Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Pentest-Report ChromaWay ETH Bridge 05.-06.2023

Cure53, Dr.-Ing. M. Heiderich, Dipl.-Ing. G. Szivos, M. Haunschmid

Index

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Introduction

“We are the creators of relational blockchain, a class of blockchain platforms that combine the power and flexibility of mature relational database systems with the secure collaboration and disruptive potential of blockchain”

From https://chromaway.com/

This document pertains to a penetration test and source code audit against the ChromaWay Postchain EIF project and codebase, as requested by ChromaWay AB in May 2023 and performed by Cure53 throughout CW22 and CW23. The assessment actions were fulfilled during an allocation of fifteen work days and were structured into a single work package (WP), as follows:

- **WP1**: White-box penetration testing & code auditing against the ChromaWay Postchain EIF project & code

Cure53 was granted access to sources, meticulous assisting documentation, test-user credentials, and supplementary access entities in adherence with the preselected methodology, white-box. A team comprising three senior testers was assembled to complete all phases of the assignment - including preparation, execution, and finalization - based on their relevant know-how and expertise with similar frameworks.

The active assignment was preceded by a number of preliminary actions, which were completed in the week prior (CW21 May 2023) to enable a productive working environment.

During the test, communication was facilitated through a dedicated and shared Zulip chat. This chat was open to all personnel involved in the test from both the ChromaWay and Cure53 teams. The collaboration process was conducted amicably and fluidly on the whole, with few cross-team questions required. The scope received optimal and transparent preparation, thus no noteworthy delays or hindrances were encountered at all. The test team relayed abundant status updates pertinent to the test and notable findings, though live reporting was not specifically requested for this audit.

Cure53’s approaches yielded a sum total of eight findings following widespread coverage over the key scope items. Two of the findings were deemed to be security vulnerabilities, and the remaining six represented common weaknesses exhibiting minor exploitation likelihood.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

In essence, this is undoubtedly a small volume of findings - particularly considering that this is the first external audit between the two organizations - which instills confidence regarding the security resilience integrated for the ChromaWay ETH Bridge. Moreover, this positive viewpoint is corroborated by the fact that none of the tickets exceeded a severity marker of *Medium*.

To summarize, based on the outcomes encountered following the finalization of this project, the ChromaWay team deserves every plaudit for their admirable implementation, which imbues evident protection and hardening for the ETH Bridge. This foundation is a robust starting point upon which an exemplary security standard can be achieved, should the developer team heed the guidance offered throughout this report.

In terms of the report structure moving forward, a selection of core segments are outlined forthwith. Firstly, the scope, test setup, and available materials are enumerated in the ensuing chapter’s bullet points.

This is followed by a proportion entitled *Test Methodology*, which serves to clarify to the client the depth of coverage and variety of risk estimation stratagem conducted, in spite of the lack of major impact findings. Afterward, the report provides all findings in descending and chronological order of detection, starting with the *Identified Vulnerabilities* and ending with the *Miscellaneous Issues*. An expert technical synopsis, Proof-of-Concept (PoC) or steps to reproduce, and suggested fix proposals are outlined in each ticket.

To finalize proceedings, Cure53 elaborates on the general impressions garnered throughout this engagement, with complementing viewpoints concerning the perceived security posture of the ChromaWay Postchain EIF project and codebase scope under scrutiny.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Scope

- **Penetration tests & code audits against the ChromaWay Postchain EIF project**
  - **WP1**: White-box penetration testing & code auditing against the ChromaWay Postchain EIF project & code
    - **Sources:**
      - https://gitlab.com/chromaway/postchain-eif/-/tree/dev
    - **Documentation:**
      - White paper on Chromia***:***
        - https://chromia.com/whitepaper/
      - General documentation:
        - https://docs.chromia.com/
  - **Test-user credentials:**
    - The Cure53 team created the following (EVM) testing accounts:
      - *0x1616BFA1Ba4a5628545a2f11Bc95924712726231*
      - *0x2732b052E8BadcaD0D9Ab46C4f55024aB823d698*
      - *0x786ACCAdf853CC23A81c3D6a38a4476FAC46C6d5*
      - *0x7141CEfbAf13272da7395cfcEE35D8EF4b19cE41*
      - *0x41d51824eD56bBA0319127254df82E34343d3E10*
      - *0x12D0A10c1eE1beBA5cFB96d6d859Ae95bc6aa824*
      - *0xB753C04dFe028512d45bD4F89f313C09002d85A7*
    - The ChromaWay development team supplied the first three of the previously mentioned wallets with 1000 ALICE tokens each
  - **Test-supporting material was shared with Cure53**
  - **All relevant sources were shared with Cure53**

