Polymath
Polymesh Feature Review

Security Assessment Report

POLYMATH

Prepared for Polymath, Inc.
September 30, 2021 (version 1.1)

Project Team:

Technical Testing	Bryan C. Geraghty and Loren Browman
Technical Editing	Sean Bradly and Lacey Kasten
Project Management	Molly Vukusich

Table of Contents

Engagement Overview

Assessment Components and Objectives

Polymath, Inc. ("Polymath") recently engaged Atredis Partners ("Atredis") to perform a review of four specific features that have been added to the Polymesh platform. Objectives included validation that the new features were developed with security best practices in mind, and to obtain third party validation that any significant vulnerabilities present in these features were identified for remediation.

Testing was performed from August 30, through September 14, 2021 by Bryan C. Geraghty and Loren Browman of the Atredis Partners team, with Molly Vukusich providing project management and delivery oversight. For Atredis Partners' assessment methodology, please see Appendix I of this document, and for team biographies, please see Appendix II. Specific testing components and testing tasks are included below.

COMPONENT	ENGAGEMENT TASKS
Polymath Polymesh Feature Review	
Assessment Targets	• Specific feature additions to the Polymesh blockchain: • Support for custom primary issuance agent and corporate action agent permissions • Pending settlement functionality to handle failed instructions • Ability for one key to pay for another key's transactions • Rewards logic to pay out rewards earned on the ITN as POLYX on the mainnet subject to a PIP being approved
Assessment Tasks	• Full review of agent permissions and settlement features • Source-assisted penetration testing of the features described in the Assessment Targets section above • Review of source code related to changes • Analyze each feature workflow from bootstrap through execution to identify code branches, associated threats, and potential vulnerabilities • Where code architecture allows, development of unit tests to cover abuse cases • Active testing and proof-of-concept development on local and Alcyone chains
Reporting and Analysis	
Analysis and Deliverables	• Status Reporting and Realtime Communication • Comprehensive Engagement Deliverable • Engagement Outbrief and Remediation Review

The ultimate goal of the assessment was to provide a clear picture of risks, vulnerabilities, and exposures as they relate to accepted security best practices, such as those created by the National Institute of Standards and Technology (NIST), Open Web Application Security Project (OWASP), or the Center for Internet Security (CIS). Augmenting these, Atredis Partners also draws on its extensive experience in secure development and in testing high-criticality applications and advanced exploitation.

Engagement Tasks

Atredis Partners performed the following tasks, at a high level, for in-scope targets during the engagement.

Application Penetration Testing

For relevant web applications, APIs and web services, Atredis performed automated and manual application penetration testing of these components, applying generally accepted testing best practices as derived from OWASP and the Web Application Security Consortium (WASC).

Testing was performed from the perspective of an anonymous intruder, identifying scenarios from the perspective of an opportunistic, Internet-based threat actor with no knowledge of the environment, as well as from the perspective of a user working to laterally move through the environment to bypass security restrictions and user access levels. Where relevant, Atredis Partners utilized both automated fuzzing and fault injection frameworks as well as purpose-built, task-specific testing tools tailored to the application and platforms under review.

Runtime Analysis

For relevant software targets identified during the course of this engagement, Atredis performed runtime analysis, using debugging and build tools to analyze application flow to aid in software security analysis. Where relevant, purpose-built tools such as fuzzers and customized network clients may have been utilized to aid in vulnerability identification.

Source Code Analysis

Atredis reviewed the in-scope application source code, with an eye for security-relevant software defects. To aid in vulnerability discovery, application components were mapped out and modeled until a thorough understanding of execution flow, code paths, and application design and architecture was obtained. To aid in this process, the assessment team engaged key stakeholders and members of the development team, where possible, to provide structured walkthroughs and interviews, helping the team rapidly gain an understanding of the application’s design and development lifecycle.

Executive Summary

As mentioned in the Engagement Overview section above, this engagement focused on four (4) new feature sets that have recently been added to the Polymesh system. These include External Agents permissions, changes to the settlements functionality to handle failed instructions, the ability for the owner of a key to pay for other's transactions, and ITN (Incentivized Testnet) rewards payout functionality. Most of this functionality was implemented in the External Agents, Settlements, Relayer, and Rewards pallets, respectively.

Testing for this engagement was primarily performed through unit testing in a mocked environment in order to carefully control environment and extrinsic parameters, then verified through interaction with local development mode nodes through the PolkadotJS UI and custom JavaScript test harnesses. The `v1_mainnet` branch of the public Polymesh repository was used as the target for the source code review and building nodes.

Authentication testing covered unauthenticated (un-signed) and authenticated (signed) extrinsic calls related to in-scope functionality. Authorization testing covered root (full chain owner), CDD (Customer Due Diligence) membership, and External Agents permissions (full, pallet, and extrinsic-level). Each of the in-scope components were also reviewed in-depth for input handling and business logic weakness that would result in security vulnerabilities.

Key Conclusions

Atredis found the External Agents permissions, Relayer, and Rewards functionality to be robustly implemented, well covered by unit tests, and to function as expected. No security vulnerabilities were identified in these components.

Two issues were identified in the Settlements functionality. The first allows disruption of settlement activities and the second is a business logic flaw that allows settlement instructions to execute outside of their date parameters. These issues are explained in detail in the Findings and Recommendations section, below.

Update: On September 27, 2021, Atredis performed remediation testing at the request of Polymath. Remediation status of each issue can be found in the Findings Detail table below. A short description can be found under the Remediation Status header in each issue.

Feature Overview

Polymesh is a blockchain technology focused on managing digital security tokens. Polymesh is built using the modular Substrate[1] framework for implementing vital components such as consensus and governance through its FRAME[2] (Framework for Runtime Aggregation of Modularized Entities) runtime system.

FRAME provides a framework for constructing runtime modules called pallets. Pallets are Rust modules of a defined structure that can be used to implement custom runtime logic. The Polymath team has authored custom pallets to implement new Polymesh features, some of which are under review for this assessment and listed below.

