METATRUST

Security Assessment for

DLC.link-attestor

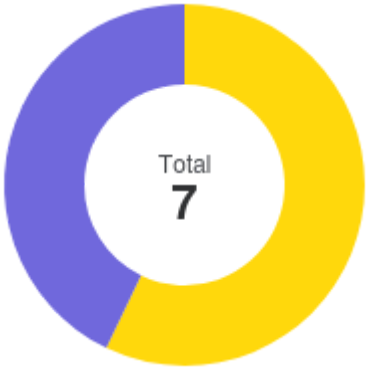
May 05, 2024

Executive Summary

Overview	
Project Name	DLC.link-attestor
Codebase URL	https://github.com/DLC-link/dlc-attestor-stack
Scan Engine	Security Analyzer
Scan Time	2024/05/05 00:02:45
Commit Id	8a3775d36f2f8eea0815b737d23f36c69378264b

Total	
Critical Issues	0
High risk Issues	0
Medium risk Issues	4
Low risk Issues	3
Informational Issues	0

Critical Issues	The issue can cause large economic losses, large-scale data disorder, loss of control of authority management, failure of key functions, or indirectly affect the correct operation of other smart contracts interacting with it.
High Risk Issues	The issue puts a large number of users' sensitive information at risk or is reasonably likely to lead to catastrophic impacts on clients' reputations or serious financial implications for clients and users.
Medium Risk Issues	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
Low Risk Issues	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low-impact in view of the client's business circumstances.
Informational Issue	The issue does not pose an immediate risk but is relevant to security best practices or Defence in Depth.



Critical Issues	0%	0
High risk Issues	0%	0
Medium risk Issues	57%	4
Low risk Issues	43%	3
Informational Issues	0%	0

Summary of Findings

MetaScan security assessment was performed on **May 05, 2024 00:02:45** on project **DLC.link-attestor** with the repository on branch **default branch**. The assessment was carried out by scanning the project's codebase using the scan engine **Security Analyzer**. There are in total **7** vulnerabilities / security risks discovered during the scanning session, among which **4** medium risk vulnerabilities, **3** low risk vulnerabilities,

ID	Description	Severity	Alleviation
MSA-001	CSRF due to CORS middleware with a setting that allows requests from any origin	Medium risk	Acknowledged
MSA-002	Data Synchronization Mechanism May Result in Loss of Event Data Between Peers	Medium risk	Acknowledged
MSA-003	Failure to Transition EVMDLCStatus::Closed Events to PsbtEventStatus::Closed Status in update_closing_psbt_events_to_closed Function	Medium risk	Fixed
MSA-004	Lack of authentication for http server	Medium risk	Acknowledged
MSA-005	`serde_json::to_writer` does not sync the data, and thus does not guarantee the data ends up on disk	Low risk	Fixed
MSA-006	Base64 malleable risk	Low risk	Acknowledged
MSA-007	Inadequate Signature Validation in AttestorEVMSignature Constructor	Low risk	Fixed

Findings

Medium risk (4)

1. CSRF due to CORS middleware with a setting that allows requests from any origin

Medium risk | Security Analyzer

In routers module, the function `build_success_response` and `build_error_response` are used to build responses with several header configurations. These configurations allow requests from any origin by setting `AllowOrigins` to `*`, which may open a CSRF(Cross-site Request Forgery) vulnerability for the application. This is because it allows any website to access the API by sending requests, allowing an attacker to potentially make unauthorized requests. Reference: https://portswigger.net/web-security/cors/access-control-allow-origin

