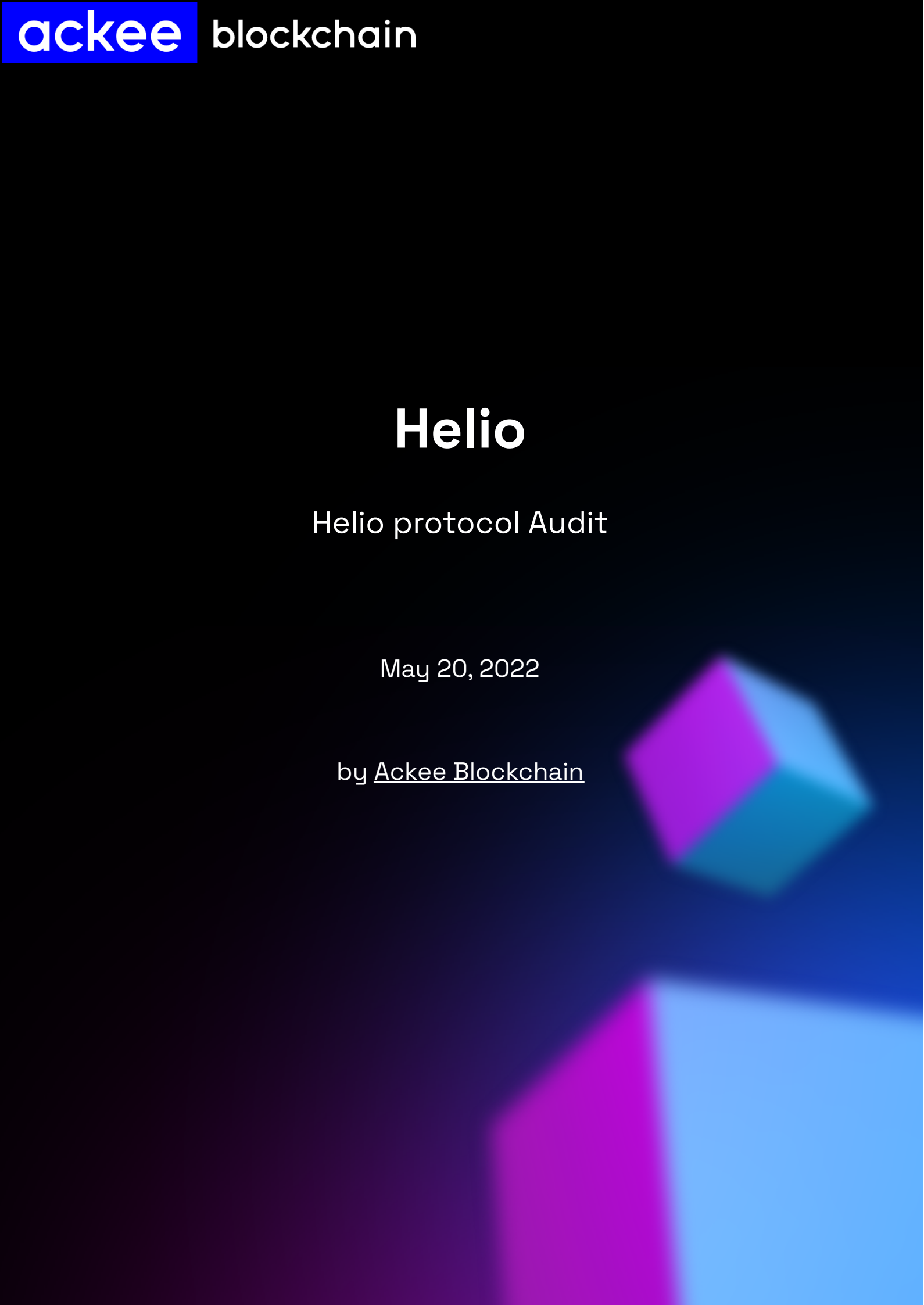
ackee blockchain

Helio

Helio protocol Audit

May 20, 2022

by Ackee Blockchain



Contents

1. Document Revisions

0.1	Critical issue report, pre-audit version	May 18, 2022
1.0	Final report	May 20, 2022
1.1	Fix review	Jul 22, 2022

2. Overview

This document presents our findings in reviewed contracts.

