Fine penetration tests for fine websites

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

Pentest-Report Axelar Core & Components 04.2022

Cure53, Dr.-Ing. M. Heiderich, M. Rupp, Dr. A. Pirker, Dipl.-Ing. D. Gstir,
R. Weinberger, D. Oberhollenzer

Index

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

Introduction

"Axelar is a decentralized interoperability network connecting all blockchains, assets and apps through a universal set of protocols and APIs. Deploy your dApp on the blockchain best suited for your use case. Through a single integration with the Axelar API unlock access to multiple interconnected Axelar networks, users, assets, liquidity and data."

From https://axelar.network/

This report - entitled AXE-02 - details the scope, results, and conclusory summaries of a cryptography review, security assessment, and source code audit against the Axelar Core and associated components. The work was requested by the Axelar Foundation in December 2021 and initiated by Cure53 in April 2022, namely in CW14 and CW15. A total of twenty-eight days were invested to reach the coverage expected for this project. The testing conducted for AXE-02 was divided into three separate work packages (WPs) for execution efficiency, as follows:

- **WP1**: Security assessments & code reviews against Axelar Core & components
- **WP2**: Security assessments & code reviews against Axelar JS SDK
- **WP3**: Security assessments & code reviews against Axelar web and backend

The vast majority of components within this test scope have already been subject to assessment in previous audit engagements, as per report AXE-01. This review therefore marks the second test iteration against most of these scope items by the Cure53 testing team.

Cure53 was provided with source codes, test-environment access, test-assisting information and documentation, as well as any alternative means of access required to complete the audit. For these purposes, the methodology chosen was white-box and a team of six senior testers was assigned to the project's preparation, testing, audit execution, and finalization. All preparatory actions were completed in late March and early April 2022, namely in CW13, to ensure that the testing phase could proceed without hindrance.

Communications were facilitated via a dedicated, shared Slack channel deployed to combine the workspaces of Axelar and Cure53, thereby allowing an optimal collaborative working environment to flourish. All participatory personnel from both parties were invited to partake throughout the test preparations and discussions. One can denote that communications proceeded smoothly on the whole. The scope was well-prepared and clear, no noteworthy roadblocks were encountered throughout testing, and cross-team queries were kept to a minimum as a result.

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

Axelar delivered excellent test preparation and assisted the Cure53 team in every respect to procure maximum coverage and depth levels for this exercise.

Cure53 gave frequent status updates concerning the test and any related findings, whilst simultaneously offering prompt queries and receiving efficient, effective answers from the maintainers. Live reporting was not requested, which in hindsight may have proved useful considering the relatively high volume and severity level of the findings detected.

Regarding the findings in particular, the Cure53 team achieved comprehensive coverage over the WP1 through WP3 scope items, identifying a total of sixteen. Three of these findings were categorized as security vulnerabilities, whilst the remaining thirteen were deemed general weaknesses with lower exploitation potential. Generally speaking, the overall volume of findings detected could be perceived as relatively high compared with similarly scoped audits. However, the fact that the majority of findings were considered general weaknesses rather than security vulnerabilities - and should additionally be trivially easy to address and mitigate - remains a positive indication of the platform's security strength.

Nevertheless, this viewpoint should not detract from the inherent and continued risk of the three security vulnerabilities identified during testing, one of which was assigned a *High* severity. This issue in particular, which pertains to the risk-laden usage of local storage, should be reviewed at the earliest possible convenience. Additionally, the fact that half of all findings were discovered within WP1 indicates that a strong focus should be placed on bolstering the items included in this work package. Ultimately, the conclusion can be made that this audit identified abundant leeway for targeted improvements. Even though no issue deemed *Critical* in severity was discovered, a host of upgrades, mitigations, and best-practice implementations are required in order to reach a first-rate security posture.

The report will now shed more light on the scope and testing setup as well as provide a comprehensive breakdown of the available materials. Subsequently, the report will list all findings identified in chronological order, starting with the detected vulnerabilities and followed by the general weaknesses unearthed. Each finding will be accompanied by a technical description and Proof of Concepts (PoCs) where applicable, plus any relevant mitigatory or preventative advice to action.

In summation, the report will finalize with a conclusion in which the Cure53 team will elaborate on the impressions gained toward the general security posture of the Axelar Core and associated components, giving high-level hardening advice where applicable.

