) © 2021 HashEye Soramitsu Polkaswap Assessment | 42

Exploit Scenario An attacker gains access to the filesystem in which a peer is running and steals the peer’s private key. Recommendations Short term, avoid storing peers’ secret key material in plaintext on the drive. Store private keys (encrypted via symmetric encryption) with a password, and decrypt each key in memory when the corresponding peer is initialized. Ensure that these passwords are delivered in a safe way, such as through standard input. To automate the deployment process, store passwords on another server in the internal network and configure a deployment bot to use the secure shell (SSH) protocol to start or restart peers. Long term, use a secret management solution such as HashiCorp Vault or AWS’ Key Management Service. Note that currently, HashiCorp Vault does not support secp256k1 signatures; however, there are ongoing eff orts to add that support, tracked in issue hashicorp/vault#459 4 and pull request hashicorp/vault#11469 . We recommend tracking those eff orts and using the service once secp256k1 support has been implemented. © 2021 HashEye Soramitsu Polkaswap Assessment | 43

17. LiquiditySourceType contains mock pools Severity: Low Difficulty: High Type: Undefi ned Behavior Finding ID: TOB-PSWAP-017 Target: sora2-substrate/common/src/primitives.rs Description The LiquiditySourceType enum contains mock pools (figure 17.1), which should not be present outside of tests but are present on the sora-staging network . We set the severity of this finding to low because we did not find a way to leverage the mock pools in an exploit. However, mock code left in production builds may enable unexpected behavior when the code changes in the future. We tested the pools by trying to execute various swap operations on the sora-staging network; each attempt resulted in an error of “No route exists in a given DEX for given parameters to carry out the swap.” Additionally, we confirmed that the DEXAPI's EnabledSourceTypes storage value contains the XYKPool and MulticollateralBondingCurvePool pools, but not the mock pools. /// Enumaration of all available liquidity sources. #[derive(Encode, Decode, RuntimeDebug, PartialEq, Eq, Copy, Clone, PartialOrd, Ord)] #[cfg_attr(feature = "std" , derive(Serialize, Deserialize))] #[repr(u8)] pub enum LiquiditySourceType { XYKPool, BondingCurvePool, MulticollateralBondingCurvePool, MockPool, MockPool2, MockPool3, MockPool4, } Figure 17.1: The LiquiditySourceType enum (sora2-substrate/common/src/primitives.rs#L449-L461) Exploit Scenario A network upgrade results in code changes that make it possible to use mock pools as a liquidity source. An attacker uses these pools to his benefi t, which would not be possible if the mock pools had not been compiled in a production build. Recommendations Short term, change the LiquiditySourceType so that mock pools are compiled only in a test environment. Use the #[cfg(test)] Rust annotation to make that change. Long term, ensure that the staging and production environments do not use any code that should live only in tests. © 2021 HashEye Soramitsu Polkaswap Assessment | 44

18. A vector in the liquidity-proxy’s swap extrinsic can be used for network spamming Severity: Medium Difficulty: High Type: Data Validation Finding ID: TOB-PSWAP-018 Target: sora2-substrate/pallets/liquidity-proxy/src/lib.rs Description There are insufficient limits on selected_source_types , which is a vector in the liquidity-proxy pallet's swap extrinsic. Specifically, the vector can contain duplicate entries, and there is no check preventing it from becoming too long. Additionally, the vector’s length is not factored into calculations of the extrinsic's base weight (figure 18.1). These defi ciencies leave the system vulnerable to the following scenarios: 1. An attacker could provide a very long selected_source_types vector to spam the network with swap transactions at a relatively low cost, which could lead to a denial of service. 2. An attacker could perform swaps via calls to the generic_split method, passing in the same source multiple times. This method is called in the quote_single function (figure 18.2), which is called by the swap extrinsic’s Self::exchange function. This transaction on the sora-staging test network shows a relatively limited instance of spamming. #[pallet::weight(<T as Config>::WeightInfo::swap((*swap_amount).into()))] pub fn swap (origin: OriginFor <T>, dex_id: T ::DEXId, input_asset_id: T ::AssetId, output_asset_id: T ::AssetId, swap_amount: SwapAmount <Balance>, selected_source_types: Vec <LiquiditySourceType>, filter_mode: FilterMode ,) → DispatchResultWithPostInfo { let who = ensure_signed(origin)?; if Self ::is_forbidden_filter(&input_asset_id, &output_asset_id, &selected_source_types, &filter_mode,) { fail!(Error:: <T>::ForbiddenFilter); } let outcome = Self ::exchange(&who, &who, &input_asset_id, &output_asset_id, swap_amount, © 2021 HashEye Soramitsu Polkaswap Assessment | 45

LiquiditySourceFilter::with_mode(dex_id, filter_mode, selected_source_types),)?; // (...) Figure 18.1: The swap extrinsic (sora2-substrate/pallets/liquidity-proxy/src/lib.rs#L1575-L1625) fn quote_single (/* (...) */) → /* (...) */ { let sources = T::LiquidityRegistry::list_liquidity_sources(input_asset_id, output_asset_id, filter)?; ensure! (!sources.is_empty(), Error::<T>::UnavailableExchangePath); // Check if we have exactly one source ⇒ no split required if sources.len() == 1 { /* (...) */ } // Check if we have exactly two sources: the primary market and the secondary market // Do the "smart" swap split (with fallback) if sources.len() == 2 { /* (...) */ } // Otherwise, fall back to the general source-agnostic procedure based on sampling Self ::generic_split(sources , input_asset_id, output_asset_id, amount, skip_info) } Figure 18.2: The quote_single function, called through the Self::exchange call in the

swap extrinsic (sora2-substrate/pallets/liquidity-proxy/src/lib.rs#L544-L598) Exploit Scenario An attacker sends numerous batch transactions that make many calls to the liquidity-proxy pallet's swap extrinsic, each with a very long selected_source_types vector. In this way, the attacker forces the SORA Network to spend a very long time processing swap transactions, causing a denial of service. Recommendations Short term, implement appropriate data validation for the selected_source_types vector in the liquidity-proxy pallet's swap extrinsic. Ensure that the network rejects calls if the vector is too long or if it contains duplicates. Additionally, consider accounting for the vector's length in calculations of the extrinsic's base weight. Long term, add tests to the liquidity-proxy pallet's swap extrinsic to ensure that it properly validates the selected_source_types vector and disallows duplicate sources and excessive vector lengths. © 2021 HashEye Soramitsu Polkaswap Assessment | 46