2.1. Ackee Blockchain

Ackee Blockchain is an auditing company based in Prague, Czech Republic, specializing in audits and security assessments. Our mission is to build a stronger blockchain community by sharing knowledge – we run a free certification courses Winter School of Solana, Summer School of Solidity and teach at the Czech Technical University in Prague. Ackee Blockchain is backed by the largest VC fund focused on blockchain and DeFi in Europe, Rockaway Blockchain Fund.

2.2. Audit Methodology

The Ackee Blockchain auditing process follows a routine series of steps:

1. Code review
    a. High-level review of the specifications, sources, and instructions provided to us to make sure we understand the project's size, scope, and functionality.
    b. Detailed manual code review, which is the process of reading the source code line-by-line to identify potential vulnerabilities. We focus mainly on common classes of Solana program vulnerabilities, such as:

       missing ownership checks, missing signer authorization, signed CPI of unverified programs, cosplay of Solana accounts, missing rent exemption assertion, bump seed canonicalization, incorrect accounts closing, casting truncation, numerical precision errors, arithmetic overflows or underflows, ...

   c. Comparison of the code and given specifications, ensuring that the program logic correctly implements everything intended.
   d. Review of best practices to improve efficiency, clarity, and maintainability.

2. Testing and automated analysis
   a. Run client's tests to ensure that the system works as expected, potentially write missing unit or fuzzy tests using our testing framework Trdelnik.

3. Local deployment + hacking
   a. The programs are deployed locally, and we try to attack the system and break it. There is no specific strategy here, and each project's attack attempts are characteristic of each program audited. However, when trying to attack, we rely on the information gained from previous steps and our rich experience.

2.3. Review team

Member's Name	Position
Vladimír Marcin	Lead Auditor
Tibor Tribus	Auditor
Štefan Prokop	Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

2.4. Disclaimer

We've put our best effort into finding all vulnerabilities in the system; however, our findings shouldn't be considered a complete list of all existing issues. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

3. Executive Summary

Helio engaged Ackee Blockchain to conduct a security review of their protocol with a total time of 5 engineering days. The review took place between 16. May and 20. May 2022.

The scope included the following repository with a given commit:

- **Repository:** helio-protocol
- **Commit:** `8a6b1a20551cde8cff68d55e43baa5524692e82c`

The beginnig of the audit was dedicated to understanding the entire Helio platform. We then took a deep dive into the `helio-protocol` program. During the review, we paid particular attention to:

- Is the correctness of the program ensured (does it correctly implement the project goals)?
- Do the program correctly use dependencies or other programs they rely on (e.g., SPL dependencies)?
- Is the code vulnerable to economic attacks?

Our review resulted in 8 issues, ranging from *`Informational`* to *`Critical`* severity. Four of these issues were critical, causing either the lockup of assets or the possibility of stealing them.

The issues C1 and C2 were reported immediately after the discovery in the separate revision of this document (pre-audit version 0.1), even though the Helio protocol is not live yet.

AckeeBlockchain recommends Helio:

- to address all reported issues,

- another full audit once the issues are fixed (the reason being that we have devoted much time to exploits due to many critical issues),
- to follow Rust and Solana's best practices, some of which you can find in Appendix A.

Update July 22, 2022: Helio provided an updated codebase that addresses issues from this report. See Appendix C for a detailed discussion of the exact status of each issue. During the fix review process, we found additional three informational issues I3, I4, and I5.

4. System Overview

This section contains an outline of the audited programs. Note that this is meant for understandability purposes and does not replace project documentation.