Scope

- **Cryptography reviews, security assessments and source-code audits against Axelar core and components**
    - **WP1**: Security Assessments & Code Reviews against Axelar Core & Components
        - Key focus areas
            - General message passing between EVM chains
            - External ERC-20 token transfers between supported chains
        - Sources were shared with Cure53
            - https://github.com/axelarnetwork/axelar-core
            - https://github.com/axelarnetwork/axelar-cgp-solidity
    - **WP2**: Security Assessments & Code Reviews against Axelar JS SDK
        - Sources were shared with Cure53
            - https://github.com/axelarnetwork/axelarjs-sdk/tree/v0.5.0-alpha.8
    - **WP3**: Security Assessments & Code Reviews against Axelar Web & Backend
        - Satellite token transfer app:
            - https://satellite.axelar.network/
        - Sources were shared with Cure53
            - https://github.com/axelarnetwork/axelar-web-app/tree/develop
            - https://github.com/axelarnetwork/axelar-bridge-nestjs/tree/develop
    - **Documentation**
        - **Developer docs:**
            - https://docs.axelar.dev/roles/dev
        - **Local environment testing:**
            - https://github.com/axelarnetwork/axelar-local-dev
    - **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

Identified Vulnerabilities

The following sections list all vulnerabilities and implementation issues identified throughout the testing period. Please note that findings are listed in chronological order rather than by their degree of severity and impact. The aforementioned severity rank is simply given in brackets following the title heading for each vulnerability. Furthermore, each vulnerability is given a unique identifier (e.g., *AXE-02-001*) to facilitate any future follow-up correspondence.

AXE-02-009 WP2: Local storage usage for wallet mnemonic *(High)*

Whilst reviewing the *axelarjs-sdk-0.5.0-alpha.9* repository, the observation was made that the SDK's wallet implementation utilizes the browser's local storage to store the mnemonic protecting the wallet. Any data within the local storage remains persistent and requires manual deletion for removal or overwriting.

The described behavior becomes particularly problematic in shared desktop environments, whereby multiple users potentially use the same desktop computer to connect to the Internet. In a scenario of this nature, users could read the browser's local storage to obtain sensitive information from other users. A malicious user could, for example, abuse the retrieved information from the local storage to utilize the wallet.

Affected file:
axelarjs-sdk-0.5.0-alpha.9/src/utils/wallet.ts

Affected code:
```
export function createWallet() {
       if (globalThis.localStorage) {
              const mnemonic = globalThis.localStorage.getItem("axelar-wallet");
              if (mnemonic) {
                     const wallet = ethers.Wallet.fromMnemonic(mnemonic);
                     return wallet;
              } else {
                     const wallet = ethers.Wallet.createRandom();
                     globalThis.localStorage.setItem(
                     "axelar-wallet",
                     wallet._mnemonic().phrase
                     );
                     return wallet;
              }
       [...]
```

To mitigate this issue, it is recommended to desist storing sensitive data such as wallet mnemonics within the local storage.

AXE-02-013 WP1: Front-running attacks on ERC20 tokens *(Medium)*

Whilst evaluating the *axelar-cgp-solidity-main* repository, the discovery was made that a smart contract implementation for ERC20 tokens is integrated. For this purpose, the contract offers several functions, including *approve* and *transferFrom*. The client uses the *approve* function to set an allowance for a so-called spender, which corresponds to the maximum amount the spender can transfer. The spender uses the *transferFrom* function to transfer the amount from a sender to a recipient, which simultaneously decreases the spender's allowance regarding the sender. However, testing confirmed that the contract is vulnerable to front-running attacks, whereby the order of invocations on a smart contract leads to an undesirable state.

An attacker with an initial sender allowance of *N* could wait until the sender calls the *approve* function with amount *M*, then subsequently invoke the *transferFrom* function for amount *N*. Depending on the ordering of invocations - which the attacker can influence using gas - the smart contract may execute the attacker's *transferFrom* invocation with amount *N* before the *approve* function. This behavior essentially allows an attacker to transfer an amount constituting *N+M*.

Affected file:
axelar-cgp-solidity-main/src/ERC20.sol

Affected code:
```
function approve(address spender, uint256 amount) public virtual override
returns (bool) {
       _approve(_msgSender(), spender, amount);
       return true;
}
[...]
function transferFrom(
       address sender,
       address recipient,
       uint256 amount
) public virtual override returns (bool) {
       _transfer(sender, recipient, amount);
       _approve(sender, _msgSender(), allowance[sender][_msgSender()] - amount);
       return true;
}
```

To mitigate this issue, one can recommend implementing a pre-commit scheme[1] to eliminate the potential risk of front-running.