19. Zero-weight extrinsics can be used to spam the network Severity: Medium Difficulty: High Type: Denial of Service Finding ID: TOB-PSWAP-019 Target: sora2-substrate/pallets/liquidity-proxy/src/lib.rs Description The following extrinsics have a base weight set to zero and can therefore be used to spam the network, causing a denial of service: • PswapDistribution::claim_incentive • BridgeMultisig::register_multisig • BridgeMultisig::remove_signatory • BridgeMultisig::add_signatory • BridgeMultisig::as_multi_threshold_1 • BridgeMultisig::as_multi • BridgeMultisig::approve_as_multi • BridgeMultisig::cancel_as_multi Note, though, that BridgeMultisig::as_multi can be used for network spamming only if the account ID of the sender is not found in the Accounts storage map (specifically due to unwrap_or(0)). Exploit Scenario An attacker sends numerous batch transactions that make many calls to the pswap-distribution pallet's claim_incentive extrinsic, each for a very low fee. By spamming the network in this way, the attacker causes a denial of service of the SORA Network. Recommendations Short term, benchmark the extrinsics that have a base weight set to zero. Then, based on the results of that benchmarking, change the weights so that they cannot be used to spam the network. © 2021 HashEye Soramitsu Polkaswap Assessment | 47

20. Unused create_swap extrinsic in technical pallet Severity: Undetermined Difficulty: High Type: Undefi ned Behavior Finding ID: TOB-PSWAP-020 Target: sora2-substrate/pallets/technical/src/lib.rs Description The technical pallet contains an unused and redundant create_swap extrinsic. The extrinsic has never been called on the sora-staging test network, even though it exists there . HashEye did not perform an exhaustive analysis of this extrinsic, as it is not present on the sora-mainnet network , and the Soramitsu team indicated that it will be removed. However, based on our investigation, the create_swap extrinsic (figure 20.1) appears to be overly complex and error-prone. It takes an action argument of the SwapAction type (figure 20.2); this type has three values — PairSwapAction , DepositLiquidityAction , and WithdrawLiquidityAction —each of which has multiple fields (figure 20.3). This action is passed to the perform_create_swap function , which then calls the prepare_and_validate and perform_create_swap_unchecked functions. The former mutates certain of the action object's fields, and the latter finalizes the swap operation. The functions are also implemented separately for each action type. This complex logic, along with the fact that the action object is fully controlled by extrinsic callers, makes the create_swap extrinsic error-prone and fragile; it would be even more problematic if the aforementioned aspects of the code were changed in the future. By contrast, other functions that call the prepare_and_validate function pass in a pre-constructed action object in which only certain fields are controlled by the caller. These functions are as follows: • The XYKPool 's exchange function , which uses the PairSwapAction action (and is the only function that calls prepare_and_validate directly, rather than through the perform_create_swap function) • The XYKPool 's deposit_liquidity_unchecked function , which uses the DepositLiquidityAction action • The XYKPool 's withdraw_liquidity_unchecked function , which uses the WithdrawLiquidityAction action #[pallet::call] impl <T: Config > Pallet<T> { #[pallet::weight(<T as Config>::WeightInfo::create_swap())] pub (crate) fn create_swap (origin: OriginFor <T>, action: T ::SwapAction,) → DispatchResultWithPostInfo { let source = ensure_signed(origin)?; let mut action_mut = action; Module::<T>::perform_create_swap(source, & mut action_mut)?; © 2021 HashEye Soramitsu Polkaswap Assessment | 48

Ok (().into()) } } Figure 20.1: The create_swap extrinsic (sora2-substrate/pallets/technical/src/lib.rs#L319-L331) impl technical::Config for Runtime { // (...) type SwapAction = pool_xyk::PolySwapAction<AssetId, AccountId, TechAccountId>; Figure 20.2: The SwapAction type (sora2-substrate/runtime/src/lib.rs#L642-L650) pub struct PairSwapAction <AssetId, AccountId, TechAccountId> { /* 8 fields */ } pub struct DepositLiquidityAction <AssetId, AccountId, TechAccountId> { /* 6 fields */ } pub struct WithdrawLiquidityAction <AssetId, AccountId, TechAccountId> { /* 6 fields */ } pub enum PolySwapAction <AssetId, AccountId, TechAccountId> { PairSwap(PairSwapAction<AssetId, AccountId, TechAccountId>), DepositLiquidity(DepositLiquidityAction<AssetId, AccountId, TechAccountId>), WithdrawLiquidity(WithdrawLiquidityAction<AssetId, AccountId, TechAccountId>), } Figure 20.3: The complex PolySwapAction enum (sora2-substrate/pallets/pool-xyk/src/operations.rs#L51-L88)

Recommendations Short term, remove the technical pallet's create_swap extrinsic, which is unused and redundant. This will decrease the attack surface of the pallet. Long term, deploy only those extrinsics that will be used. © 2021 HashEye Soramitsu Polkaswap Assessment | 49