4.1. Programs

Helio protocol program

Solana program for settling payments. It enables one time payments between sender and recipient with possibility of split payments which supports spliting total payed amount between many recipients. The contract contains logic for creating continuous money streams between two parties. Recipient can withdraw due amount of tokens any time during the stream duration or after if stream is not cancelled. When cancelling stream the due amount of tokens is transferred to the recipient, remain is refunded to the sender as well as rent of the payment metadata account. The payments can be performed in SOL or any SPL token.

4.2. Actors

This part describes actors of the system, their roles, and permissions.

Sender

A sender can create a one-time payment to one or multiple recipients. A sender also can create/cancel a pay stream.

Recipient

Recipient can withdraw due amount of tokens any time during the stream duration or after if stream is not cancelled.

4.3. Trust model

Sent assets are entirely under the program's control, so neither the sender nor the recipient must trust any third party.

5. Vulnerabilities risk methodology

Each finding contains an *Impact* and *Likelihood* ratings.

If we have found a scenario in which the issue is exploitable, it will be assigned an impact of *High*, *Medium*, or *Low*, based on the direness of the consequences it has on the system. If we haven't found a way, or the issue is only exploitable given a change in configuration (such as deployment scripts, compiler configuration, use of multi-signature wallets for owners, etc.) or given a change in the codebase, then it will be assigned an impact rating of *Warning* or *Informational*.

Low to *High* impact issues also have a *Likelihood* which measures the probability of exploitability during runtime.

5.1. Finding classification

The full definitions are as follows:

Impact

- **High** - Code that activates the issue will lead to undefined or catastrophic consequences for the system.
- **Medium** - Code that activates the issue will result in consequences of serious substance.
- **Low** - Code that activates the issue will have outcomes on the system that are either recoverable or don't jeopardize its regular functioning.
- **Warning** - The issue cannot be exploited given the current code and/or configuration (such as deployment scripts, compiler configuration, use of multi-signature wallets for owners, etc.), but could be a security vulnerability if these were to change slightly. If we haven't found a way to exploit the issue given the time constraints, it might be marked as

"Warning" or higher, based on our best estimate of whether it is currently exploitable.

- **Informational** - The issue is on the border-line between code quality and security. Examples include insufficient logging for critical operations. Another example is that the issue would be security-related if code or configuration (see above) was to change.

Likelihood

- **High** - The issue is exploitable by virtually anyone under virtually any circumstance.
- **Medium** - Exploiting the issue currently requires non-trivial preconditions.
- **Low** - Exploiting the issue requires strict preconditions.

6. Findings

This section contains the list of discovered findings. Unless overriden for purposes of readability, each finding contains:

- a *Description*,
- an *Exploit scenario*, and
- a *Recommendation*

Many times, there might be multiple ways to solve or alleviate the issue, with varying requirements in terms of the necessary changes to the codebase. In that case, we will try to enumerate them all, making clear which solve the underlying issue better (albeit possibly only with architectural changes) than others.

Summary of Findings

	Type	Impact	Likelihood
C1: `withdraw_payment` and `cancel_payment` instructions will not work after the pay stream ends	Arithmetic	High	High
C2: Possibility of stealing tokens from escrow token account	PDA sharing	High	High
C3: Possibility of stuck tokens	Arithmetic	High	High

	Type	Impact	Likelihood
C4: Using the same struct for SOL payments as for token payments results in the possibility of a tokens lock attack	Type cosplay	High	High
M1: Hanging `payment_token_account(s)`	Accounts management	Low	High
L1: Using `find_program_address` instead of `create_program_address`	Compute budget	Low	Low
I1: `PaymentAccount` struct has unused fields	Code smell	Informational	N/A
I2: Unnecessary mutable modifier	Code smell	Informational	N/A

Table 1. Table of Findings

C1: `withdraw_payment` and `cancel_payment` instructions will not work after the pay stream ends

Impact:	High	Likelihood:	High
Target:	payment.rs	Type:	Arithmetic

Description

The `amount_available` method, in each case, adds to the number of intervals number one (even in cases where the division has no remainder). It may, in some cases, lead to a situation where the number of intervals is higher than it actually is. The result could be that the recipient will not be able to claim tokens for the services offered.