File(s) Affected

routes/mod.rs #25-33

```
25 pub(crate) fn build_success_response(message: String) -> Result<Response<Body>, GenericError> {
26     Ok(Response::builder()
27         .status(StatusCode::OK)
28         .header(header::CONTENT_TYPE, "application/json")
29         .header(header::ACCESS_CONTROL_ALLOW_ORIGIN, "*")
30         .header(header::ACCESS_CONTROL_ALLOW_METHODS, "*")
31         .header(header::ACCESS_CONTROL_ALLOW_HEADERS, "*")
32         .body(Body::from(message.to_string()))?)
33 }
```

routes/mod.rs #45-65

```
45 pub(crate) fn build_error_response(message: String) -> Result<Response<Body>, GenericError> {
46     warn!("error response: {}\n", message);
47     Ok(Response::builder()
48         .status(StatusCode::BAD_REQUEST)
49         .header(header::ACCESS_CONTROL_ALLOW_ORIGIN, "*")
50         .header(header::ACCESS_CONTROL_ALLOW_METHODS, "*")
51         .header(header::ACCESS_CONTROL_ALLOW_HEADERS, "*")
52         .header(header::CONTENT_TYPE, "application/json")
53         .body(Body::from(
54             json!(
55                 {
56                     "status": StatusCode::BAD_REQUEST.as_u16(),
57                     "errors": vec![ErrorResponse {
58                         message: message.to_string(),
59                         code: None,
60                     }],
61                 }
62             )
63             .to_string(),
64         ))?)
65 }
```

Recommendation

Consider restricting the allowed origins to only trusted websites.

Alleviation Acknowledged

-

2. Data Synchronization Mechanism May Result in Loss of Event Data Between Peers

Medium risk Security Analyzer

The AttestorServer.setup function is configured with num_recent_event_to_sync set to 5, and the AttestorServer.start_periodic_checker function sets the periodic_check_interval_seconds to 60. The num_recent_event_to_sync parameter is utilized within the attestor.sync_recent_psbt_events function to synchronize the latest 5 added events. This setup implies a potential data synchronization issue where if a peer adds more than 5 new events within a 60-second window, any events beyond the fifth will not be read by other peers.

For instance, if a peer adds 10 events(N1-N10) within 60 seconds, other peers synchronizing through the AttestorServer.start_periodic_checker function will only obtain the latest 5(N6-N10) events, missing the first 5(N1-N5) events. In the next 60 seconds period, although other peers will synchronize again, they will never obtain the (N1-N5) events from this peer.

This synchronization mechanism will cause that when the number of new additions to a single peer is greater than the num_recent_event_to_sync value, other peers will not be able to fully synchronize their new information, resulting in inconsistent data synchronization.

File(s) Affected

attestor_server.rs #74-77

```
74          let num_recent_event_to_sync = env::var("NUM_RECENT_EVENT_TO_SYNC")
75              .unwrap_or("5".to_string())
76              .parse::<i64>()
77              .unwrap_or(5);
```

attestor_server.rs #160-163

```
160          task::spawn(async move {
161              let periodic_check_interval_seconds: u64 = env::var("PERIODIC_CHECK_INTERVAL_SECONDS")
162                  .unwrap_or("60".to_string())
163                  .parse::<u64>()
```

Recommendation

Revise the data synchronization strategy to ensure that all events within a synchronization cycle (60 seconds) are accounted for and shared among peers. This could involve:

- Keeping track of the last synchronized event ID or timestamp and requesting all new events since that marker during the next synchronization cycle.
- Implementing a more dynamic synchronization mechanism that can handle variable numbers of events, potentially with pagination or batch processing to ensure no events are missed.
- Adding a reconciliation process that periodically checks for any missed events over a longer time span to correct any synchronization gaps that may occur.

The synchronization method within sync_recent_psbt_events should be refined to ensure comprehensive coverage of all events within the specified interval.

Alleviation Acknowledged

-

3. Failure to Transition EVMDLCStatus::Closed Events to PsbtEventStatus::Closed Status in update_closing_psbt_events_to_closed Function

Medium risk Security Analyzer

The update_closing_psbt_events_to_closed function in the Attestor module is designed to update the backend system status of PSBT events that are currently in the `PsbtEventStatus::Closing` state. According to the implementation, if an event is found with an `EVMDLCStatus::Closing`, the function proceeds to check if the associated Bitcoin transaction is confirmed and, if so, updates the event's status to `PsbtEventStatus::Closed`. However, there is no handling in the function for events that are already in the `EVMDLCStatus::Closed` state in the smart contract.