21. Sorascan does not accurately display large initial supply values Severity: Low Difficulty: Medium Type: Auditing and Logging Finding ID: TOB-PSWAP-021 Target: Sorascan Description If the initial supply of a token is too large a value, the Sorascan explorer will show an incorrect value. For example, we created an asset with an initial supply of 10 quintillion tokens, for which the Asset::register extrinsic displays an incorrect initial supply value (figure 21.1). The sora-staging test network transaction shows the same incorrect value. Because of time constraints and the low impact of this issue, HashEye did not analyze the root cause of the issue. It is also worth noting that the “SORA Network Account” tab on the Polkaswap.io wallet interface displays the correct token supply value after the token’s creation (figure 21.2). Figure 21.1: The Assets::register extrinsic shows an incorrect initial supply value. © 2021 HashEye Soramitsu Polkaswap Assessment | 50

Figure 21.2: The “SORA Network Account” display on the Polkaswap.io wallet interface shows the correct BIGTOKE token amount. Recommendations Short term, use an arbitrary-precision decimal library to handle amount types. This will ensure that the Sorascan explorer displays the correct initial supply value even when the value is very large. Long term, test the Sorascan explorer against edge-case values such as very small and very large values. The Sorascan explorer likely uses the Number type, which would explain why it displays rounded values. Testing the explorer against edge cases will help determine whether the values it displays are accurate. © 2021 HashEye Soramitsu Polkaswap Assessment | 51

22. eth-bridge Decoder.next_u8 method could panic if used Severity: Low Difficulty: High Type: Data Validation Finding ID: TOB-PSWAP-022 Target: sora2-substrate/pallets/eth-bridge/src/util.rs Description If the eth-bridge pallet's Decoder.next_u8 method (figure 22.1) were used in the code, it could panic if provided untrusted input. We set the severity of this finding to low because the code uses neither this method nor its caller, the next_signature_params method. However, if the code were changed and began using those methods, it would be at risk of a panic, as the methods could be called with untrusted input. A n attacker could craft input that, once decoded to a tokens value, would result in a panic, causing a denial of service on the network. For example, the code would panic if the value of a popped token (i.e., a token taken from the tokens vector) exceeded u32::MAX . This is because the next_u8 method, through a call to into_uint() , would convert the popped token value to a U256 type , which can hold unsigned integers of up to 256 bits. Then the method would call the x.as_u32() function on the U256 value. If that value exceeded the u32::MAX value, the as_u32 method would panic when trying to convert it. #[allow(unused)] pub fn next_u8 (& mut self) → Result < u8 , Error<T>> { self .tokens .pop() .and_then(|x| x.into_uint()) .filter(|x| x.as_u32() ⩽ u8 ::MAX as u32) .map(|x| x.as_u32() as u8) .ok_or_else(|| Error:: <T>::InvalidByte.into()) } Figure 22.1: The Decoder.next_u8 method (sora2-substrate/pallets/eth-bridge/src/util.rs#L91-L98) /// Conversion to u32 with overflow checking /// /// # Panics /// /// Panics if the number is larger than 2^32. #[inline] pub fn as_u32 (& self) → u32 { let & $name (ref arr) = self ; if ! self .fits_word() || arr[0] > u32 ::max_value() as u64 { panic! ("Integer overflow when casting to u32") } self .as_u64() as u32 } Figure 22.2: The as_u32 method called on the U256 type (paritytech/parity-common/uint/src/uint.rs#L639-L651) © 2021 HashEye Soramitsu Polkaswap Assessment | 52

Exploit Scenario The code is changed such that it begins using the Decoder.next_u8 method in the off-chain worker. An attacker takes advantage of this change and calls this method with a token value greater than U32::MAX . This causes the off-chain worker to panic, leading to a denial of service on the network. Recommendations Short term, either remove the Decoder.next_u8 method from the eth-bridge pallet or change it so that it uses the u8::try_from method for decoding, as shown in figure 22.3. That change will prevent the method from panicking (and possibly causing a denial of service) if it is called with untrusted input. #[allow(unused)] pub fn next_u8 (& mut self) → Result < u8 , Error<T>> { Ok (u8 ::try_from(self .tokens .pop() .and_then(|x| x.into_uint()) .ok_or(Error::<T>::InvalidUint)?,) .map_err(|_| Error::<T>::InvalidByte)?) } Figure 22.3: A fixed version of the Decoder.next_u8 method Long term, use a technique such as fuzzing to test all encoding and decoding functions against edge-case inputs. © 2021 HashEye Soramitsu Polkaswap Assessment | 53

23. Non-mintable assets can be created with no initial supply Severity: Informational Difficulty: High Type: Data Validation Finding ID: TOB-PSWAP-023 Target: sora2-substrate/pallets/eth-bridge/src/util.rs Description The assets pallet’s register extrinsic allows users to create assets that have an initial supply of zero—that is, non-mintable assets that cannot be used. The Polkaswap.io wallet code includes a check that mitigates this issue; however, it is still possible to create such an asset by executing the extrinsic directly through the API. An example of a script

that carries out this process is provided in Appendix F . We also created an example of a non-mintable asset with an initial supply of zero in this sora-staging test network transaction . Figure 23.1: A screenshot of the Sorascan explorer showing the successful registration of a non-mintable asset with an initial supply of zero. © 2021 HashEye Soramitsu Polkaswap Assessment | 54

Recommendations Short term, add a minimum initial supply requirement to the assets pallet’s register extrinsic. Additionally, consider adding a garbage collection method to handle assets that have no trading pairs and have not been used for a long time. © 2021 HashEye Soramitsu Polkaswap Assessment | 55