Exploit scenario

1. Suppose the pay stream creation with the following parameters:

```
amount = 1000 // tokens
start_at = 1652789479 // GMT: Tuesday 17. May 2022 12:11:19
end_at = 1652789579 // GMT: Tuesday 17. May 2022 12:12:59
interval = 10 // sec
```

2. Assume that the pay stream has ended (more than 100 seconds have elapsed since the start), and the recipient calls the `withdraw_payment` instruction.
3. In the body of this instruction, method `amount_available` is called, which returns the following value for the selected parameters:

```
// Start: 1652789479, End: 1652789579, Interval: 10, amount 1000,
withdrawal: 0
// Duration: 100, interval amount: 100, nr intervals: 11
recipient_amount = Some(1100);
```

4. However, the token escrow account for a given payment contains only 1000 tokens, so this instruction ends with error `Program log: Error: insufficient funds`. **Thus, the recipient will never get paid for the service provided, and the tokens get stuck in the escrow.**

   Note: Even if the withdrawal method is called periodically at each interval, the payment for the last interval will fail, and the tokens get stuck in the escrow token account.

Recommendation

Fix the `amount_available` method to handle the edge cases.

Go back to Findings Summary

C2: Possibility of stealing tokens from escrow token account

Impact:	High	Likelihood:	High
Target:	withdraw.rs, cancel_payment.rs	Type:	PDA sharing

Description

The `Withdraw` context does not check whether the given `payment_token_account` is associated with the `payment_account`. Since each payment has a separate `payment_token_account`, one should check that the `payment_token_account` really belongs to the given `payment_account`. Problem is in authority of the `payment_token_account` as the authority is a PDA and is shared accross all the `payment_token_accounts`. These errors allow an attacker to pass in any `payment_token_account`.

Everything mentioned also applies to the `CancelPayment` context.

Exploit scenario

1. Suppose the pay stream creation with the following parameters:

```
amount = 1000 // tokens
start_at = 1652789479 // GMT: Tuesday 17. May 2022 12:11:19
end_at = 1652789499 // GMT: Tuesday 17. May 2022 12:11:39
interval = 5 // sec
```

2. Now the attacker will create a valid payment in which he will act as both a sender and a recipient (he will have authority over two wallets). His pay stream will have the following parameters (notice that the attacker's stream ends before the valid payment):

```
amount = 500 // tokens
start_at = 1652789484 // GMT: Tuesday 17. May 2022 12:11:24
end_at = 1652789494 // GMT: Tuesday 17. May 2022 12:11:34
interval = 5 // sec
```

3. Assume that the attacker's pay stream has ended (more than 10 seconds have elapsed since the start), and the recipient (attacker) calls the `withdraw_payment` instruction. However, instead of using the `payment_token_account` of his payment, he uses the `payment_token_account` associated with the payment created in the first step. So the `Withdraw` context will look like this:

```
pub struct Withdraw<'info> {
    pub recipient: // attacker's wallet 2
    pub recipient_token_account: // attacker's wallet account 2
    pub payment_account: // payment created in the second step
    pub payment_token_account: // escrow for the payment from step 1
    pub pda_signer: // PDA authority of the helio program
    pub token_program: // token program
    pub system_program: // system program
}
```

4. Now, as the `amount_avaliable` method is not working correctly (see issue C1), it will return the value `Some(750)`. **So the attacker can gain an additional 250 tokens by front-running the withdrawal of the valid recipient.**

Recommendation

Do not share the one PDA authority for all the `payment_token_accounts`. Create separate PDA authority for each payment by using the `payment_account` 's public key as one of the seeds. It will create the save association between each `payment_account` and `payment_token_account`.