Settlements Pallet

The transferring of assets between accounts is handled in the Settlements pallet. This pallet implements logic to ensure all parties involved in a transaction have agreed to the terms of the trade prior to execution.

Previously, settlements which failed, due to reasons such as leg failure or the failing of compliance rules, required involved parties to create a new instruction manually. A new settlement feature has been implemented that allows for the rescheduling of failed instructions.

Failed instructions are now marked as “failed” during the instruction phase when a failure condition is met. The `reschedule_instruction()` dispatchable function handles the rescheduling of failed instructions and accepts the origin user and the instruction to be rescheduled.

Atredis notes that the permissions check performed during instruction rescheduling only validates that the origin user is allowed to perform the associated extrinsic calls. This behavior allows any valid system user to reschedule any failed instruction regardless of if they participated in any leg of the instruction. This permissive feature does not necessarily introduce a security issue, as all parties must still provide affirmations for the instruction to execute successfully.

The rescheduling mechanism was also analyzed for potential Denial of Service (DoS) attack vectors where a bad actor may cause resource exhaustion on the Polymesh node. This attack was found to be technically possible, but mitigated by financial deterrents. The attacker would have to pay fees for every failed instruction, making this attack very costly to sustain over a long period of time.

[1] https://substrate.dev/

[2] https://substrate.dev/docs/en/knowledgebase/runtime/frame

Aside from the rescheduling mechanism mentioned above, the Settlements Pallet was reviewed thoroughly for potential issues which may allow users to tamper with the settlement process. An issue was identified and reported in the finding, Settlements: Improper Permissions on Instruction Rejection, which allows a malicious user to rejection any pending instruction resulting in disruption of trading on the network.

All remaining extrinsic functions were verified to be using adequate permission checks contained within the Identity pallet. Most importantly, extrinsic functions which add instructions validate that the user created the venue by calling `venue_for_management()`. Additionally, when affirming instructions, proper checks are implemented to ensure the requesting user has custody of the associated portfolio using `ensure_portfolio_custody_and_permission()`.

Existing controls were also examined in order to ensure rules are properly enforced. It was found that when creating instructions, users can specify two timestamps to indicate a window of time where the instruction is valid to execute. These times are only subject to very basic input validation and are not enforced during instruction execution. This is noted in the finding titled, Settlements: Dates on Instructions Not Enforced.

External Agents Pallet

External Agents permissions are a new set of features that allow the owner of a specific Polymesh asset to delegate permissions over that asset to other Polymesh CDD members. Permissions can be created that allow users to execute all extrinsic calls, all extrinsic calls within a specific pallet, or only specific extrinsic calls for the associated asset.

At the time of this engagement, this worked by attaching an agent group to the asset, which defined the permissions granted to members of the group, then adding members to the group. In order to add members to a group, a user with the `addAuthorization` permission on the asset would first create a `BecomeAgent` authorization for the member to be added. Then the member to be added was required to accept the authorization. Group permissions could also be modified after users had been added to a group, members could be removed from a group, and members of a group could leave that group.

Atredis Partners assessed these features through code review, custom unit tests, a custom JavaScript test harness that utilized the `@polkadot/api` SDK, and interaction through the PolkadotJS UI. The code review included full coverage of the External Agents Pallet and essential coverage of supporting functionality, like bootstrap, identity, and transaction payment code. All implementations of the external agent permissions checks were also reviewed. Testing covered all permutations of permissions configurations and states.

When an extrinsic call was made, there were two phases of permissions checks that were performed. The first phase occurred in the custom Transaction Payment pallet before the actual extrinsic function was called. This phase validated that the caller has a valid identity, that the identity was a CDD member, that the identity's CDD claim was valid, and that the caller had enough funds to perform the call. The second phase occurred when the extrinsic call had been dispatched, and is where the External Agents permissions checks were performed.

Permissions in the second phase were primarily checked from within the extrinsic function by explicit calls to `frame_system::ensure_root`, `identity::ensure_perms`, or `external_agents::ensure_perms`. The `frame_system::ensure_root` function is a Substrate-provided function which validates that the call is being performed by the chain's root account, typically through the use of a `sudo:sudo` extrinsic call. The `identity::ensure_perms` function validates that the call is signed, that the caller has a valid identity, and that the identity has not been frozen. The `external_agents::ensure_perms` function performs all of the validation that the identity permissions check performs, and also validates that the caller has the correct permissions assigned to perform the call against the asset specified in the call.

In a few cases, direct calls were made to `external_agents::ensure_agent_asset_perms`, which is called by `ensure_perms` with the same parameters. The `external_agents::ensure_asset_perms` function was also used when no agent permission is checked, as in the `external_agents::abdicate` call, or when an optional agent permission check is performed, as in the `sto::stop` call.

In all but one case, the permissions controls and checks behaved as expected. The one exception was when agents were added to a group that was permitted to call the `external_agents::create_group` or `external_agents::set_group_permissions` extrinsic. Due to the fact that permissions apply to the ***asset*** that a group is created for, members with these permissions can affect all other groups attached to the same asset. While not a security vulnerability, asset owners should be made clearly aware of this behavior.

Also, it is important to highlight that while the External Agents permissions checks were reviewed extensively, many of the Polymesh pallets have their own business requirements and permissions that are not handled by the External Agents permissions checks. One example of this is explained in the finding, Settlements: Improper Permissions on Instruction Rejection. These types of checks were not reviewed in pallets that were not in scope for this engagement.

Relayer Pallet

The Relayer pallet allows for a user to pay for another user’s transaction fees in POLYX (Polymesh tokens). The user paying the fees is considered the “subsidizer” in this situation and will cover the fees associated with another user’s transaction.

This feature has been implemented for users who do not want to maintain a POLYX balance. These users can instead pay a national currency off-chain to an entity such as a brokerage, the brokerage then maintains a POLYX balance and covers the transaction fee cost.