[1] https://ethereum-contract-security-techniques-and-tips.readthedocs.io/en/latest/[...]-tod-front-running

AXE-02-014 WP1: Lack of recipient validation in ERC20 token transfer (*Medium*)

Similarly to the previous ticket, another weakness was identified following the discovery that the *axelar-cgp-solidity-main* repository includes a smart contract implementation for ERC20 tokens. For that purpose, the contract implements the *transferFrom* function, which allows a spender to transfer a defined amount from the sender to a recipient. The maximum amount a spender is able to transfer is determined by the spender's allowance. However, the contract does not verify the recipient of the transfer.

An attacker with a sender allowance would be able to transfer the allowed amount to arbitrary recipients. Regarding this scenario and the Axelar complex, the recipient would already be known upon contract creation, and could therefore feasibly be checked when invoking the *transferFrom* method.

Affected file:
axelar-cgp-solidity-main/src/ERC20.sol

Affected code:
```
function transferFrom(
       address sender,
       address recipient,
       uint256 amount
) public virtual override returns (bool) {
       _transfer(sender, recipient, amount);
       _approve(sender, _msgSender(), allowance[sender][_msgSender()] - amount);
       return true;
}
```

To mitigate this issue, the recommendation can be made to provide the intended recipient(s) of the transfer to the contract constructor and subsequently verify the recipient(s) on calls to the *transferFrom* function.

Miscellaneous Issues

This section covers any and all noteworthy findings that did not lead to an exploit but might 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 way to be called. Conclusively, while a vulnerability is present, an exploit might not always be possible.

AXE-02-001 WP3: Manipulable asset transfer via proxies *(Info)*

Whilst assessing the *axelar-bridge-nestjs-develop* repository, the discovery was made that the backend supports asset transfer. During this process, the web application invokes the *POST transfer-assets* endpoints providing a Data Transfer Object (DTO) containing the transfer details. Additionally, the DTO contains a signature computed via the web application by leveraging the user's private key, over a so-called One-Time-Code (OTC). However, this signature does not take transfer details such as the destination address into account.

In this scenario, an attacker located between the web application and the backend would be able to modify transaction details, including the destination address. Therefore, an attacker could potentially re-route the transfer of assets to their destination.

One must emphasize that the communication between the web application and backend utilizes HTTPS. Nevertheless, some proxies and firewalls terminate SSL/TLS traffic when performing deep packet inspection, for example. This behavior may grant an attacker with access to such a proxy the opportunity to mount the aforementioned attack.

Affected file:
axelar-bridge-nestjs-develop/src/bridge/service/bridge.service.ts

Affected code:
```
async transferAssets(dto: TransferAssetsDto_v2, traceId?: string) {
       // verify that the otc is still valid
       const otc = await this.getOtcFromAddress(dto.publicAddress);

       // verify signature
       this.verifySignature(otc, dto.signature, dto.publicAddress);
       [...]
}
```

To mitigate this issue, one can advise signing the entire DTO rather than merely the message containing the OTC.

AXE-02-002 WP3: Log injection to bridge NestJS backend *(Low)*

Whilst reviewing the *axelar-bridge-nestjs-develop* repository, the observation was made that the bridge backend contains an endpoint for writing logs. Specifically, the *LogController* offers a logging method that accepts a log entry as a DTO. Here, testing confirmed this method lacks an authentication process and is therefore susceptible to log injection.

An attacker could inject arbitrary log entries to the bridge backend by sending HTTP requests to the unprotected log endpoint.

Affected file:
axelar-bridge-nestjs-develop/src/log/log.controller.ts

Affected code:
```
@Post()
log(@Headers('x-traceid') traceId: string, @Body() dto: LogPayload) {
       return this.logService.log(dto, traceId);
}
```

To mitigate this issue, one can advise integrating sufficient protection for the logging endpoint in terms of authentication. This could be achieved so as to ensure that only the satellite web application can send logging entries to the backend.

AXE-02-003 WP3: WebSocket room connection lacks authentication *(Low)*

Whilst evaluating the *axelar-bridge-nestjs-develop* repository, the discovery was made that the bridge backend accepts WebSocket connections. The primary purpose of these connections is to connect a client to a given room. The bridge backend communicates information concerning workflows via WebSocket rooms but does not authenticate the user during this process. As a result, the bridge backend accepts all incoming WebSocket connections for a room.