24. Off-chain worker can panic if the Ethereum API returns a null block_number Severity: Low Difficulty: High Type: Data Validation Finding ID: TOB-PSWAP-024 Target: eth-bridge Description The Ethereum bridge off-chain worker can panic if the Ethereum API to which it connects becomes malicious and returns a null block_number . A panic is possible because the off-chain worker uses the .expect() function to retrieve block numbers. This occurs in the following code paths: ● The parse_incoming_request function, shown in figure 24.1 ● The parse_cancel_incoming_request function ● The parse_old_incoming_request_method_call function In the parse_incoming_request code path, external data is fetched by the handle_offchain_request function (figure 24.2), through the load_tx_receipt function (figure 24.3), which shows that the data is from the Ethereum API. If this API is hacked or operates incorrectly, it may return a null block_number , causing the off-chain worker to panic. fn parse_incoming_request (tx_receipt: TransactionReceipt , incoming_pre_request: LoadIncomingTransactionRequest <T>,) → Result <IncomingRequest<T>, Error<T>> { // (...) let at_height = tx_receipt .block_number .expect("'block_number' is null only when the log/transaction is pending; qed") .as_u64(); Figure 24.1: The parse_incoming_request function (sora2-substrate/pallets/eth-bridge/src/offchain/mod.rs#L408-L426) fn handle_offchain_request (request: OffchainRequest <T>) → Result <(), Error<T>> { // (...) LoadIncomingRequest::Transaction(request) ⇒ { // (...) let tx = Self ::load_tx_receipt(tx_hash, network_id)?; let mut incoming_request = Self ::parse_incoming_request(tx, request)?; Figure 24.2: The handle_offchain_request function (sora2-substrate/pallets/eth-bridge/src/offchain/handle.rs#L244-L256) pub fn load_tx_receipt (hash: H256 , © 2021 HashEye Soramitsu Polkaswap Assessment | 56

network_id: T ::NetworkId,) → Result <TransactionReceipt, Error<T>> { let hash = types::H256(hash. 0); let tx_receipt = Self ::eth_json_rpc_request::<_, TransactionReceipt>("eth_getTransactionReceipt" , & vec! [hash], network_id,)?; let to = tx_receipt .to .map(|x| H160(x. 0)) .ok_or(Error::<T>::UnknownContractAddress)?; Self ::ensure_known_contract(to, network_id)?; Ok (tx_receipt) } Figure 24.3: The load_tx_receipt function fetches data from the external API (sora2-substrate/pallets/eth-bridge/src/offchain/http.rs#L259-L275). Recommendations Short term, change the off-chain worker so that it no longer expects block_number values returned by the Ethereum API to be non-null. Additionally, ensure that the system always checks the values returned by external APIs. Long term, add unit tests to the off-chain worker codebase to ensure that it correctly handles all invalid and edge-case values fetched from the Ethereum API. © 2021 HashEye Soramitsu Polkaswap Assessment | 57

A. Vulnerability Classifications Vulnerability Classes Class Description Access Controls Related to authorization of users and assessment of rights Auditing and Logging Related to auditing of actions or logging of problems Authentication Related to the identification of users Configuration Related to security configurations of servers, devices, or software Cryptography Related to protecting the privacy or integrity of data Data Exposure Related to unintended exposure of sensitive information Data Validation Related to improper reliance on the structure or values of data Denial of Service Related to causing a system failure Documentation Related to documentation errors, omissions, or inaccuracies Error Reporting Related to the reporting of error conditions in a secure fashion Patching Related to keeping software up to date Session Management Related to the identification of authenticated users Testing Related to test methodology or test coverage Timing Related to race conditions, locking, or the order of operations Undefi ned Behavior Related to undefi ned behavior triggered by the program Severity Categories Severity Description Informational The issue does not pose an immediate risk but is relevant to security best practices or Defense in Depth. Undetermined The extent of the risk was not determined during this engagement. Low The risk is relatively small or is not a risk the customer has indicated is important. © 2021 HashEye Soramitsu Polkaswap Assessment | 58

Medium Individual users’ information is at risk; exploitation could pose reputational, legal, or moderate financial risks to the client. High The issue could affect numerous users and have serious reputational, legal, or financial implications for the client. Diffi culty Levels Diffi culty Description Undetermined The difficulty of exploitation was not determined during this engagement. Low The flaw is commonly exploited; public tools for its exploitation exist or can be scripted. Medium An attacker must write an exploit or will need in-depth knowledge of a complex system. High An attacker must have privileged insider access to the system, may need to know extremely complex

technical details, or must discover other weaknesses to exploit this issue. © 2021 HashEye Soramitsu Polkaswap Assessment | 59

B. Code Maturity Classifications Code Maturity Classes Category Name Description Access Controls Related to the authentication and authorization of components Arithmetic Related to the proper use of mathematical operations and semantics Assembly Use Related to the use of inline assembly Centralization Related to the existence of a single point of failure Code Stability Related to the recent frequency of code updates Upgradeability Related to contract upgradeability Function Composition Related to separation of the logic into functions with clear purposes Front-Running Related to resilience against front-running Key Management Related to the existence of proper procedures for key generation, distribution, and access Monitoring Related to the use of events and monitoring procedures Specification Related to the expected codebase documentation Testing & Verification Related to the use of testing techniques (unit tests, fuzzing, symbolic execution, etc.) Rating Criteria Rating Description Strong The component was reviewed, and no concerns were found. Satisfactory The component had only minor issues. Moderate The component had some issues. Weak The component led to multiple issues; more issues might be present. © 2021 HashEye Soramitsu Polkaswap Assessment | 60

Missing The component was missing. Not Applicable The component is not applicable. Not Considered The component was not reviewed. Further Investigation Required The component requires further investigation. © 2021 HashEye Soramitsu Polkaswap Assessment | 61