The Relayer Pallet does not implement any features such as receipts to track or acknowledge off-chain transactions. The result is that off-chain transactions for POLYX fees are not viewable on Polymesh. The relationship between the subsidizer and the user receiving the subsidy is considered out-of-band and which may result in a lack of transparency.

Any system user may become a subsidizer for another user but only one subsidizer is permitted per account. When setting up a subsidizer relationship, a `paying_key` must first be assigned to the user being subsidized. This is accomplished by calling `Relayer::set_paying_key()` which registers an identity authorization for the account being subsidized and must be accepted by `Relayer::accept_paying_key()`. During `Relayer::accept_paying_key()`, any present `paying_key` value is removed and finally written in `<Subsidies<T>>::insert()`.

It is the responsibility of subsidizers to set limits on the amount on POLYX users may use to execute transactions. This limit is initialized in `Relayer::set_paying_key()` and can be updated by calling `Relayer::update_polyx_limit()`. Updating the POLX limit is protected from unauthorized amount increases from users other than the subsidizer in `ensure_is_paying_key()`.

Atredis reviewed processes implemented in the Relayer pallet and unit tests written to date to determine if new test cases may simulate new attacks not considered. The unit tests for the Relayer pallet were found to be comprehensive and covered the follow security related scenarios:

- Increasing subsidy amounts without subsidizer knowledge or approval
- Removal of paying keys by unauthorized accounts
- Updating subsidy amounts on accounts without any paying key present
- Integer overflow conditions when updating POLYX

Atredis Partners did not identify any security related issues related to the Relayer pallet.

Rewards Pallet

The rewards process allows users who participate in the ITN network to earn rewards that can be transferred into another account at a future date. When rewards are ready to be paid out, the root user must fund the `itn_rewards` treasury account and set the `ItnRewardStatus` for all accounts that will receive an award to `Unclaimed` with their award balance. The funding process can be done at genesis of a new chain or through the `balances::setBalance` and `rewards::setItnRewardStatus` extrinsic. When a user wants to claim their reward, they must send an un-signed `rewards::claimItnReward` call containing a signature field that contains `reward_address + "claim_itn_reward"`, signed by the account that owns `itn_address`.

The rewards claiming process was extensively reviewed for security weaknesses through a combination of code review, unit tests, a custom Node.js test harness that leveraged the Polkadot API, and the manual interaction with the Polkadot Apps interface.

Atredis Partners verified that the process first validates that the `ItnRewardStatus` for `in_address` is `Unclaimed`, then it reconstructs the expected signature and uses the Substrate-provided `verify` function to verify the signature. Once the parameters have been validated, the reward funds specified for `itn_address` plus 1 POLY[3] are transferred from the `itn_rewards` treasury account into the `reward_address`. Then the reward amount is bonded and the `ItnRewardStatus` for `itn_address` is set to `Claimed`. If the `itn_rewards` treasury account does not have enough funds to complete the transfer, the entire transaction fails and the `ItnRewardStatus` for `itn_address` is unchanged. No security issues were identified in the rewards functionality.

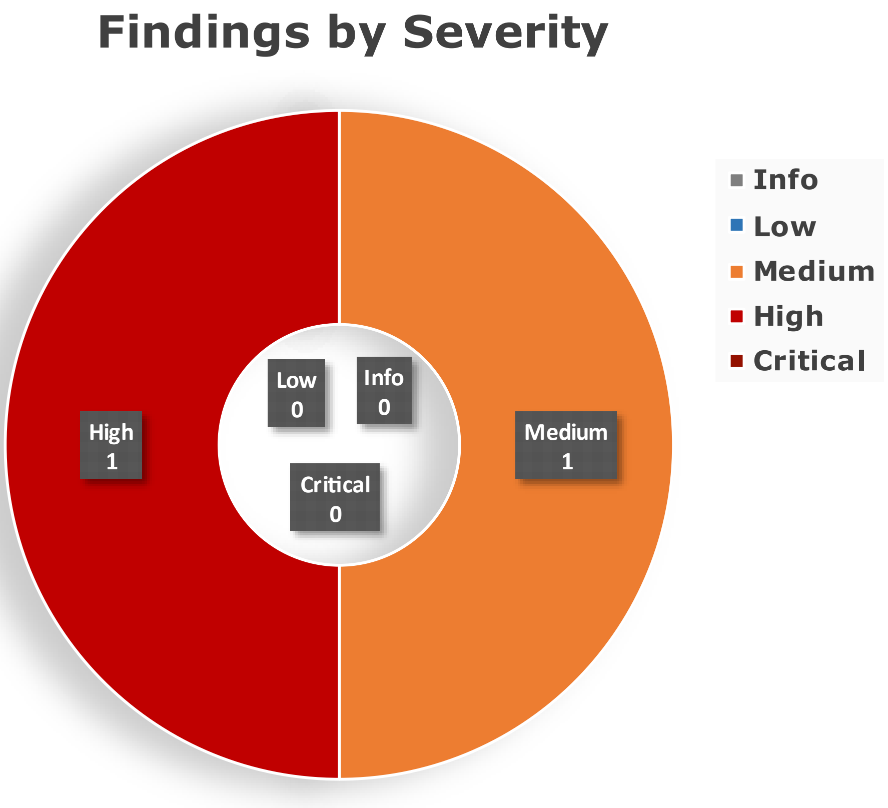
[3] https://blog.polymath.network/polyx-the-token-that-fuels-polymesh-ffd99175496b

Findings Summary

In performing testing for this assessment, Atredis Partners identified **one (1) high**, **one (1) medium** severity finding.