An attacker in possession of a room ID can connect to a WebSocket room and gather information regarding the transfer of assets. Depending on the nature of the information that the backend communicates to the user in this room, sensitive information could feasibly be leaked.

Affected file:
axelar-bridge-nestjs-develop/src/socket/socket.gateway.ts

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

Affected code:
```
@SubscribeMessage('room:join')
async onJoinRoom(
       @MessageBody() roomId: string,
       @ConnectedSocket() socket: Socket,
): Promise<boolean> {
       await socket.join(roomId);
       return true;
}
```

To mitigate this issue, one can recommend authenticating the user upon joining a WebSocket room. This can be achieved by storing information inside the backend containing details relating to the identity of the user that triggers the transfer of assets, and checking it upon joining. Subsequently, one should verify the identity of the user joining the web room via a signature, similarly to the *POST transfer-assets* API endpoint process.

AXE-02-004 WP3: Lack of validation upon asset transfer *(Info)*

Whilst reviewing the *axelar-bridge-nestjs-develop* repository, the discovery was made that the bridge backend supports an endpoint for transferring assets. When performing a transfer, the user sends a Data Transfer Object (DTO) to the backend containing the transfer details, providing an asset to be transferred by the backend. Before submitting the transfer to other microservices, the bridge backend derives an asset common key from the DTO.

An attacker could provide an asset that the bridge backend cannot map to an asset common key, which may cause further and hitherto unforeseen issues in other microservices.

Affected file:
axelar-bridge-nestjs-develop/src/bridge/service/bridge-utils.service.ts

Affected code:
```
getAssetCommonKeyBasedOnEnv(asset: string): string {
       const targetAsset = this.allAssets.find(
              (_asset) =>
                     _asset.common_key[this.environment].toLowerCase() ===
                     asset.toLowerCase(),
       );
       return targetAsset?.common_key[this.environment]?.toLowerCase();
}
```

To mitigate this issue, one can recommend desisting the submission of the transaction to other microservices, in the eventuality that no common asset key can be derived from the DTO.

AXE-02-005 WP3: Lack of rate limiting for asset transfer *(Low)*

Whilst assessing the *axelar-bridge-nestjs-develop* repository, the discovery was made that the bridge backend utilizes a One-Time-Code (OTC) to verify the user's identity upon initiating a transfer of assets. For that purpose, the user first acquires an OTC and signs a predefined message containing the OTC using its private key. By default, the OTC remains valid for 5 minutes, but the user can extend this validity period by asking again for an OTC, which yields the same OTC and thus extends the validity period. The user then provides this signature to the bridge backend upon initiating the transfer of assets, then the backend checks the signature. In the eventuality the signature check fails, the backend aborts the transfer; however, the endpoint does not perform any rate limiting.

An attacker could abuse the lack of rate limiting for the transfer-asset endpoint and attempt to brute-force the signature of the OTC message. Since these attacks mostly depend on the comparison of provided and expected signatures and how this process is implemented - and considering the fact that testing could not confirm the method by which the utilized package performs a check of this nature - the severity of this issue was downgraded to *Info.*

Affected file:
axelar-bridge-nestjs-develop/src/bridge/controller/bridge.controller.ts

Affected code:
```
@Post('transfer-assets')
@HttpCode(HttpStatus.OK)
transferAssets(
	@Body() dto: TransferAssetsDto_v2,
	@Headers('x-trace-id') traceId: string,
) {
	return this.bridgeService.transferAssets(dto, traceId);
}
```

To mitigate this issue, it is recommended to protect the transfer-assets endpoint by utilizing a rate limiter. This would thereby eliminate the risk of brute-force attacks.

AXE-02-006 WP1: Lack of admin zero-address check (*Info*)

Whilst reviewing the *axelar-cgp-solidity-main* repository, the discovery was made that the *AxelarGatewayMultisig.sol* and *AxelarGatewaySinglesig.sol* contracts both permit the definition of admins for critical operations. Here, admins are able to upgrade the setup of the smart contracts, as well as freeze or unfreeze tokens. Toward this, testing confirmed that the *_setAdmins* function does not check for zero addresses like the *_setOperators* or *_setOwners* functions.