In other words, when an event satisfies both the `PsbtEventStatus::Closing` and `EVMDLCStatus::Closed` states, it will not be able to update from the `PsbtEventStatus::Closing` state to the `PsbtEventStatus::Closed` state.

File(s) Affected

attestor/mod.rs #1267-1300

```
1267    pub async fn update_closing_psbt_events_to_closed(&self) {
1268        let Ok(closing_events) = self.get_closing_psbt_events().await else {
1269            error!("Error getting closing psbt events");
1270            return;
1271        };
1272
1273        for event in closing_events {
1274            let loop_iteration_result: Result<(), GenericAttestorError> = async {
1275                let dlc = self
1276                .bi_handler.get_dlc_info(&event.uuid, &event.chain_name)
1277                    .await?;
1278                match dlc.status {
1279                    EVMDLCStatus::Closing => {
1280                         // Still in the closing state on-chain: this is expected. Let's check if the
1281                         self.try_close_psbt(event.clone()).await?;
1282                    }
1283                    _ => {
1284                        error!("[State Machine Validation] Got a PSTB event in the {} state that is i
1285                               the {} state on-chain... no repair path known. skipping", event.status
1286                    }
1287                }
1288                Ok(())
1289            }
1290            .await;
1291
1292            match loop_iteration_result {
1293                Ok(_) => (),
1294                Err(e) => error!(
1295                    "Error updating closing psbt events to closed: {}",
1296                    e.message
1297                ),
1298            }
1299        }
1300    }
```

Recommendation

To ensure the backend system accurately reflects the actual state of the smart contract, the update_closing_psbt_events_to_closed function needs to be modified to handle EVMDLCStatus::Closed events appropriately. The following code should be implemented:

```
match dlc.status {
    EVMDLCStatus::Closing => {
        // Existing logic to handle Closing status...
    }
    EVMDLCStatus::Closed => {
        // If the smart contract status is already Closed, update the backend status accordingly
        event.status = PsbtEventStatus::Closed;
        self.update_psbt_event(event.clone()).await?;
    }
    // Handle other cases...
}
```

This change will ensure that events with EVMDLCStatus::Closed and PsbtEventStatus::Closing status in the smart contract are correctly transitioned to PsbtEventStatus::Closed in the backend, maintaining consistency across the system.

Alleviation Fixed

-

4. Lack of authentication for http server

Medium risk Security Analyzer

In **`main.rs`**, the **`main()`** function will start a server to handle http request:

```
 let http_service = make_service_fn(move || {
        let attestor = attestor.clone();
        let threshold_handler = threshold_signature_handler.clone();
        let peers = peers.clone();
        async move {
_            Ok::<_, GenericError>(service_fn(move |req| {
                handle_request(
                    req,
                    attestor.clone(),
                    threshold_handler.to_owned(),
                    peers.clone(),
                )
            }))
        }
    });

    #[allow(clippy::expect_used)]
    let address: std::net::SocketAddr = (attestor_ip, attestor_backend_port).into();

    let server = Server::bind(&address).serve(http_service);
```

The issue is that it lacks of authentication mechanism. If a hacker discovers the IP and port of this server, he can directly invoke the http request to perform dangerous operations, for example, executing the **`/create***`** request to create a large amount of psbt events.

File(s) Affected

main.rs #70-84

```
70      let http_service = make_service_fn(move |_| {
71          let attestor = attestor.clone();
72          let threshold_handler = threshold_signature_handler.clone();
73          let peers = peers.clone();
74          async move {
75              Ok::<_, GenericError>(service_fn(move |req| {
76                  handle_request(
77                      req,
78                      attestor.clone(),
79                      threshold_handler.to_owned(),
80                      peers.clone(),
81                  )
82              }))
83          }
84      });
```

Recommendation

Consider using JWT or making sure the server will not be exposed to public.