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Test Methodology

This report's *Test Methodology* segment documents the myriad approaches applied during the engagement, with supporting information concerning Cure53's thought processes and degree of coverage, in lieu of the detection of major risk vectors. This section is divided into two proportions based on the alternate codebases reviewed, specifically *postchain-eif-contracts* and *postchain-eif-core*. The third codebase, *postchain-eif-ui*, was deemed out-of-scope by the client and thus omitted from these passages, though was nonetheless leveraged to gain an exhaustive understanding of the application and connected functionality.

WP1 - Part 1: *postchain-eif-contracts*

This section offers an overview of the tests conducted against the smart contracts utilized in ChromaWay's ETH Bridge project. The smart contracts written in Solidity necessitate specific tooling and methodologies in comparison with other applications.

The application's functionality was grouped into different libraries and contracts, adopting industry-standard libraries from OpenZeppelin. Static analysis of the Solidity codebase pinpointed a number of pertinent focus areas, which led to the discovery of an issue related to an absent return value check within the transfer methods, as discussed in ticket CRW-01-005.

Since smart contracts require an alternative approach to security than traditional applications, industry-standard checklists such as the OWASP WSTG/MSTG were not applicable in this context. Cure53 instead referred to the Smart Contract Security Verification Standard (SCSVS)[1] for guidelines informing the examination of the supplied Solidity code. A few auditing areas highlighted in these guidelines were also considered out-of-scope, though the vast majority could be applied directly whilst inspecting the contracts.

- **Architecture**: Since the smart contracts do not exist in isolation, this review aspect focussed on the contract deployment methods; established contract event logging procedures; and inconsistencies between contracts and their ensuing behaviors in the event of exploitation. Positively, Cure53 was unable to identify any connected issues.

- **Access controls**: The consultant team strove to ascertain whether the contracts included any form of role-based access controls; whether *onlyOwner* was applied correctly; and if any incorporated functionality could facilitate privilege escalation.

[1] https://github.com/securing/SCSVS

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Here, the observation was made that the *fund* and *fundNFT* methods may incur user financial loss, as detailed in ticket CRW-01-002.

- **Blockchain data**: Due to the fact that smart contracts do not provide a built-in mechanism to store secret data securely, assessment initiatives were conducted to verify whether sensitive data was saved on-chain and could be susceptible to disclosure or unforeseen exploitation. These endeavors proved unfruitful in detecting any associated erroneous behaviors.

- **Communications**: The usage of libraries and other, possibly untrusted contracts poses inherent security risks for the application. Here, potential dependencies - as well as interfaces to libraries and other contracts - were subjected to an extensive review process. The ensuing dependency analyses highlighted that some were outdated and library-associated security advisories were publicly known. Notably, Cure53 positively acknowledged that none of the features within these advisories deemed vulnerable were deployed by the application. For supplementary guidance on this finding, please refer to ticket CRW-01-001.

- **Arithmetic**: Calculations in smart contracts evoke certain security implications. However, since Solidity 0.8.x was utilized in the contracts, the likelihood for integer over- or under-flows was negated under these circumstances. Nonetheless, Cure53 deemed it apt to perform correlatory checks for incorrect comparisons and calculations, though these similarly yielded a lack of results.

- **Malicious input handling**: Generally speaking, user-supplied input should never be trusted and the application should enforce rigid input validation accordingly. Subsequently, the inputs supplied to public contract functions and connected validation logic were rigorously probed, though no notable vulnerabilities were located in this respect.

- **Gas usage & limitations**: Cure53 focussed on validating the possibility to use alternate contract functionality to exhaust gas or introduce a DoS scenario, though these did not evoke any security consequences.

- **Business logic**: The auditors implemented test methods to verify whether the contract logic applied smart contract anti-patterns, such as amending the execution flow based on contract balance or block data (e.g. hash or timestamp). Similarly, no connected areas of concern were detected.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

- **DoS**: Due to the immutability of deployed contracts, the potential for funding locks or general DoS attacks was estimated, though likewise this aspect offered negligible risk.

- **Token**: The implementation of token functionality should adhere to industry-standard patterns and libraries. With this in mind, a plethora of strategies were applied to ascertain whether the application contracts exhibited comprehensive best-practice compliance, or indeed whether any prevalent security flaws were persisted. These efforts concluded with no noteworthy findings to report.