C. Code Quality Recommendations • The expression returned from the list_enabled_sources_for_trading_pair function does not need to be wrapped in Ok(...) . • Remove the permissions pallet’s Mode enum . Its Mode::Forbid kind is not used anywhere in the code, so the whole enum can be removed; the pallet should then be refactored accordingly. • Remove the check_permission_maybe_with_parameters function from the assets pallet. It first checks for a scoped permission; if it does not find one, it checks for an unlimited-scope permission. This strategy of falling back to an unlimited-scope permission check is already implemented in the check_permission_with_scope function , which the assets pallet function calls. • Add the missing comments about Ethereum events to the parse_main_event function. Currently, only the ChangePeers(address,bool) hash is commented; the two other hashes lack appropriate comments indicating which function signatures they are constructed from . • In the off-chain worker , move the format string literals used for storage keys (shown in the screenshot below) to one place. That way, if two format strings have the same prefi x, it will be easier to detect the collision during development. Additionally, consider adding an automated check for this issue during testing or compilation. • The from field in the IncomingTransfer structure is not used and should be removed. • In the assets pallet's burn_from function, rename the “ to ” argument to “ from ,” which better refl ects its purpose. • Fix the StorageOverflow error description , which contains a comment from a Substrate node-template example : /// Errors should have helpful documentation associated with them. StorageOverflow, • Change the name of the claim_incentive function, which calculates weights, to claim_rewards . This will make the name of the function consistent with the name of the claim_rewards extrinsic , which will make the code more readable. © 2021 HashEye Soramitsu Polkaswap Assessment | 62

• The IncentiveDistributionFailed event is not used. There is a TODO comment regarding its future implementation. • Fix the into_balance function's TODO in the fixed_wrapper.rs . This function is currently used only in tests but should be fixed if it will be used in production code. • The case when an element is found in RequestsQueue map but missing in Requests map should be handled. • Delete the unnecessary code described in the handle_offchain_request function . © 2021 HashEye Soramitsu Polkaswap Assessment | 63

D. ERC20 Token Transfer Semantics This appendix lists the transfer and transferFrom return value semantics of the ERC20 tokens owned by the SORA Network’s Ethereum bridge contract. We used this list to identify tokens that could be leveraged to exploit the issues described in TOB-PSWAP-001 and TOB-PSWAP-009 as well as upgradeable tokens that should be monitored, as described in TOB-PSWAP-005 . This list is also maintained on a Google Sheets spreadsheet, , with a list of defl ationary/inflationary HashEye: ERC20 Token Semantics [PUBLIC] tokens taken from etherscan.io on the second sheet of the document. Token Ticker Symbol Contract Is it a proxy? Implementation transfer transferFrom Dai Stablecoin DAI Code No True or revert True or revert USD Coin USDC Proxy code Yes C ode True or revert True or revert Phala PHA Code No True or revert True or revert RARE.Unique RARE Code No True or revert True or revert ChainLink LINK Code No True or revert True or revert Uniswap UNI Code No True or revert True or revert FOTO FOTO Code No True or revert True or revert unification.com/x fund xFUND Code No True or revert True or revert FANS.UNIQUE FANS Code No True or revert True or revert Compound COMP Code No True or revert True or revert RLC RLC Code No True or revert True or revert Ocean OCEAN Code No True or revert True or revert Robonomics XRT Code No True or revert True or revert Graph GRT Code No True or revert True or revert Binance USD BUSD Proxy

code Yes code True or revert True or revert Republic REN Code Yo True or revert True or revert Wrapped BTC WBTC Code No True or revert True or revert Litentry LIT Code No True or revert True or revert Aave AAVE Code Yes code True or revert True or revert Matic MATIC Code No True or revert True or revert yearn.finance YFI Code No True or revert True or revert Akropolis AKRO Proxy code Yes code True or revert True or revert STAKE STAKE Code No True or revert True or revert © 2021 HashEye Soramitsu Polkaswap Assessment | 64

Reef.finance REEF Code No True or revert True or revert THORChain ETH.RUNE RUNE Code No True or revert True or revert TrueUSD TUSD Proxy code Yes code True or revert True or revert Wrapped UST UST Code No True or revert True or revert DIAToken DIA Code No True or revert True or revert renBTC renBTC Code No True or revert True or revert HUSD HUSD Code No True or revert True or revert Crypto.com Coin CRO Code No True or revert True or revert Cream CREAM Code No True or revert True or revert Paxos Standard PAX Proxy code Yes code True or revert True or revert StaFi FIS Code No True or revert True or revert SwissBorg CHSB Proxy code Yes code May return false May return false HoloToken HOT Code No True or revert True or revert Nexo NEXO Code No True or revert True or revert BAT BAT Code No May return false May return false Curve DAO Token CRV Code No True or revert True or revert Bitfinex LEO Token LEO Code No True or revert True or revert SushiToken SUSHI Code No True or revert True or revert FTT FTX Token Code No True or revert True or revert Maker MKR Code No True or revert True or revert Decentraland MANA Code No True or revert True or revert KyberNetwork KNC Code No True or revert True or revert UMA Voting Token v1 UMA Code No True or revert True or revert Telcoin TEL Code No True or revert True or revert Shabu Shabu KOBE Code No True or revert True or revert HuobiToken HT Code No May return false May return false OKB OKB Proxy code Yes code True or revert True or revert IDEX Token IDEX Code No True or revert True or revert Compound USD Coin cUSDC Code No May return false May return false ChronoBase TIK Proxy code Yes code True or revert True or revert Polkadex PDEX Code No True or revert True or revert © 2021 HashEye Soramitsu Polkaswap Assessment | 65

SingularityNET AGI Code no True or revert True or revert © 2021 HashEye Soramitsu Polkaswap Assessment | 66