Affected file:
axelar-cgp-solidity-main/src/AdminMultisigBase.sol

Affected code:
```
function _setAdmins(
       uint256 adminEpoch,
       address[] memory accounts,
       uint256 threshold
) internal {
       uint256 adminLength = accounts.length;

       if (adminLength < threshold) revert InvalidAdmins();

       if (threshold == uint256(0)) revert InvalidAdminThreshold();

       _setAdminThreshold(adminEpoch, threshold);
       _setAdminCount(adminEpoch, adminLength);

       for (uint256 i; i < adminLength; i++) {
              address account = accounts[i];
              // Check that the account wasn't already set as an admin for this
              epoch.
              if (_isAdmin(adminEpoch, account)) revert DuplicateAdmin(account);
              // Set this account as the i-th admin in this epoch (needed to we
              can clear topic votes in `onlyAdmin`).
              _setAdmin(adminEpoch, i, account);
              _setIsAdmin(adminEpoch, account, true);
       }
}
```

To mitigate this issue, one can advise checking for the zero address before adding an admin address to the storage of the smart contract.

Fix note*: One can pertinently note that this issue has also been discussed with the customer during the course of this security assessment; and even though the described behavior does not have a security impact at the time of testing, it was mutually agreed to list this issue within the report for completeness.*

AXE-02-007 WP1: Lack of approve-contract call parameter validation (*Info*)

Whilst reviewing the *axelar-cgp-solidity-main* repository, the discovery was made that the *AxelarGatewayMultisig.sol* and *AxelarGatewaySinglesig.sol* contracts both offer functions for approving a contract call with and without mint. For this purpose, the functions receive several parameters from a command; however, both contracts fail to verify the validity of these parameters.

Affected file:
axelar-cgp-solidity-main/src/AxelarGateway.sol

Affected code:
```
function _approveContractCall(
        bytes32 commandId,
        string memory sourceChain,
        string memory sourceAddress,
        address contractAddress,
        bytes32 payloadHash
) internal {
        _setContractCallApproved(commandId, sourceChain, sourceAddress,
        contractAddress, payloadHash);
        emit ContractCallApproved(commandId, sourceChain, sourceAddress,
        contractAddress, payloadHash);
}

function _approveContractCallWithMint(
        bytes32 commandId,
        string memory sourceChain,
        string memory sourceAddress,
        address contractAddress,
        bytes32 payloadHash,
        string memory symbol,
        uint256 amount
) internal {
        _setContractCallApprovedWithMint(
                commandId,
                sourceChain,
                sourceAddress,
                contractAddress,
                payloadHash,
                symbol,
                amount
        );
        emit ContractCallApprovedWithMint(
                commandId,
                sourceChain,
                sourceAddress,
                contractAddress,
                payloadHash,
                symbol,
                amount
        );
}
```

To mitigate this issue, one can advise implementing a parameter validation for the affected functions.

***Fix note**: One can pertinently note that this issue has also been discussed with the customer during the course of this security assessment; and even though the described behavior does not have a security impact at the time of testing, it was mutually agreed to list this issue within the report for completeness.*

AXE-02-008 WP1: Outdated dependency usage (*Info*)

Whilst examining the *axelar-core-main* repository, the discovery was made that an outdated dependency is utilized in *go.mod*. Various source files in *axelar-core-main* leverage the *btcec* module, which has been subject to many fixes in the past, such as that which has been described in the following commit:

https://github.com/btcsuite/btcd/commit/73f7eac903576be8c8c08b2069c7a400be047ec7

Whilst *v0.22.0-beta* constitutes the most recent release of *btcd* itself, the *btcec* module within *btcd* can be updated to a more recent release tag at the time of testing.

Affected file:
axelar-core-main/go.mod

Affected code:
```
require (
        github.com/armon/go-metrics v0.3.10
        github.com/axelarnetwork/tm-events v0.0.0-20220221000027-80a0d7f2077e
        github.com/axelarnetwork/utils v0.0.0-20220203232147-bf4a42f338e8
        github.com/btcsuite/btcd v0.22.0-beta
        github.com/btcsuite/btcutil v1.0.3-0.20201208143702-a53e38424cce
        github.com/cosmos/cosmos-sdk v0.45.1
```

To mitigate this issue, it is recommended to either state *btcec* directly in *go.mod* or use a *git sha* as the version for *btcd*.