- **Code clarity**: Whilst unclear code in general does not imbue immediate security risk, a myriad array of unforeseen negative implications may be evoked. As such, function and variable naming was checked to assess the risk of possible user or developer confusion, which may otherwise introduce security vulnerabilities. Albeit, no faults were observed in this regard.

In addition to checklist-based testing, manual testing and reviews were employed. This involved interacting with the contracts via the command line and modifying the existing test cases in the codebase. Manual testing primarily focused on examining the interaction amongst various application components, since this dynamic interplay presents challenges for static or automated testing and introduces heightened security concerns.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

WP1 - Part 2: *postchain-eif-core*

The *postchain-eif-core* codebase included both Kotlin and Rell code, and contained the Postchain side of the Bridge. The project's Kotlin component initially underwent high-level static analysis, which returned no areas of interest. As a result, manual deep-dive code reviews and testing were conducted in an attempt to yield potential compromise vectors.

Besides parsing GVT plus configuration and event processing, the framework's Kotlin characteristics exposed minimal attack surface on the whole, particularly considering that it was inaccessible via the supplied alpha access to the My Neighbour Alice application.

The second *postchain-eif-core* proportion was constructed in Rell, a custom-developed programming language that offers an SQL-esque syntax to access data on the private blockchain. To obtain a sweeping understanding of the queries and operations possible in the application, as well as learn the Rell language itself, the Cure53 team carefully studied the supplied documentation. Subsequently, the Rell code was manually explored, particularly in relation to the interface between the EVM blockchain and Rell module. Here, ticket CRW-01-007 pertains to a vulnerability that may be exploited by a malicious actor in order to divert deposited funds to a third-party account.

Finally, a host of sensitive functionality entities - such as authentication and authorization - were systematically appraised. Notably, the client promptly supplied any absent dependencies and libraries upon request in this respect.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Identified Vulnerabilities

The following section lists all vulnerabilities and implementation issues identified during the testing period. Notably, findings are cited in chronological order rather than by degree of impact, with the severity rank offered in brackets following the title heading for each vulnerability. Furthermore, all tickets are given a unique identifier (e.g., *CRW-01-001*) to facilitate any future follow-up correspondence.

CRW-01-005 WP1: Unchecked transfer method return values *(Medium)*

During the source code audit, the discovery was made that the *transfer* and *transferFrom* methods in the ERC20 token standard are designed to return a boolean value indicating whether the operation had succeeded or otherwise. Although most token implementations revert on failed transfers, some only return *false*.

Owing to the fact that the TokenBridge does not check for this return value in several instances, an attacker could perform actions with unintended side effects. Most notably, they could call the *TokenBridge.deposit* method to force the implemented *token.transferFrom* method to return *false*, which would facilitate receiving the balance inside the TokenBridge contract for free. In that case, the *DepositedERC20* event is also emitted, meaning that they would receive the respective amounts of tokens on the Postchain as a result.

Nonetheless, due to the fact that every token requires administrator approval, this ticket's severity marker was downgraded to *Medium*.

Affected file:
postchain-eif-contracts/contracts/TokenBridge.sol

Affected code:

```
function deposit(IERC20 token, uint256 amount, bytes32 ft3_account_id)
isAllowToken(token) public returns (bool) {
    (string memory name, string memory symbol, uint8 decimals) =
_getTokenInfo(token);
    token.transferFrom(msg.sender, address(this), amount);
    _balances[token] += amount;
    emit DepositedERC20(msg.sender, token, ft3_account_id, networkId, amount,
name, symbol, decimals);
    return true;}
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Additional instances of this antipattern were identified that lack processes to check the return value, as enumerated below:

- *TokenBridge.fund* #148
- *TokenBridge.withdraw* #156
- *TokenBridge.withdrawBySnapshot* #274
- *TokenBridge.emergencyWithdraw* #310

To mitigate this issue, Cure53 advises strictly validating the return values of all *transfer* and *transferFrom* method calls. Another effective solution would be to utilize the *SafeERC20*[2] interface, which provides wrappers around ERC20 operations that throw upon failure (i.e. when the token contract returns *false*), similarly to the implementation already established for NFTBridge.

CRW-01-007 WP1: Potential fund loss via forged transactions *(Medium)*

The core functionality of the tested smart contract was to provide a bridge for transferring funds between a conventional blockchain and the Postchain blockchain. This was achieved via a smart contract, which acted as the interface for users. This smart contract offered several functions, including the *deposit* method to initially transfer funds from the conventional blockchain to Postchain, as well as the *withdrawToPostchain* method, which permitted users to return their funds to Postchain in the event withdrawing to an EVM chain is infeasible.

Both the *deposit* and *withdrawToPostchain* methods emit an *DepositedERC20* event upon successful completion. This event contains information such as the caller's public key, the token amount, and the FT3 account ID wherein the transferred funds will be deposited. However, at the time of testing, the bridge's Postchain side fails to verify that the provided FT3 account belongs to the actual function caller. This behavior could be exploited by an attacker in combination with another vulnerability (such as XSS) by altering the user's FT3 account ID to an attacker-controlled entity.

Affected file:
postchain-eif-core/src/rell/eif/module.rell

Affected code:
```
operation __evm_block(network_id: integer, evm_block_height: integer,
evm_block_hash: byte_array, events: list<event_data>) {