Atredis defines vulnerability severity ranking as follows:

- **Critical:** These vulnerabilities expose systems and applications to immediate threat of compromise by a dedicated or opportunistic attacker.
- **High:** These vulnerabilities entail greater effort for attackers to exploit and may result in successful network compromise within a relatively short time.
- **Medium:** These vulnerabilities may not lead to network compromise but could be leveraged by attackers to attack other systems or applications components or be chained together with multiple medium findings to constitute a successful compromise.
- **Low:** These vulnerabilities are largely concerned with improper disclosure of information and should be resolved. They may provide attackers with important information that could lead to additional attack vectors or lower the level of effort necessary to exploit a system.



Findings and Recommendations

The following section outlines findings identified via manual and automated testing over the course of this engagement. Where necessary, specific artifacts to validate or replicate issues are included, as well as Atredis Partners' views on finding severity and recommended remediation.

Findings Summary

The below tables summarize the number and severity of the unique issues identified throughout the engagement.

CRITICAL	HIGH	MEDIUM	LOW	INFO
0	1	1	0	0

Findings Detail

FINDING NAME	SEVERITY	REMEDIATION
Settlements: Improper Permissions on Instruction Rejection	High	Remediated
Settlements: Dates on Instructions Not Enforced	Medium	No Change

Settlements: Improper Permissions on Instruction Rejection

Severity: High, Remediated

Remediation Status

On September 27, 2021, Atredis reviewed the remediation that Polymath applied to the `develop` *branch of the public Polymesh repository. After reviewing the code changes and re-executing the unit test shown below, Atredis found the issue to be remediated.*

Finding Overview

The Settlement Pallet contains a dispatchable function allowing for the explicit rejection of pending instructions. It was found that any valid user account can reject any pending instruction regardless of the requesting user being involved in or having permission to the reject the instruction. Additionally, it may be possible to automate requests which reject all future instructions which would significantly disrupt trading on Polymesh.

Finding Detail

The `reject_instruction()` function was identified as only performing basic validation on the requesting origin account.

```
pub fn reject_instruction(origin, instruction_id: u64) {
    let PermissionedCallOriginData {
        primary_did,
        ..
    } = Identity::<T>::ensure_origin_call_permissions(origin)?;
    ensure!(
        Self::instruction_details(instruction_id).status != InstructionStatus::Unknown,
        Error::<T>::UnknownInstruction
    );
    let legs = InstructionLegs::iter_prefix(instruction_id).collect::<Vec<_>>();
    Self::unsafe_unclaim_receipts(instruction_id, &legs);
    Self::unchecked_release_locks(instruction_id, &legs);
    let _ = T::Scheduler::cancel_named((SETTLEMENT_INSTRUCTION_EXECUTION, instruction_id).e
ncode());
    Self::prune_instruction(instruction_id);
    Self::deposit_event(RawEvent::InstructionRejected(primary_did, instruction_id));
}
```

pallets/settlement/src/lib.rs:686

The `ensure_origin_call_permissions(origin)` function only verifies the origin is valid and that they have permission to make the associated extrinsic call. `ensure_origin_call_permissions(origin)` does not accept the instruction identifier as an argument to validate if the origin has permission to modify the instruction.

```
pub fn ensure_origin_call_permissions(
    origin: <T as frame_system::Config>::Origin,
) -> Result<PermissionedCallOriginData<T::AccountId>, DispatchError> {
    let sender = ensure_signed(origin)?;
    let AccountCallPermissionsData {
        primary_did,
        secondary_key,
    } = CallPermissions::<T>::ensure_call_permissions(&sender)?;
    Ok(PermissionedCallOriginData {
        sender,
        primary_did,
        secondary_key,
    })
}
```

pallets/identity/src/lib.rs:1970

A unit test was written to confirm the issue. The unit test sets up a two legged instruction between users `Alice` and `Bob`, the instruction is then affirmed by both parties. Before the block increments and instructions are executed, user `Ferdie` successfully rejects the instruction:

```
[SNIPPED for brevity]

println!("Instruction status: {:?}", Settlement::instruction_details(instruction_counter).s
tatus);
println!("Ferdie rejects instruction");
// Ferdie rejects instruction
assert_ok!(
    Settlement::reject_instruction(
        ferdie.origin(),
        instruction_counter
    )
);
println!("Instruction status: {:?}", Settlement::instruction_details(instruction_counter).s
tatus);

[SNIPPED for brevity]
```

Unit test snippet

When the unit test is run, the instruction is successfully displayed as rejected.

```
# cargo test --package polymesh-runtime-tests settlement_test::reject_perms -- --nocapture
running 1 test
Instruction status: Pending
Ferdie rejects instruction
Instruction status: Unknown
```

Unit test output

Rejected instructions are marked with an unknown status so they cannot not be rescheduled in the future. Affirmations are removed and locked assets are released forcing the instruction to be completed again manually.

Recommendation(s)

Consider performing an additional validation check to ensure users requesting to reject instructions have permission to do so. This may require the user or their designated external agents to have a stake in at least one leg of the instruction being rejected.

References

CWE-732: Incorrect Permission Assignment for Critical Resource:
https://cwe.mitre.org/data/definitions/732.html

Settlements: Dates on Instructions Not Enforced

Severity: Medium

Remediation Status

Polymath notes that the instruction dates logic is implemented this way by design, and that it is noted in documentation that these dates are not enforced.

Finding Overview

Dates may optionally be added to instructions as part of the settlement process. The `trade_date` and `value_date` may optionally be set to allow for a defined period in which the instruction may be executed. It was found that neither of the date parameters are enforced. As a result, instructions may execute prior to the `trade_date` and never expire regardless of the `value_date` specified when adding instructions.

Publicly available documentation (see References section below) suggests expiry dates are enforced. Client applications may implement optional expiry dates which give the end user a false sense that expire dates are enforced when making financial decisions.