E. Token Integration Checklist The following checklist provides recommendations for interactions with arbitrary tokens. Every unchecked item should be justified, and its associated risks, understood. An up-to-date version of the checklist can be found in crytic/building-secure-contracts . For convenience, all Slither utilities can be run directly on a token address, such as the following: slither-check-erc 0xdac17f958d2ee523a2206206994597c13d831ec7 TetherToken To follow this checklist, use the below output from Slither for the token: - slither-check-erc [target] [contractName] [optional: --erc ERC_NUMBER] - slither [target] --print human-summary - slither [target] --print contract-summary - slither-prop . --contract ContractName # requires configuration, and use of Echidna and Manticore General Security Considerations ☐ The contract has a security review. Avoid interacting with contracts that lack a security review. Check the length of the assessment (i.e., the level of eff ort), the reputation of the security firm, and the number and severity of the findings. ☐ You have contacted the developers. You may need to alert their team to an incident. Look for appropriate contacts on blockchain-security-contacts . ☐ They have a security mailing list for critical announcements. Their team should advise users (like you!) when critical issues are found or when upgrades occur. ERC Conformity Slither includes a utility, slither-check-erc , that reviews the conformance of a token to many related ERC standards. Use slither-check-erc to review the following: ☐ Transfer and transferFrom return a boolean. Several tokens do not return a boolean on these functions. As a result, their calls in the contract might fail. ☐ The name , decimals , and symbol functions are present if used. These functions are optional in the ERC20 standard and may not be present. If they are present, make sure that they cannot change over the lifetime of a token. ☐ Decimals returns a uint8 . Several tokens incorrectly return a uint256 . In such cases, ensure that the value returned is below 255. © 2021 HashEye Soramitsu Polkaswap Assessment | 67

☐ The token mitigates the known ERC20 race condition . The ERC20 standard has a known ERC20 race condition that must be mitigated to prevent attackers from stealing tokens. ☐ The token is not an ERC777 token and has no external function call in transfer or transferFrom . External calls in the transfer functions can lead to reentrancies. Slither includes a utility, slither-prop , that generates unit tests and security properties that can discover many common ERC flaws. Use slither-prop to review the following: ☐ The contract passes all unit tests and security properties from slither-prop . Run the generated unit tests and then check the properties with Echidna and Manticore . Finally, there are certain characteristics that are difficult to identify automatically. Conduct a manual review of the following conditions: ☐ Transfer and transferFrom should not take a fee. Defl ationary tokens can lead to unexpected behavior. ☐ Potential interest earned from the token is taken into account. Some tokens distribute interest to token holders. This interest may be trapped in the contract if not taken into account. Contract Composition ☐ The

contract avoids unnecessary complexity. The token should be a simple contract; a token with complex code requires a higher standard of review. Use Slither’s human-summary printer to identify complex code. ☐ The contract uses SafeMath . Contracts that do not use SafeMath require a higher standard of review. Inspect the contract by hand for SafeMath usage. ☐ The contract has only a few non-token-related functions. Non-token-related functions increase the likelihood of an issue in the contract. Use Slither’s contract-summary printer to broadly review the code used in the contract. ☐ The token has only one address. Tokens with multiple entry points for balance updates can break internal bookkeeping based on the address (e.g., balances[token_address][msg.sender] may not refl ect the actual balance). Owner Privileges ☐ The token is not upgradeable. Upgradeable contracts may change their rules over time. Use Slither’s human-summary printer to determine if the contract is upgradeable. ☐ The owner has limited minting capabilities. Malicious or compromised owners can abuse minting capabilities. Use Slither’s human-summary printer to review minting capabilities, and consider manually reviewing the code.

☐ The token is not pausable. Malicious or compromised owners can trap contracts relying on pausable tokens. Identify pausable code by hand. ☐ The owner cannot blacklist the contract. Malicious or compromised owners can trap contracts relying on tokens with a blacklist. Identify blacklisting features by hand. ☐ The team behind the token is known and can be held responsible for abuse. Contracts with anonymous development teams or teams that reside in legal shelters require a higher standard of review. Token Scarcity Reviews of token scarcity issues must be executed manually. Check for the following conditions: ☐ The supply is owned by more than a few users. If a few users own most of the tokens, they can influence operations based on the tokens’ repartition. ☐ The total supply is sufficient. Tokens with a low total supply can be easily manipulated. ☐ The tokens are located in more than a few exchanges. If all the tokens are in one exchange, a compromise of the exchange could compromise the contract relying on the token. ☐ Users understand the risks associated with a large amount of funds or flash loans. Contracts relying on the token balance must account for attackers with a large amount of funds or attacks executed through flash loans. ☐ The token does not allow flash minting. Flash minting can lead to substantial swings in the balance and the total supply, which necessitate strict and comprehensive overflow checks in the operation of the token.

F. Proof of Concept for Creating a Non-Mintable Asset with an Initial Supply of Zero This appendix provides a proof-of-concept Python script (figure F.1) that we used to create the non-mintable asset EMPT7 on the sora-staging test network, as shown in TOB-PSWAP-023 . To run the script, it is necessary to install the substrate-interface and scalecodec Python packages and to download the custom_types_polkaswap.json file. # Before running: # 1) pip install scalecodec==0.11.13 substrate-interface==0.13.8 # 2) wget https://raw.githubusercontent.com/sora-xor/PythonBot/master/custom_types.json # 3) fill in the 'KEYPAIR DETAILS' below # 4) Set proper url from substrateinterface import SubstrateInterface, Keypair from substrateinterface.exceptions import SubstrateRequestException from scalecodec.type_registry import load_type_registry_file substrate = SubstrateInterface(#url='wss://ws.mof.sora.org', url= 'wss://ws.stage.sora2.soramitsu.co.jp/' , ss58_format= 69 , type_registry_preset= 'default' , # taken from https://raw.githubusercontent.com/sora-xor/PythonBot/master/custom_types.json type_registry=load_type_registry_file('custom_types_polkaswap.json'),) # KEYPAIR DETAILS: This needs to be filled or better, changed to load keys from other source keypair = Keypair.create_from_mnemonic('<KEY MNEMONICS HERE>') call = substrate.compose_call(call_module= 'Assets' , call_function= 'register' , call_params={ 'symbol' : 'EMPT7' , # May require changing 'name' : 'emptytokenn' , # May require changing 'initial_supply' : 0 , 'is_mintable' : False }) try : extrinsic = substrate.create_signed_extrinsic(call=call, keypair=keypair) receipt = substrate.submit_extrinsic(extrinsic, wait_for_inclusion= False) print ("Extrinsic ' {} ' sent" .format(receipt.extrinsic_hash)) except Exception as e: print ("Failed to send: {} " .format(e)) Figure F.1: The script used to create a non-mintable asset with an initial supply of zero.