[...]
```

[2] https://docs.openzeppelin.com/contracts/4.x/api/token/erc20#SafeERC20

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
val beneficiary =
to_zero_padded_hex(byte_array.from_gtv(event.indexed_values[0]));
val token_address =
to_zero_padded_hex(byte_array.from_gtv(event.indexed_values[1]));
val ft3_account_id =
byte_array.from_gtv(event.indexed_values[2]);log("ft3_account_id: " +
ft3_account_id.to_hex());

val ft3_account = acc.account @? { .id == ft3_account_id };

[...]

ft3_balance.amount += amount;
val ft3_beneficiary = to_zero_padded_hex(evm.evm_account @ { .account ==
ft3_account } ( .address ));
val account_link =
get_or_create_evm_account_link(token_mapping.token.network_id, ft3_beneficiary);
val state = build_account_state(network_id, ft3_beneficiary, ft3_account);
```

To mitigate this issue, Cure53 suggests verifying that the provided FT3 account ID belongs to the caller, thereby preventing malicious transactions that incur loss of funds. It should be noted that this vulnerability can only be exploited in combination with flaws, which would allow an attacker to execute code, inject variables in the victim's browser, or otherwise convince the user to sign a malicious transaction.

One can pertinently note that this issue has already been discussed with the client during the course of this security assessment. The development team verified the proposal to adopt the EVM address rather than FT3 account ID during the fund depositing process in the future.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Miscellaneous Issues

This section covers any and all noteworthy findings that did not incur an exploit but may assist an attacker in successfully achieving malicious objectives in the future. Most of these results are vulnerable code snippets that did not provide an easy method by which to be called. Conclusively, whilst a vulnerability is present, an exploit may not always be possible.

CRW-01-001 WP1: Use of outdated libraries as dependencies *(Info)*

The *postchain-eif-contracts* module depends on contract implementations provided by OpenZeppelin. Whilst the incorporation of battle-tested libraries is generally considered a sound practice and none of the contracts was directly affected by any of these faults, one can nonetheless recommend ensuring that the dependencies are up-to-date. Modern package managers offer functionality to check dependencies for vulnerabilities and even resolve them automatically.

Affected dependencies:

- ***"@openzeppelin/contracts": "^4.4.2",***
- ***"@openzeppelin/contracts-upgradeable": "^4.5.2"***

The versions in question were released over a year ago; several security advisories for the stated versions have since been published[3]. Please note that all version information (used and to-be used) are based on information gathered at the time of the audit.

To mitigate this issue, Cure53 advises ensuring that all leveraged software are updated to the most recent available versions, since older versions often contain known (or unknown) vulnerabilities that may be susceptible to attacker exploitation.

CRW-01-002 WP1: Absent ACL may incur loss of funds *(Low)*

Cure53 noted that both the *fund* and *fundNFT* of *TokenBridge.sol* and *NFTBridge.sol* respectively can be called by any user. However, the emitted events offer negligible effect on the Bridge's Postchain side, meaning that a user will lose access to their funds when using the *fund* or *fundNFT* methods.

Affected files:

- *postchain-eif-contracts/contracts/TokenBridge.sol*
- *postchain-eif-contracts/contracts/NFTBridge.sol*

[3] https://github.com/OpenZeppelin/openzeppelin-contracts/security/advisories

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Affected code:

```
function fund(IERC20 token, uint256 amount) isAllowToken(token) public returns
(bool) {
        token.transferFrom(msg.sender, address(this), amount);
        _balances[token] += amount;
        emit FundedERC20(msg.sender, token, amount);
        return true;
}
```

This area of concern was also discussed with the client during the active assignment, which verified that bridge contract funding will be exclusively performed by administrators. As such, Cure53 advises utilizing the *onlyOwner* modifier in the contracts for the *fund* and *fundNFT* functions.