Finding Detail

The Settlements Pallet contains the `add_instruction()` dispatch function which is called when adding instructions for later execution. The caller provides various arguments to `add_instruction()` including the `trade_date` and `value_date` which specify a valid window of time where the instruction can be interacted with and executed.

```
/// Adds a new instruction.
///
/// # Arguments
/// * `venue_id` - ID of the venue this instruction belongs to.
/// * `settlement_type` - Defines if the instruction should be settled
///    in the next block after receiving all affirmations or waiting till a specific block.
/// * `trade_date` - Optional date from which people can interact with this instruction.
/// * `value_date` - Optional date after which the instruction should be settled (not enfor
ced)
/// * `legs` - Legs included in this instruction.
///
/// # Weight
/// `950_000_000 + 1_000_000 * legs.len()`
pub fn add_instruction(
    origin,
    venue_id: u64,
    settlement_type: SettlementType<T::BlockNumber>,
    trade_date: Option<T::Moment>,
    value_date: Option<T::Moment>,
    legs: Vec<Leg>,
) {
    let did = Identity::<T>::ensure_perms(origin)?;
    Self::base_add_instruction(did, venue_id, settlement_type, trade_date, value_date, leg
s)?;
}
```

`pallets/settlement/src/lib.rs:599`

To demonstrate the issue, a unit test was created with the `trade_date` and `value_date` timestamps being set the current time. The test thread was also delayed to ensure instruction expiry:

```
let current_time = u64::try_from(Utc::now().timestamp()).unwrap();
println!("Current TimeStamp in ms: {:?}", current_time);

// add expired instruction
assert_ok!(Settlement::add_instruction(
    alice.origin(),
    venue_counter,
    SettlementType::SettleOnBlock(block_number),
    Some(current_time),
    Some(current_time),
    legs.clone()
));

let ten_seconds = time::Duration::from_millis(1000 * 10);
thread::sleep(ten_seconds);
```

`value_date` unit test snippet

Running the unit test above shows the instruction is successfully executed regardless of `value_date`:

```
running 1 test
Current TimeStamp in ms: 1631309872
Instruction status prior to execution: Pending
Instruction status after execution: Unknown
test settlement_test::settle_on_block_expired ... ok

test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured; 465 filtered out; finished in 1
0.75s
```

value_date unit test results

Reviewing the source code for the `Instruction` struct, commentary was observed suggesting the `value_date` is not currently enforced:

```
/// Details about an instruction
#[derive(Encode, Decode, Default, Clone, PartialEq, Eq, Debug, PartialOrd, Ord)]
pub struct Instruction<Moment, BlockNumber> {
    /// Unique instruction id. It is an auto incrementing number
    pub instruction_id: u64,
    /// Id of the venue this instruction belongs to
    pub venue_id: u64,
    /// Status of the instruction
    pub status: InstructionStatus,
    /// Type of settlement used for this instruction
    pub settlement_type: SettlementType<BlockNumber>,
    /// Date at which this instruction was created
    pub created_at: Option<Moment>,
    /// Date from which this instruction is valid
    pub trade_date: Option<Moment>,
    /// Date after which the instruction should be settled (not enforced)
    pub value_date: Option<Moment>,
}
```

pallets/settlement/src/lib.rs:205

A unit test was also created in order to validate that the `trade_date` is being enforced. This was accomplished by creating an instruction to execute in the year 2040.

```
// add instruction not be valid till 2040
assert_ok!(Settlement::add_instruction(
    alice.origin(),
    venue_counter,
    SettlementType::SettleOnBlock(block_number),
    Some(2206300411),
    None,
    legs.clone()
));
```

trade_date unit test snippet

Running the unit test above shows the instruction is successfully executed regardless of the `trade_date`:

```
running 1 test
Current TimeStamp in ms: 1631308503
Instruction status prior to execution: Pending
Instruction status after execution: Unknown
test settlement_test::settle_on_block_future_instruction ... ok
```

`trade_date` unit test results

Recommendation(s)

Consider adding additional logic needed to enforce both the `trade_date` and `value_date` parameters. `value_date` and `trade_date` need to be enforced during instruction execution. This will likely be in the `execute_instruction()` function which validates various conditions such as ensuring the instruction has no pending affirmations. Instructions should fail execution when `current_time < trade_date` or `current_time > value_date`. Failed instructions should trigger the `InstructionFailed` event or a custom event and be treated as a failed instruction which may optionally be rescheduled will a new expiry time set in the future.

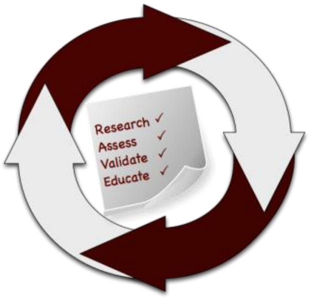
References

CWE-672: Operation on a Resource after Expiration or Release
https://cwe.mitre.org/data/definitions/672.html

“A party can also not affirm a leg, i.e. play for time, which would leave the instruction in limbo until some other resolution, like a cancellation or the instruction's expiry date being reached.”
https://developers.polymesh.live/settlement/settlement-process

“Authorization can only be provided before the instructions expiry data, and after its valid_from date.” and “If an instruction fails, is rejected or expires, all asset locks are removed (and receipts marked as unused).”
https://developers.polymesh.live/polymesh-docs/primitives/settlement

Appendix I: Assessment Methodology



Atredis Partners draws on our extensive experience in penetration testing, reverse engineering, hardware/software exploitation, and embedded systems design to tailor each assessment to the specific targets, attacker profile, and threat scenarios relevant to our client's business drivers and agreed upon rules of engagement.

Where applicable, we also draw on and reference specific industry best practices, regulations, and principles of sound systems and software design to help our clients improve their products while simultaneously making them more stable and secure.

Our team takes guidance from industry-wide standards and practices such as the National Institute of Standards and Technology's (NIST) Special Publications, the Open Web Application Security Project (OWASP), and the Center for Internet Security (CIS).

Throughout the engagement, we communicate findings as they are identified and validated, and schedule ongoing engagement meetings and touchpoints, keeping our process open and transparent and working closely with our clients to focus testing efforts where they provide the most value.