G. Fuzzing Polkaswap with test-fuzz We used test-fuzz , a framework for convenient AFL fuzzing, to fuzz certain functions in the Polkaswap codebase. To use test-fuzz , one must mark a function for fuzzing; the framework will then generate a fuzzing harness for it, along with additional code that will serialize the marked function's arguments when it is called in tests. To run test-fuzz , take the following steps: 1. Add test-fuzz and serde as dependencies. 2. Force-install (or reinstall) AFL via the cargo install afl --force command. (Issue rust-fuzz/afl.rs#183 may necessitate reinstallation.) 3. Add the #[test_fuzz] macro to the function to be fuzzed. If the function is in a trait, add #[test_fuzz::test_fuzz_impl] before the " impl " block; it may also be necessary to perform argument concretization for the test_fuzz attribute. Additional information is provided in the example patch below and the test-fuzz documentation. 4. Execute cargo test so that the function to be fuzzed will be executed and its input will be serialized. The test-fuzz macro will work only if the target function is launched by tests, as it needs to save an initial corpus of the function

arguments. 5. Run the fuzzing with the following command: cargo test-fuzz --target <function> 6. If the code cannot be compiled in its entirety (because of a dependency issue in another crate, for example), it may be necessary to run the tool against a given package by using the --package <package> argument. The AFL fuzzer will run indefi nitely. We recommend running it for as long as possible; however, if it runs for a long time without finding new paths, it may make sense to stop the fuzzing, investigate its code coverage, and generate inputs that reach the previously unreached paths. Figure G.3 includes a patch that adds test-fuzz to the project as well as a fuzzing harness for the parse_deposit_event function. This patch can be applied to commit 9d72e6c of the project through the git apply <file> command. Then to run the fuzzing, invoke following commands: - cargo test -p eth-bridge - cargo test-fuzz -p eth-bridge --target=parse_deposit_event Figure G.1 shows the AFL window that should appear to display the fuzzing progress. © 2021 HashEye Soramitsu Polkaswap Assessment | 71

Figure G.1: The AFL window displaying the progress of the fuzzing. If running test-fuzz results in an error of " failed to download `parity-db v0.2.3` " (as shown in figure G.2), use the cargo update -p parity-db command to update the dependency. Error: `cargo metadata` exited with an error: error: failed to download `parity-db v0.2.3` Caused by: unable to get packages from source Caused by: failed to parse manifest at `/Users/dc/.cargo/registry/src/github.com-1ecc6299db9ec823/parity-db-0.2.3/Cargo.toml` Caused by: failed to parse the version requirement `0.11 ` for dependency `parking_lot` Caused by: expected comma after minor version number, found '\t' Figure G.2: A potential test-fuzz dependency-related error, which can be resolved by updating the parity-db package. © 2021 HashEye Soramitsu Polkaswap Assessment | 72

diff --git a/src/sora2-substrate/pallets/eth-bridge/Cargo.toml b/src/sora2-substrate/pallets/eth-bridge/Cargo.toml index 8a18512..d9dcbfc 100644 --- a/src/sora2-substrate/pallets/eth-bridge/Cargo.toml +++ b/src/sora2-substrate/pallets/eth-bridge/Cargo.toml @@ -19,7 +19,7 @@ rlp = { version = "0.4.6", default-features = false } rustc-hex = { version = "2.1.0", default-features = false } serde = { version = "1.0", features = ["alloc", "derive"], default-features = false } serde_json = { version = "1.0", default-features = false, features = ["alloc"] } -sp-core = { version = "3", default-features = false } +# sp-core = { version = "3", default-features = false } sp-std = { version = "3", default-features = false } tiny-keccak = { version = "2", default-features = false } pallet-scheduler = { version = "3", default-features = false } @@ -32,11 +32,13 @@ ethabi-derive = { git = "https://github.com/vmarkushin/ethabi.git", branch = "no ethereum-types = { git = "https://github.com/vmarkushin/parity-common.git", branch = "no-std", package = "ethereum-types", default-features = false, features = ['serialize', 'codec'] } jsonrpc-core = { git = "https://github.com/vmarkushin/jsonrpc.git", branch = "no-std", package = "jsonrpc-core", default-features = false } -assets = { path = "../assets", default-features = false } -common = { path = "../../common", default-features = false } -permissions = { path = "../permissions", default-features = false } +# assets = { path = "../assets", default-features = false } +# common = { path = "../../common", default-features = false } +# permissions = { path = "../permissions", default-features = false } -[dev-dependencies] +test-fuzz = { version = "0.1.0-alpha.24", default-features = false, features = ["serde_cbor"] } + +# [dev-dependencies] async-std = { version = "1.5", features = ["attributes", "unstable"] } currencies = { version = "0.4", package = "orml-currencies", default-features = false } env_logger = "0.8.1" diff --git a/src/sora2-substrate/pallets/eth-bridge/src/lib.rs b/src/sora2-substrate/pallets/eth-bridge/src/lib.rs index df41c69..50bf953 100644 --- a/src/sora2-substrate/pallets/eth-bridge/src/lib.rs +++ b/src/sora2-substrate/pallets/eth-bridge/src/lib.rs @@ -142,7 +142,7 @@ mod migrations; pub mod offchain; pub mod requests; mod rpc; -#[cfg(test)] +// #[cfg(test)] mod tests; pub mod types; mod util; @@ -364,7 +364,7 @@ pub mod pallet { type GetEthNetworkId: Get<Self::NetworkId>; /// Weight information for extrinsics in this pallet. type WeightInfo: WeightInfo; - #[cfg(test)] + // #[cfg(test)] type Mock: tests::mock::Mock; #[pallet::constant] diff --git a/src/sora2-substrate/pallets/eth-bridge/src/offchain/mod.rs b/src/sora2-substrate/pallets/eth-bridge/src/offchain/mod.rs index 7324af5..f058ced 100644 --- a/src/sora2-substrate/pallets/eth-bridge/src/offchain/mod.rs +++ b/src/sora2-substrate/pallets/eth-bridge/src/offchain/mod.rs @@ -92,7 +92,9 @@ pub mod crypto { } } +#[test_fuzz::test_fuzz_impl] impl<T: Config> Pallet<T> { + #[test_fuzz::test_fuzz(concretize_impl = "crate::tests::mock::Runtime")] fn parse_deposit_event(log: &Log,) -> Result<DepositEvent<Address, T::AccountId, Balance>, Error<T>> { diff --git a/src/sora2-substrate/pallets/eth-bridge/src/tests/mock.rs b/src/sora2-substrate/pallets/eth-bridge/src/tests/mock.rs index 8896dce..9b5e96e 100644 --- a/src/sora2-substrate/pallets/eth-bridge/src/tests/mock.rs +++ b/src/sora2-substrate/pallets/eth-bridge/src/tests/mock.rs © 2021 HashEye Soramitsu Polkaswap Assessment | 73