AXE-02-010 WP1: Potentially-insecure nonce generation *(Info)*

Whilst assessing the *axelar-core-main* repository, the discovery was made that the *GetNonce* function will generate a low-grade nonce if a *gasMeter* is not supplied. In this eventuality, the function will return the sha256 value of 26 null bytes and ignore the supplied hash.

Affected file:
axelar-core-main/utils/slice.go

Affected code:
```
func GetNonce(hash tmbytes.HexBytes, gasMeter sdk.GasMeter) Nonce {
	bz := make([]byte, 16)
	if gasMeter != nil {
		binary.LittleEndian.PutUint64(bz, uint64(gasMeter.GasConsumed()))
		bz = append(bz, hash...)
	}
	return sha256.Sum256(bz)
}
```

To mitigate this issue, one can advise altering the function so as to ensure that omitting the *gasMeter* value will result in an error instead of generating a constant nonce.

AXE-02-011 WP1: Potential lack of signature check *(Info)*

Whilst evaluating the *axelar-core-main* repository, the discovery was made that the *Broadcast* function notes in the comment that *all* messages require a valid signature. However, the current implementation only checks whether the *first* message holds a signature.

Affected file:
axelar-core-main/cmd/axelard/cmd/vald/broadcaster/broadcast.go

Affected code:
```
// Broadcast bundles the given messages into a single transaction and submits it
to the blockchain.
// If there are more than one message, all messages must have the single same
signer
func Broadcast(ctx sdkClient.Context, txf tx.Factory, msgs []sdk.Msg)
(*sdk.TxResponse, error) {
	if len(msgs) == 0 {
		return nil, fmt.Errorf("call broadcast with at least one message")
	}
```

```
// By convention the first signer of a tx pays the fees
if len(msgs[0].GetSigners()) == 0 {
        return nil, fmt.Errorf("messages must have at least one signer")
}
```

To mitigate this issue, one can advise reviewing the need for this function entirely. If deemed required, either the comment or the code should be sufficiently fixed.

AXE-02-012 WP2: Lack of null checks (*Info*)

Whilst reviewing the *axelarjs-sdk* repository, the discovery was made that the *joinRoomAndWaitDepositConfirmationEvent* function seems to support the case whereby the *waitCb* function is null, but during time-out assumes that *waitCb* is always a callable function.

Affected file:
axelarjs-sdk-0.5.0-alpha.9/src/services/SocketService.ts

Affected code:
```
public joinRoomAndWaitDepositConfirmationEvent(roomId: string, waitCb: any) {
        return new Promise(async (resolve) => {
        await this.createSocket();
        const ms = 1.8e6; //30 minutes
        const timeout = setTimeout(() => {
        waitCb({ timedOut: true });
        this.disconnect();
        }, ms);
        this.socket.emit("room:join", roomId, () => {
        this.socket.on("bridge-event", (data: any) => {
        waitCb && waitCb(data);
        resolve(data);
        clearTimeout(timeout);
        this.disconnect();
        });
```

To mitigate this issue, one can advise calling *waitCb* only in the eventuality that it does not constitute *null*.

AXE-02-015 WP3: Use of outdated JavaScript dependencies *(Low)*

Whilst reviewing the *axelar-web-app-develop* repository, the discovery was made that outdated dependencies are referenced. Specifically, the *nanoid* and *axios* dependencies utilized by this repository contain known vulnerabilities.

Affected file:
axelar-web-app-develop/package.json

The presence of this weakness can be verified by running *npm audit* after *npm install*:

```
$ npm audit
[...]
axios  <0.21.2
Severity: high
Incorrect Comparison in axios - https://github.com/advisories/GHSA-cph5-m8f7-
6c5x
fix available via `npm audit fix --force`
Will install @keplr-wallet/types@0.8.8, which is a breaking change
node_modules/secretjs/node_modules/axios
  secretjs  <=0.10.4 || 0.12.0 - 0.17.6-beta.6
  Depends on vulnerable versions of axios
  node_modules/secretjs
    @keplr-wallet/types  >=0.8.11-rc.0
    Depends on vulnerable versions of secretjs
    node_modules/@keplr-wallet/types
      @keplr-wallet/cosmos  >=0.8.11-rc.0
      Depends on vulnerable versions of @keplr-wallet/types
      node_modules/@keplr-wallet/cosmos
      @keplr-wallet/unit  >=0.8.11-rc.0
      Depends on vulnerable versions of @keplr-wallet/types
      node_modules/@keplr-wallet/unit
[...]
nanoid  3.0.0 - 3.1.30
Severity: moderate
Exposure of Sensitive Information to an Unauthorized Actor in nanoid - 
https://github.com/advisories/GHSA-qrpm-p2h7-hrv2
fix available via `npm audit fix`
node_modules/nanoid
[...]
```

To mitigate this issue, one can recommend updating these dependencies to the latest version available, thereby mitigating all known associated vulnerabilities.