CRW-01-003 WP1: Disclosure of sensitive information in source code *(Info)*

Rather than a comprehensive production-ready application, the provided *postchain-eif-ui* UI project alternatively represented a PoC or demo application that served to demonstrate potential interactions with the implemented ecosystem. This was consequently evaluated by the testing team to increase awareness concerning Postchain blockchain interaction methods.

With this in mind, Cure53's assessment procedures verified the disclosure of sensitive information in the application within the *Bridge.tsx* file. As the following snippet demonstrates, the private keys of two accounts have been hardcoded.

Affected file:
postchain-eif-ui/src/Bridge.tsx

Affected code:

```
[...]
const userPUB = Buffer.from(
    "038f888dec563b5bc253e87abc90afd26c3287021d10236ea19d248043dc39e0b8",
    "hex"
  );
  const userPRIV = Buffer.from(
    "71b[REDACTED]825",
    "hex"
  );
  const adminPUB = Buffer.from(
    "02a829e1d7fffbd856a04b53ec7d478d8896803b571c7700ec464d6a9d4f0e3bbd",
    "hex"
  );
  const adminPRIV = Buffer.from(
    "2e8[REDACTED]aeb9",
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
    "hex"
  );
[...]
```

Even though this application was only developed to demonstrate the capabilities of the Postchain project, one can still recommend complying with industry standards regarding the protection of potentially sensitive information, such as passwords or private keys. Best practice integration stipulates hardcoding sensitive information in the source code or preventing leakage of said information via version control systems, such as git. This can be achieved by storing sensitive information in an external source, such as an untracked configuration file or an environment variable on the hosting system.

CRW-01-004 WP1: Absent logging functionality *(Info)*

Code examinations and ensuing client verification confirmed a lack of logging capability to monitor suspicious behavior related to the Bridge's smart contracts. Despite the fact that the ChromaWay team is planning to incorporate a mechanism of this nature in the future, Cure53 deemed it apt to document this finding for completist purposes.

To mitigate this issue, one can recommend deploying a logging toolset to ingest all events emitted by the utilized smart contracts, which would implement alerts should any suspicious activities emerge. This approach would guarantee that any potential exploitation of issues such as CRW-01-005 can be detected as soon as possible.

CRW-01-006 WP1: Bridge functionality unpausable *(Info)*

Testing verified the presence of functionality to mitigate or halt contract exploitation. However, this strategy was deemed insufficient in general and a more comprehensive approach should be employed.

After the bridge contract has been deployed for 90 days, it is possible to drain all contract funds as an administrator. This could effectively negate some exploitation scenarios against the bridge, but will also deny access to user funds. Any successful exploitation attempts before the hardcoded 90 day time frame cannot therefore be halted by the client. In addition, the applied emergency withdrawal feature imbues extraneous complexity to the functionality restoration process.

To mitigate this issue, Cure53 advises implementing the OpenZeppelin contract entitled *Pausable*[4]. This contract offers the *whenPaused* and *whenNotPaused* modifiers, which can be leveraged to render certain functionality unavailable when the contract is paused. As a result, potential exploitation scenarios such as that described in ticket CRW-01-005 can be effectively neutralized.

[4] https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/security/Pausable.sol

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

CRW-01-008 WP1: Double withdrawal possible in *mass-exit* scenario *(Low)*

The *mass-exit* feature provides the possibility to withdraw funds based on a recent snapshot, whereby the state root is periodically synced to the EVM chain. Upon deciding to trigger a *mass-exit*, the admin defines the block height - and thus the state - from which withdrawals by snapshot can be performed.

From this moment onward, new withdrawal requests can no longer be initiated. However, existing open requests can still be withdrawn. This behavior is essential from a security perspective, since the balances from open withdrawal requests are not included in the snapshot and therefore must be separately handled.

In the potential scenario whereby an attacker initiates a withdrawal shortly before a *mass-exit* is triggered, the withdrawal request may not be captured in the selected snapshot. As a result, the balance associated with the withdrawal request would be included in the snapshot whilst remaining available for regular withdrawal. This effectively allows the attacker to double their balance by withdrawing both via a snapshot and their existing withdrawal.