In most engagements, our primary focus is on creating purpose-built test suites and toolchains to evaluate the target, but we do utilize off-the-shelf tools where applicable as well, both for general patch audit and best practice validation as well as to ensure a comprehensive and consistent baseline is obtained.

Research and Profiling Phase

Our research-driven approach to testing begins with a detailed examination of the target, where we model the behavior of the application, network, and software components in their default state. We map out hosts and network services, patch levels, and application versions. We frequently use a number of private and public data sources to collect Open Source Intelligence about the target, and collaborate with client personnel to further inform our testing objectives.

For network and web application assessments, we perform network and host discovery as well as map out all available application interfaces and inputs. For hardware assessments, we study the design and implementation, down to a circuit-debugging level. In reviewing source code or compiled application code, we map out application flow and call trees and develop a solid working understanding of how the application behaves, thus helping focus our validation and testing efforts on areas where vulnerabilities might have the highest impact to the application's security or integrity.

Analysis and Instrumentation Phase

Once we have developed a thorough understanding of the target, we use a number of specialized and custom-developed tools to perform vulnerability discovery as well as binary, protocol, and runtime analysis, frequently creating engagement-specific software tools which we share with our clients at the close of any engagement.

We identify and implement means to monitor and instrument the behavior of the target, utilizing debugging, decompilation and runtime analysis, as well as making use of memory and filesystem forensics analysis to create a comprehensive attack modeling testbed. Where they exist, we also use common off-the-shelf, open-source and any extant vendor-proprietary tools to aid in testing and evaluation.

Validation and Attack Phase

Using our understanding of the target, our team creates a series of highly-specific attack and fault injection test cases and scenarios. Our selection of test cases and testing viewpoints are based on our understanding of which approaches are most relevant to the target and will gain results in the most efficient manner, and built in collaboration with our client during the engagement.

Once our test cases are validated and specific attacks are confirmed, we create proof-of-concept artifacts and pursue confirmed attacks to identify extent of potential damage, risk to the environment, and reliability of each attack scenario. We also gather all the necessary data to confirm vulnerabilities identified and work to identify and document specific root causes and all relevant instances in software, hardware, or firmware where a given issue exists.

Education and Evidentiary Phase

At the conclusion of active testing, our team gathers all raw data, relevant custom toolchains, and applicable testing artifacts, parses and normalizes these results, and presents an initial findings brief to our clients, so that remediation can begin while a more formal document is created. Additionally, our team shares confirmed high-risk findings throughout the engagement so that our clients may begin to address any critical issues as soon as they are identified.

After the outbrief and initial findings review, we develop a detailed research deliverable report that provides not only our findings and recommendations but also an open and transparent narrative about our testing process, observations and specific challenges in developing attacks against our targets, from the real world perspective of a skilled, motivated attacker.

Automation and Off-The-Shelf Tools

Where applicable or useful, our team does utilize licensed and open-source software to aid us throughout the evaluation process. These tools and their output are considered secondary to manual human analysis, but nonetheless provide a valuable secondary source of data, after careful validation and reduction of false positives.

For runtime analysis and debugging, we rely extensively on Hopper, IDA Pro and Hex-Rays, as well as platform-specific runtime debuggers, and develop fuzzing, memory analysis, and other testing tools primarily in Ruby and Python.

In source auditing, we typically work in Visual Studio, Xcode and Eclipse IDE, as well as other markup tools. For automated source code analysis we will typically use the most appropriate toolchain for the target, unless client preference dictates another tool.

Network discovery and exploitation make use of Nessus, Metasploit, and other open-source scanning tools, again deferring to client preference where applicable. Web application runtime analysis relies extensively on the Burp Suite, Fuzzer and Scanner, as well as purpose-built automation tools built in Go, Ruby and Python.

Engagement Deliverables

Atredis Partners deliverables include a detailed overview of testing steps and testing dates, as well as our understanding of the specific risk profile developed from performing the objectives of the given engagement.

In the engagement summary we focus on “big picture” recommendations and a high-level overview of shared attributes of vulnerabilities identified and organizational-level recommendations that might address these findings.

In the findings section of the document, we provide detailed information about vulnerabilities identified, provide relevant steps and proof-of-concept code to replicate these findings, and our recommended approach to remediate the issues, developing these recommendations collaboratively with our clients before finalization of the document.

Our team typically makes use of both DREAD and NIST CVE for risk scoring and naming, but as part of our charter as a client-driven and collaborative consultancy, we can vary our scoring model to a given client’s preferred risk model, and in many cases will create our findings using the client’s internal findings templates, if requested.

Sample deliverables can be provided upon request, but due to the highly specific and confidential nature of Atredis Partners’ work, these deliverables will be heavily sanitized, and give only a very general sense of the document structure.

Appendix II: Engagement Team Biographies

Shawn Moyer, Founding Partner and CEO

Shawn Moyer scopes, plans, and coordinates security research and consulting projects for the Atredis Partners team, including reverse engineering, binary analysis, advanced penetration testing, and private vulnerability research. As CEO, Shawn works with the Atredis leadership team to build and grow the Atredis culture, making Atredis Partners a home for some of the best minds in information security, and ensuring Atredis continues to deliver research and consulting services that exceed our client's expectations.

Experience

Shawn brings over 25 years of experience in information security, with an extensive background in penetration testing, advanced security research including extensive work in mobile and Smart Grid security, as well as advanced threat modeling and embedded reverse engineering.

Shawn has served as a team lead and consultant in enterprise security for numerous large initiatives in the financial sector and the federal government, including IBM Internet Security Systems' X-Force, MasterCard, a large Federal agency, and Wells Fargo Securities, all focusing on emerging network and application attacks and defenses.

In 2010, Shawn created Accuvant Labs' Applied Research practice, delivering advanced research-driven consulting to numerous clients on mobile platforms, critical infrastructure, medical devices and countless other targets, growing the practice 1800% in its first year.