AXE-02-016 WP1: Potential Cosmos consensus stall via panic (*Medium*)

Whilst evaluating the *axelar-core-main* repository, the observation was made that all modules except *x/evm* do not recover from a *panic()* in the *EndBlocker* module handler. As the Cosmos SDK documentation states, the code in *BeginBlocker* and *EndBlocker* can decelerate or even halt the chain.[2] Thus, any *panic* that is triggered therein can have fatal consequences for the chain's operation, resulting in a Denial-of-Service scenario in the eventuality an attacker is able to trigger this remotely.[3]

Two examples wherein this behavior could occur within the Axelar framework are the *x/tss* and *x/reward* modules. Both run code in their respective *EndBlocker* handlers, which can trigger *panic()* calls. Besides the obvious *panic()* calls directly in the respective *abci.go* files, other less obvious calls can also trigger a panic. For example, *sdk.NewCoin(asset, amount)* will panic if the value of *asset* does not match a regular expression pattern.

Affected files:

- *axelar-core-main/x/reward/abci.go*
- *axelar-core-main/x/tss/abci.go*

To mitigate this issue, one can recommend integrating recovery logic to all Cosmos module *EndBlocker* handlers to avoid a chain stall.

[2] https://docs.cosmos.network/master/building-modules/beginblock-endblock.html
[3] https://github.com/althea-net/cosmos-gravity-bridge/issues/348

Conclusions

The impressions gained during this report - which details and extrapolates on all findings identified during the CW14 and CW15 testing against the Axelar Core and associated components by the Cure53 team - will now be discussed at length. To summarize, the confirmation can be made that the components under scrutiny have garnered a mixed impression.

The assessment focused on seven repositories in total, namely:

- *axelar-bridge-nestjs-develop*
- *axelar-bridge-worker-nestjs-main*
- *axelar-cgp-solidity-main*
- *axelar-core-main*
- *axelar-local-dev-main*
- *axelar-web-app-develop*
- *axelarjs-sdk-0.5.0-alpha.9*

The repositories were grouped into three work packages with individual scope assignments for each. Following the allocation of twenty-eight work days upon the targets in scope, six senior members of the Cure53 testing team detected a total of sixteen findings. The uncovered presence of these issues underlines the essential requirement for extensive improvements in the affected areas. In essence, Cure53 observed ample evidence of both security strength and weakness within the complex and its surroundings.

Generally speaking, the completion of the audit confirmed that some Axelar complex components were not always capable of handling all risk scenarios, exposing room for improvement across various aspects and endpoints. However, that only three vulnerabilities were unveiled and documented is worthy of mention. The remaining thirteen identified issues represent flaws, which is a relatively high volume for a scope of this complexity. Crucially, none of the issues detected received a *Critical* impact marker. From a technical viewpoint, a selection of endpoints lacked sufficient safeguarding against attack and required security-centered upgrades. The following paragraphs shed further light on the details.

The first work package corresponds to the security assessment and code review of Axelar Core and associated components. The primary scope pertained to general message passing between EVM chains and external ERC-20 token transfers between supported chains.

The source code of the repositories for this WP is compositionally vast and highly complex. The repositories include the software upon which nodes and validators operate - written in Golang - as well as smart contracts of the software complex, which is written in Solidity. Toward this, heightened effort was required for the testers to understand the method by which the Axelar Core and smart contracts interact and operate. However, the code base itself is soundly composed and efficiently organized; the testing team detected evidence of enhancements applied following previous security assessments and general awareness of secure programming principles.

In summation for the first work package, two vulnerabilities and six miscellaneous issues were identified. The two vulnerabilities were both identified within the smart contracts. The first pertains to front-running attacks upon the ERC-20 contract, which would permit an attacker to spend more than the sender intends via the ordering of contract invocations. This issue is documented via ticket AXE-02-013.