@@ -435,6 +435,24 @@ construct_runtime!(}); +impl<'de> serde::Deserialize<'de> for Runtime { + fn deserialize<D>(deserializer: D) -> Result<Self, D::Error> + where + D: serde::Deserializer<'de>, + { + <()>::deserialize(deserializer).map(|_| Runtime) + } +} + +impl serde::Serialize for Runtime { + fn serialize<S>(&self, serializer: S) -> Result<S::Ok, S::Error> + where + S: serde::Serializer,

+ { + ().serialize(serializer) + } +} + pub type SubstrateAccountId = <<Signature as Verify>::Signer as IdentifyAccount>::AccountId; pub trait Mock { diff --git a/src/sora2-substrate/pallets/eth-bridge/src/types/substrate.rs b/src/sora2-substrate/pallets/eth-bridge/src/types/substrate.rs index b65f380..430a719 100644 --- a/src/sora2-substrate/pallets/eth-bridge/src/types/substrate.rs +++ b/src/sora2-substrate/pallets/eth-bridge/src/types/substrate.rs @@ -32,7 +32,7 @@ use crate::types::{H256, U64}; use alloc::string::String; use codec::{Decode, Encode}; use serde::Deserialize; -#[cfg(test)] +// #[cfg(test)] use serde::Serialize; use sp_std::vec::Vec; @@ -41,7 +41,7 @@ use sp_std::vec::Vec; #[derive(PartialEq, Eq, Clone, Default, Encode, Decode)] pub struct OpaqueExtrinsic(Vec<u8>); -#[cfg(test)] +// #[cfg(test)] impl ::serde::Serialize for OpaqueExtrinsic { fn serialize<S>(&self, seq: S) -> Result<S::Ok, S::Error> where @@ -65,8 +65,7 @@ impl<'a> serde::Deserialize<'a> for OpaqueExtrinsic { } } -#[derive(Deserialize)] -#[cfg_attr(test, derive(Serialize))] +#[derive(Deserialize, Serialize)] #[serde(rename_all = "camelCase")] pub struct SubstrateHeaderLimited { /// The parent hash. @@ -85,8 +84,7 @@ pub struct SubstrateHeaderLimited { pub digest: (), } -#[derive(Deserialize)] -#[cfg_attr(test, derive(Serialize))] +#[derive(Deserialize, Serialize)] #[serde(rename_all = "camelCase")] pub struct SubstrateBlockLimited { /// The block header. @@ -95,8 +93,7 @@ pub struct SubstrateBlockLimited { pub extrinsics: Vec<OpaqueExtrinsic>, } -#[derive(Deserialize)] -#[cfg_attr(test, derive(Serialize))] +#[derive(Deserialize, Serialize)] #[serde(rename_all = "camelCase")] pub struct SubstrateSignedBlockLimited {

/// Full block. diff --git a/src/sora2-substrate/pallets/iroha-migration/Cargo.toml b/src/sora2-substrate/pallets/iroha-migration/Cargo.toml index fc0305b..c2503e1 100644 --- a/src/sora2-substrate/pallets/iroha-migration/Cargo.toml +++ b/src/sora2-substrate/pallets/iroha-migration/Cargo.toml @@ -49,7 +49,7 @@ default = ["std"] std = ["codec/std", - "frame-benchmarking?/std", + "frame-benchmarking/std", "frame-support/std", "frame-system/std", "pallet-multisig/std", diff --git a/src/sora2-substrate/runtime/Cargo.toml b/src/sora2-substrate/runtime/Cargo.toml index 9c9553b..770bf57 100644 --- a/src/sora2-substrate/runtime/Cargo.toml +++ b/src/sora2-substrate/runtime/Cargo.toml @@ -160,7 +160,7 @@ std = ['eth-bridge/std', 'eth-bridge-runtime-api/std', 'farming/std', - 'faucet?/std', + 'faucet/std', 'iroha-migration/std', 'iroha-migration-runtime-api/std', 'liquidity-proxy/std', @@ -192,7 +192,7 @@ runtime-benchmarks = ["dex-api-benchmarking", "eth-bridge/runtime-benchmarks", "farming/runtime-benchmarks", - "faucet?/runtime-benchmarks", + "faucet/runtime-benchmarks", "frame-benchmarking", "frame-support/runtime-benchmarks", "frame-system-benchmarking", Figure G.3: The patch is used to run test-fuzz over the Polkaswap codebase. The changes made to cargo.lock by the compiler are excluded from this listing; however, the file should regenerate itself after Cargo.toml has been changed.