Alleviation Acknowledged

-

Low risk (3)

1. `serde_json::to_writer` does not sync the data, and thus does not guarantee the data ends up on disk

Low risk Security Analyzer

In function `write_public_key_package_and_key_package_to_file`, `serde_json::to_writer`will indeed call `fs::write`. But it does not guarantee the data has arrived to the destination when `fs::write` returns. At least on Windows if you use `fs::write` and then reboot, the data is never written to disk.

File(s) Affected

threshold/mod.rs #127-137

```
127     pub fn write_public_key_package_and_key_package_to_file(
128         &self,
129         public_key_package: PublicKeyPackage,
130         key_package: KeyPackage,
131     ) -> Result<(), ThresholdError> {
132         let identifier_as_string = serde_json::to_string(&self.identifier)?;
133         let file_name = format!("kp-{}.json", identifier_as_string.trim_matches('"'));
134         let file = std::fs::File::create(file_name)?;
135         serde_json::to_writer(file, &(key_package, public_key_package))?;
136         Ok(())
137     }
```

Recommendation

Consider following fix:

```
pub fn write_public_key_package_and_key_package_to_file(
        &self,
        public_key_package: PublicKeyPackage,
        key_package: KeyPackage,
    ) -> Result<(), ThresholdError> {
        let identifier_as_string = serde_json::to_string(&self.identifier)?;
        let file_name = format!("kp-{}.json", identifier_as_string.trim_matches('"'));
        let mut file = std::fs::File::create(file_name)?;
        serde_json::to_writer(file, &(key_package, public_key_package))?;
        file.sync_all()?;
        Ok(())
    }
```

Alleviation Fixed

-

2. Base64 malleable risk

Low risk Security Analyzer

Base64 is not a rigorous serialization algorithm and is not suitable for use in blockchain systems. You can find the potential decoding attack for Base64 here: https://eprint.iacr.org/2022/361.pdf

File(s) Affected

attestor/storage_handler.rs #75-75

```
75              base64::decode(event_response.content).map_err(AttestorError::Base64DecodeError)?;
```

Recommendation

Consider using Base58 instead

Alleviation Acknowledged

-

3. Inadequate Signature Validation in AttestorEVMSignature Constructor

Low risk Security Analyzer

An audit of the AttestorEVMSignature module revealed that the constructor's signature validation checks are insufficient. The new method only verifies the signature's length to be 132 characters and that it starts with 0x. No further validation is conducted to ensure the signature's structural integrity, such as checking the signature components (r, s, and v values).

The Solidity reference implementation provided from the repository (https://github.com/protofire/zeppelin-solidity/blob/master/contracts/ECRecovery.sol) offers a more robust validation scheme. It checks the signature length to be exactly 65 bytes (expected length of an ECDSA signature), extracts the r, s, and v components, and verifies that the v component is either 27 or 28 after normalization (or 0 and 1 before normalization), which are the only valid values for an Ethereum signature's recovery byte.

File(s) Affected

attestor/mod.rs #94-103

```
impl AttestorEVMSignature {
    pub fn new(signature: String) -> Result<Self, GenericAttestorError> {
        if signature.len() != 132 || &signature[0..2] != "0x" {
            return Err(GenericAttestorError {
                message: "Invalid signature length or format".to_string(),
            });
        }
        Ok(AttestorEVMSignature { signature })
    }
}
```

Recommendation

It is recommended to implement a more comprehensive signature validation mechanism in Rust, similar to the Solidity implementation. The proposed changes include:

3. Checking the signature length to match the expected length of an ECDSA signature.
4. Extracting the r, s, and v components from the signature.
5. Validating the recovery byte (v) to be within the expected range.

```
function recover(bytes32 hash, bytes sig) public pure returns (address) {
  bytes32 r;
  bytes32 s;
  uint8 v;

  //Check the signature length
  if (sig.length != 65) {
    return (address(0));
  }

  // Divide the signature in r, s and v variables
  assembly {
    r := mload(add(sig, 32))
    s := mload(add(sig, 64))
    v := byte(0, mload(add(sig, 96)))
  }

  // Version of signature should be 27 or 28, but 0 and 1 are also possible versions
  if (v < 27) {
    v += 27;
  }

  // If the version is correct return the signer address
  if (v != 27 && v != 28) {
    return (address(0));
  } else {
    return ecrecover(hash, v, r, s);
  }
}
```

The Rust implementation should be updated to include detailed validation similar to the ECRecovery library's recover function in Solidity.

Alleviation Fixed

-

Disclaimer

This report is governed by the stipulations (including but not limited to service descriptions, confidentiality, disclaimers, and liability limitations) outlined in the Services Agreement, or as detailed in the scope of services and terms provided to you, the Customer or Company, within the context of the Agreement. The Company is permitted to use this report only as allowed under the terms of the Agreement. Without explicit written permission from MetaTrust, this report must not be shared, disclosed, referenced, or depended upon by any third parties, nor should copies be distributed to anyone other than the Company.

It is important to clarify that this report neither endorses nor disapproves any specific project or team. It should not be viewed as a reflection of the economic value or potential of any product or asset developed by teams or projects engaging MetaTrust for security evaluations. This report does not guarantee that the technology assessed is completely free of bugs, nor does it comment on the business practices, models, or legal compliance of the technology's creators.

This report is not intended to serve as investment advice or a tool for investment decisions related to any project. It represents a thorough assessment process aimed at enhancing code quality and mitigating risks inherent in cryptographic tokens and blockchain technology. Blockchain and cryptographic assets inherently carry ongoing risks. MetaTrust's role is to support companies and individuals in their security diligence and to reduce risks associated with the use of emerging and evolving technologies. However, MetaTrust does not guarantee the security or functionality of the technologies it evaluates.

MetaTrust's assessment services are contingent on various dependencies and are continuously evolving. Accessing or using these services, including reports and materials, is at your own risk, on an as-is and as-available basis. Cryptographic tokens are novel technologies with inherent technical risks and uncertainties. The assessment reports may contain inaccuracies, such as false positives or negatives, and unpredictable outcomes. The services may rely on multiple third-party layers.

All services, labels, assessment reports, work products, and other materials, or any results from their use, are provided "as is" and "as available," with all faults and defects, without any warranty. MetaTrust expressly disclaims all warranties, whether express, implied, statutory, or otherwise, including but not limited to warranties of merchantability, fitness for a particular purpose, title, non-infringement, and any warranties arising from course of dealing, usage, or trade practice. MetaTrust does not guarantee that the services, reports, or materials will meet specific requirements, be error-free, or be compatible with other software, systems, or services.

Neither MetaTrust nor its agents make any representations or warranties regarding the accuracy, reliability, or currency of any content provided through the services. MetaTrust is not liable for any content inaccuracies, personal injuries, property damages, or any loss resulting from the use of the services, reports, or materials.

Third-party materials are provided "as is," and any warranty concerning them is strictly between the Customer and the third-party owner or distributor. The services, reports, and materials are intended solely for the Customer and should not be relied upon by others or shared without MetaTrust's consent. No third party or representative thereof shall have any rights or claims against MetaTrust regarding these services, reports, or materials.

The provisions and warranties of MetaTrust in this agreement are exclusively for the Customer's benefit. No third party has any rights or claims against MetaTrust regarding these provisions or warranties. For clarity, the services, including any assessment reports or materials, should not be used as financial, tax, legal, regulatory, or other forms of advice.