The second vulnerability, as documented in ticket AXE-02-014, pertains to an absent check regarding the recipient of an ERC-20 transfer, which could lead to a transfer to an unintended recipient. All miscellaneous issues here correspond to generic bugs within the smart contracts, outdated dependencies, and a lack of cryptographic checks. Another noteworthy class of issues discovered during this audit constitutes the lack of panic recovery upon the Axelar modules' *EndBlocker* handler. The testing team also noted that alternative *x/evm* modules such as *x/reward* and *x/tss* also lack handlers, which could potentially facilitate DoS attacks.

In summary, sufficient and thorough coverage for this work package was achieved. However, one must emphasize that some code-paths still require a deep-dive review; not all code-paths were comprehensively examined owing to the size of the repositories in scope and the allocated time frame of this audit. Beyond the core component, Cure53 investigated the Axelar JS SDK comprising the second work package. The primary objective here was to deep-dive review the SDK for common attack vectors and scenarios against SDKs, including (but not limited to) information leakage, use of improper cryptography, logical flaws that facilitate security issues, and misuse of the SDKs API.

The SDK in question is medium-sized and implemented in Typescript; the testing team observed a cleanly written and organized repository source code for this work package. In total, only one vulnerability and one miscellaneous issue was identified for this work package, which is an excellent outcome for these components in scope. The vulnerability - as documented in ticket AXE-02-009 - corresponds to a wallet mnemonic leakage due to the continued usage of the browser's local storage, whilst the miscellaneous issue concerns a lack of null checks in the SDK (see AXE-02-012).

The third and final work package constituted a security assessment and code review of the Axelar web and backend. Here, the primary objective was to review the scope items for common vulnerabilities that typically blight web applications and backends, including (but not limited to) DoS, absent authentication and authorization processes, information leakage, cross-site-scripting, and unsafe cryptography.

The repositories for this WP were deemed moderate in size. The web application is written in React.JS, whereas both backend repositories are written in NodeJS (via Typescript). All associated repositories exhibited a clean structure and strong coding style, which made it significantly easier for Cure53 to understand all components when applying testing techniques.

Notably, updated versions of some source-code areas were provided by the customer after the commencement of testing. Whilst appreciated, these updates bestowed significant alterations to the code previously provided. This corroborates the viewpoint that the code is still in a constant state of development, therefore one can strongly advise conducting additional code audits once this evolutionary framework iteration has settled.

In summary for WP3, positively the testing team only detected five miscellaneous issues, all of which were considered of minor severity. These primarily concern a lack of protection upon various endpoints from an authentication perspective, as well as a lack of rate limiting. Generally speaking, the reviewed source code repositories garnered a positive impression from a programming perspective. The platform exhibited an observably clean coding style and displayed evidence of best-practice adherence, thereby contributing to the overall strength of the complex.

The components in scope would certainly benefit from hardening improvements, even though security was evidently a concern during the implementation process. Addressing the identified vulnerabilities within the smart contracts and the insecure storage of the wallet mnemonic will help to minimize the attack surface and elevate the security posture of Axelar.

The Axelar software project would certainly benefit from recurrent security assessments, as the immense complexity of the core components in particular is challenging to handle from a security perspective. This would also help to ensure that no stone is left unturned regarding the detection of lingering security issues, which may have been the case considering the sheer magnitude of the application in its entirety. Additionally, any alterations made within one system area may have an unintentional security impact elsewhere, which yet again highlights the importance of regular security auditing.

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

In conclusion, this project draws attention to specific components that require security enhancement in order to elevate the complex. One can strongly recommend persistent planning and execution of external and internal actions on security aspects, particularly before any substantial changes are deployed. This would ensure that new features cannot introduce undesired security vulnerabilities. This strategy will undoubtedly reduce the volume of present security issues and increase the platform's resilience against attacks in general.

In Cure53's opinion, future research should also encompass internal architecture and infrastructure investigations, which would provide significant value toward determining whether the components are as protected internally as they are externally. These efforts will undoubtedly provide significant assistance toward identifying more intricate and deeply-entrenched issues that may occur during the development process. Cure53 will gladly assist in further engagements to ensure that all security-strengthening objectives are achieved in this regard.

Cure53 would like to thank Vladimir Agaev, Canh Trinh, Canh Trinh, Sammy Liu, and Kiryl Yermakou from the Axelar Foundation team for their excellent project coordination, support and assistance, both before and during this assignment.