Prior to Accuvant, Shawn helped develop FishNet Security's penetration testing team as a principal security consultant, growing red team offerings and advanced penetration testing services, while being twice selected as a consulting MVP.

Key Accomplishments

Shawn has written on emerging threats and other topics for Information Security Magazine and ZDNet, and his research has been featured in the Washington Post, BusinessWeek, NPR and the New York Times. Shawn is a twelve-time speaker at the Black Hat Briefings and has been an invited speaker at other notable security conferences around the world.

Shawn is likely best known for delivering the first public research on social network security, pointing out much of the threat landscape still exists on social network platforms today. Shawn also co-authored an analysis of the state of the art in web browser exploit mitigation, creating the first in-depth comparison of browser security models along with Dr. Charlie Miller, Chris Valasek, Ryan Smith, Joshua Drake, and Paul Mehta.

Shawn studied Computer and Network Information Systems at Missouri University and the University of Louisiana at Lafayette, holds numerous information security certifications, and has been a frequent presenter at national and international security industry conferences.

Josh Thomas, Founding Partner and COO

Josh Thomas's specialties include advanced hardware and software reverse engineering, malware and rootkit development and discovery, and software development. Josh has extensive experience in developing secure solutions for mobile platforms and a deep understanding of cellular architecture. Josh previously held a TS clearance, and has worked in many sensitive, cleared environments.

Experience

Josh began his career 14 years ago in network administration and software development. Prior to moving his focus primarily to security, Josh wrote Artificial Intelligence and cryptographic solutions for the Department of Defense. Josh has extensive hands on knowledge of mobile devices and cellular infrastructure. He is also dedicated to hardware reverse engineering and embedded device exploitation.

Josh most recently was a Senior Research Scientist with Accuvant's Applied Research team and has worked as a Senior Research Developer at The MITRE Corporation. At MITRE, Josh performed analyses of the Android, Apple, Symbian and BlackBerry security models as well as other non-mobile embedded platforms and worked closely with the vendors and project sponsors.

Josh also developed an open-source mesh networking solution for Smart phone communications that bypasses the need for physical infrastructure, performed advanced spectrum analysis for cleared communications, and designed a secure satellite communications system required to handle the most sensitive communications possible while also being resilient against the highest levels of waveform interference.

Prior to his tenure at The MITRE Corporation, Josh developed Artificial Intelligence and embedded cryptographic solutions for General Dynamics and other organizations. Josh projects including the design and development of robust routing architecture for UAV/UGV autonomous vehicles, and battlefield troop movement predictive scenario generation.

Key Accomplishments

Josh is the recipient of three DARPA Cyber Fast Track grants for advanced security research, and has presented at multiple security industry conferences, including BlackHat, DEF CON, DerbyCon and ToorCon. Josh is the lead developer and maintainer of the open-source SPAN mesh networking project for Android, has published and reviewed papers for IEEE, and holds a pending patent related to NAND flash memory hiding techniques.

Josh holds a bachelor's in Computer Science from Texas A&M University and has been a frequent presenter at national and international security industry conferences.

Bryan C. Geraghty, Principal Research Consultant

Bryan leads and executes highly technical application and network security assessments, as well as adversarial simulation assessments. He specializes in cryptography and reverse engineering.

Experience

Bryan has over 20 years of experience building and exploiting networks, software, and hardware systems. His deep background in systems administration, software development, and cryptography has been demonstrably beneficial for security assessments of custom or unique applications in industries such as healthcare, manufacturing, marketing, banking, utilities, and entertainment.

Key Accomplishments

Bryan is a creator and maintainer of several open-source security tools. He is also a nationally recognized speaker; often presenting research on topics such as software, hardware, and communications protocol attacks, and participating in offense-oriented panel discussions. Bryan is also an organizing-board member of multiple Kansas City security events, and a staff volunteer & organizer of official events at DEF CON.

Loren Browman, Senior Research Consultant

Loren Browman has over 10 years of experience in both consulting and federal law enforcement environments. His experiences range from deep security research in federal government to product and application testing for Fortune 500 corporations. Loren is a recognized subject matter expert (SME) in securing IoT products and advanced hardware testing methodology. Areas of expertise include reverse engineering of hardware, firmware, and communication protocols.

Experience

Loren has conducted numerous large scale product security assessments including challenging black box security assessments and secure design reviews.

Prior to joining Atredis, Loren was an operations supervisor and security researcher for the Royal Canadian Mounted Police (RCMP). This role included providing technical expertise to support police investigations and leading security research efforts in order to circumvent security mechanisms and develop deployable capabilities.

Key Accomplishments

Loren has developed numerous tools for accelerating research on a wide range of products. This includes the development of a fuzzing suite for automotive Electronic Control Units over CAN bus vehicle networks, this led to the discovery of multiple hidden services and exploits. More recently, Loren published nrfsec, a tool for automating firmware recovery vulnerability on secured nrf51 System on Chips.

Loren has studied Electrical and Computer Engineering at the British Columbia Institute of Technology and has attended various specialized training sessions including the Arm IoT Exploit Laboratory, Power Analysis and Glitching and is an Offensive Security Certified Professional (OSCP).

Molly Vukusich, Client Operations Associate

Molly Vukusich supports nearly every phase of the project lifecycle at Atredis Partners, from pre-sales, to project planning and management, to project delivery, readout and follow-up. She aims to increase efficiency of project execution and client communication for the benefit of both the consultants and clients.

Experience

Molly has over 11 years of experience in marketing and project management roles in various industries such as Healthcare, Finance, Sports & Recreation, and Non-Profit. Her experience includes copywriting and editing (both technical and promotional), creative strategy development, data analysis, event planning, graphic design, and website management.

Key Accomplishments

Molly earned a bachelor’s degree in Mass Communications with an emphasis in Advertising and Public Relations from Oklahoma City University.