To mitigate this issue, the ChromaWay team should sufficiently estimate the likelihood of financial loss in this scenario and decide whether to accept the associated risks or remove the *mass-exit* functionality entirely.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Conclusions

The impressions gained during this assignment will now be discussed in depth. The findings identified during Cure53's CW22 and CW23 testing against the Postchain ecosystem - including the Bridge smart contracts and My Neighbor Alice application - attest to the integration of performant security effectiveness by the ChromaWay team.

Notably, this exercise marked the inaugural collaboration between ChromaWay and Cure53. Specific emphasis was placed on auditing and analyzing selected aspects of the Postchain-EVM Bridge, honing in on fund deposit and withdrawal functionality. These mechanisms permit system users to deposit funds (various supported tokens such as ALICE) from a conventional blockchain - such as BNB to a separately operating Postchain blockchain - or withdraw deposited funds from Postchain to BNB.

The testing team conducted source code evaluation procedures for each in-scope components, including (but not limited to) the smart contracts, the Bridge implementation on the Postchain side, and the core node implementation. Dynamic testing was also applied in an attempt to uncover any potential attack vectors on the provided test blockchains.

Despite the fact that the provided source code was reasonably well organized, the testing team required a lengthier time frame than typically expected to gain an overarching understanding of its premise. This was due to the inherent complexity of utilizing different languages for alternate software components, which were written in JavaScript, Kotlin, Rell, and Solidity to varying degrees. Nevertheless, the provided code enabled adequate comprehension of the software's internal behavior during the dynamic analysis.

The auditor's primary area of focus constituted the implemented smart contracts, which were deep-dive investigated with regards to logical flaws and typical related attack scenarios, as explained within the dedicated Test Methodology chapter.

Here, one must underline the client's credit-worthy communication and swift assistance throughout the course of the review. Cure53 would like to express appreciation to the ChromaWay developer team for their support and clarification of complex or otherwise opaque characteristics. For this purpose, a dedicated Zulip channel was established, which was absolutely integral toward efficient query resolution and feedback provision in the event of malfunctioning processes.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Concerning the findings encountered, two tangible vulnerabilities and six miscellaneous issues were documented. The most risk-susceptible of those are outlined in tickets CRW-01-005 and CRW-01-007. If an attacker were to leverage these weaknesses, they may be able to increase their balance on the Postchain side of the bridge without transferring any funds to the EVM side, as well as divert the fund deposits to a third-party FT3 account. Both scenarios may facilitate financial loss on the EVM side and as such should be addressed at the earliest possible convenience.

The system's security posture would certainly benefit from enhancements, for which the area of additional security controls was pinpointed as particularly deserving and listed within *Miscellaneous Issues* (six in total). Despite this recommendation, the majority of issues were merely assigned an *Info* severity rating. To extrapolate some of the more pertinent findings, Cure53 noted that smart contract users could potentially lose funds in the event they call the *fund* rather than *deposit* function, as discussed in ticket CRW-01-002.

Likewise, the source code review revealed that outdated libraries were leveraged within the smart contract, which is further summarized in ticket CRW-01-001. Even though the provided UI project was not necessarily in scope since it constituted a PoC application, the auditing team nonetheless identified sensitive information hardcoded in the source code, which is discouraged even in a demo application context (see CRW-01-003).

Furthermore, Cure53's analyses verified that the smart contract did not implement a circuit brake functionality, which could be adopted to pause any further transactions in the event of an emergency (e.g. an emergent exploitation circumstance). Finally, the testers acknowledged the absence of an adequate process to enable detection of exploitations at the point of origination, as stipulated in ticket CRW-01-004. In this respect, however, one should note that the development team has already stated their intention to incorporate logging and monitoring in the near future.

Generally speaking, Cure53 completed this project having observed ample evidence that first-rate security performance was of high priority during the system's initial construction. Similarly, the developers comply with a swathe of best practice measures related to secure software development. Nevertheless, one can highly advise resolving all tickets raised herein to raise the platform to a first-rate security standard.

Moving forward, ChromaWay ETH Bridge would certainly benefit from regular security assessments. The complexity of the system poses myriad challenges from a security perspective, and amendments installed in the system may incur an exponential (and detrimental) effect on other aspects if unconsidered.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Cure53 would like to thank August Botsford, Viktor Plane, Ludvig Öberg, Robert Wideberg, Rayyan Jafri, Ha Dang, and Thomas Barker from the ChromaWay AB team for their excellent project coordination, support, and assistance, both before and during this assignment.