Sean Bradly, Principal Research Consultant

Sean Bradly is an expert security researcher with 20 years of experience in general software development and nearly 15 years with a focus on security. He has spent many of these years researching, auditing, reverse engineering, exploiting, designing, implementing, maintaining, and delivering both software and hardware pertaining to all manner of subject matter.

Experience

Sean has held many roles within the industry, starting in the year 2000 as a junior programmer and quickly moving into other realms such as systems automation, embedded development, security engineering, and security consulting.

Sean got his start in computer security while developing an automated network vulnerability scanning service (TrustWatch) in 2006. He then went on to BreakingPoint Systems where he designed network testing software, writing network protocol simulators and exploit traffic generators with focus on supporting both realistic and fuzzed test cases. He also extensively researched security vulnerabilities by hunting for undiscovered bugs, scouring publicly available information, and frequently reverse engineering vendor software update files to craft new exploits for inclusion into the product.

In addition, Sean held the position of Senior Security Consultant with Leviathan Security Group, frequently leading audits on everything from embedded device firmware to web applications as well as building tools to automate analysis and better identify potential security issues.

Most recently before joining Atredis, Sean was a partner at Inverse Limit (InvLim), working on aggressively-paced research and development contracts with clients such as DARPA and Google.

Key Accomplishments

In 2013, Sean authored a custom hypervisor and analysis engine for Project MAIM (part of a DARPA Cyber Fast-Track research grant) to study the differences in CPU instruction sets of different vendors' implementations.

In 2015, Sean designed and implemented an open source, cross platform (ARM/OpenRISC), embedded operating system written from scratch to host sensitive cryptographic applications. In six months, Inverse Limit's team of four were able to deliver custom circuit board with a custom OpenRISC CPU that included accelerated cryptography, the bespoke operating system, and demo applications. This (along with the hardware and other components) was all open sourced as Google's Project Vault and was presented by Pieter Zatko at Google I/O.

Sean also designed and implemented a custom TCP/IP protocol stack for BreakingPoint Systems' Security Engine in order to audit network appliances by realistically simulating attack traffic from tens of thousands of exploits.

Lacey Kasten, Client Operations, Technical Writer and Editor

Lacey Kasten helps facilitate client operations and deliverable creation/development at Atredis Partners. She supports pre-sales project scoping and back-end operations efforts, shepherding of the technical writing style and voice at Atredis, and the final quality assurance review of penetration test deliverables prior to engagement completion. Lacey seeks to provide readable, understandable communication to Atredis Partners' clientele. She stays embedded in the Information Security community and is passionate about accessible and equitable knowledge transfer in all mediums across a wide span of Cyber Security, Threat Intelligence, National Security, and Open Source topics.

Experience

Lacey has worked in communications roles from within the Fine Art and Design industry, Museum and Non-Profit Philanthropy space, Biomedical Computer Science, Higher Education Public Relations, and Event and Tradeshow industry throughout her career. Her work spans writing (technical, copy editing, social media marketing, and blogging), editing and mentorship of writers in the Information Security space, content creation (web development, event planning, graphic design, and photography), and film and movie production.

Key Accomplishments

Lacey achieved a bachelor's degree in Communication Design from the Pacific Northwest College of Art in Portland, Oregon. Lacey has contributed to Open Source Intelligence (OSINT) tool projects, notably Chanscan, and participated in beta testing, documentation creation, and project management for other Open Source development projects. Currently, Lacey supports the FLOSS (Free/Libre/Open Source Software) community by serving on the core organizing staff of SeaGL (Seattle GNU/Linux) conference. Previously, she was a member on the Board of Directors for the largest Information Security conference in the United States Pacific Northwest, Security BSides PDX, and served the charitable 501(c)(3) as coordinator of Sponsorship and Endowment.

Appendix III: About Atredis Partners

Atredis Partners was created in 2013 by a team of security industry veterans who wanted to prioritize offering quality and client needs over the pressure to grow rapidly at the expense of delivery and execution. We wanted to build something better, for the long haul.

In six years, Atredis Partners has doubled in size annually, and has been named three times to the Saint Louis Business Journal's "Fifty Fastest Growing Companies" and "Ten Fastest Growing Tech Companies". Consecutively for the past three years, Atredis Partners has been listed on the Inc. 5,000 list of fastest growing private companies in the United States.

The Atredis team is made up of some of the greatest minds in Information Security research and penetration testing, and we've built our business on a reputation for delivering deeper, more advanced assessments than any other firm in our industry.

Atredis Partners team members have presented research over forty times at the BlackHat Briefings conference in Europe, Japan, and the United States, as well as many other notable security conferences, including RSA, ShmooCon, DerbyCon, BSides, and PacSec/CanSec. Most of our team hold one or more advanced degrees in Computer Science or engineering, as well as many other industry certifications and designations. Atredis team members have authored several books, including *The Android Hacker's Handbook*, *The iOS Hacker's Handbook*, *Wicked Cool Shell Scripts*, *Gray Hat C#*, and *Black Hat Go*.

While our client base is by definition confidential and we often operate under strict nondisclosure agreements, Atredis Partners has delivered notable public security research on improving the security at Google, Microsoft, The Linux Foundation, Motorola, Samsung and HTC products, and were the first security research firm to be named in Qualcomm's Product Security Hall of Fame. We've received four research grants from the Defense Advanced Research Project Agency (DARPA), participated in research for the CNCF (Cloud Native Computing Foundation) to advance the security of Kubernetes, worked with OSTIF (The Open Source Technology Improvement Fund) and The Linux Foundation on the Core Infrastructure Initiative to improve the security and safety of the Linux Kernel, and have identified entirely new classes of vulnerabilities in hardware, software, and the infrastructure of the World Wide Web.

In 2015, we expanded our services portfolio to include a wide range of advanced risk and security program management consulting, expanding our services reach to extend from the technical trenches into the boardroom. The Atredis Risk and Advisory team has extensive experience building mature security programs, performing risk and readiness assessments, and serving as trusted partners to our clients to ensure the right people are making informed decisions about risk and risk management.