Go back to Findings Summary

C3: Possibility of stuck tokens

Impact:	High	Likelihood:	High
Target:	payment.rs	Type:	Arithmetic

Description

The `amount_available` method will not work if the `amount * interval < duration`. In such cases, the `interval_amount` will be zero (due to truncating division), and it causes the result of the `amount_available` method to be zero, even though the escrow account contains some tokens.

Exploit scenario

1. Suppose the pay stream creation with the following parameters:

```
amount = 10 // tokens
start_at = 1652943677 // GMT: Thursday 19. May 2022 7:01:17
end_at = 1652943707 // GMT: Thursday 19. May 2022 7:01:47
interval = 2 // seconds
```

2. Assume that the attacker's pay stream has ended (more than 30 seconds have elapsed since the start), and the recipient (attacker) calls the `withdraw_payment` instruction.
3. In the body of this instruction, method `amount_available` is called, which returns the following value for the selected parameters:

```
// Start: 1652943677, End: 1652943707, Interval: 2,amount 10,
withdrawal: 0
// Duration: 30, interval amount: 0, nr intervals: 16
recipient_amount = Some(0);
```

4. The `amount_available` method returns zero, so the `withdraw_payment`

> instruction fails with the error: `Error Code: EmptyAccount. Error Number: 6004. Error Message: The account is empty, nothing to withdraw…`

Recommendation

Fix the `amount_available` method, and test it properly.

Go back to Findings Summary

C4: Using the same struct for SOL payments as for token payments results in the possibility of a tokens lock attack

Impact:	High	Likelihood:	High
Target:	cancel_payment.rs	Type:	Arithmetic

Description

Since both types of payments (SOL and payment token) share the `PaymentAccount` structure, the program allows sending a valid token payment (`payment_account` of token pay stream) into SOL instructions instead of SOL payment.

Exploit scenario

1. Suppose the pay stream creation with the following parameters:

```
amount = 1000 // tokens
start_at = 1652952586 // GMT: Thursday 19. May 2022 9:29:46
end_at = 1652952686 // GMT: Thursday 19. May 2022 9:31:26
interval = 10 // seconds
```

2. Assume that the pay stream has ended (more than 30 seconds have elapsed since the start). Note: Once the payment stream ends, anyone can call the `cancel_payment` instruction.
3. The attacker sends 1000 lamports to the `payment_account` created for the payment from the first step and calls the `cancel_sol_payment` instruction where instead of some SOL `payment_account`, he passes the `payment_account` from the first step, which is now credited with additional 1000 lamports (so the SOL transfer won't fail). Because SOL payments and token payments share the same data type for `payment_account`, the helio program

accepts this account.

4. Therefore, the payment recipient will receive 1000 lamports on his SOL wallet, but his 1000 tokens, which he was initially supposed to receive, will remain stuck in the escrow account. Now imagine the situation that these tokens were, for example, the USDC. The attacker, for a loss of about $0.00005072 (at the time of this writing), was able to lock tokens worth of $1000.
5. If the recipient now tries to call the `withdraw_payment` instruction, he will get this error: `payment_account. Error Code: AccountNotInitialized. Error Number: 3012. Error Message: The program expected this account to be already initialized.` as the payment_account does not exist.

Recommendation

There are at least two options how to deal with this error:

- Do not share the same data structure for token and SOL payments.
- Look at SOL payments as token payments and work with a wrapped SOL token.

Go back to Findings Summary

M1: Hanging `payment_token_account`(s)

Impact:	Low	Likelihood:	High
Target:	cancel_payment.rs	Type:	Accounts management

Description

There is currently no way to close (and by that, redeem the fees from) `payment_token_account` accounts. The fees (lamports paid for the account creation) stay stuck there forever.

Recommendation

The token program offers the option of closing the token account using the `TokenProgram::CloseAccount` instruction. This instruction requires the signature of the owner of the given token account, which is a PDA, and therefore it is necessary to call this instruction in the `cancel_payment` instruction.

Go back to Findings Summary

L1: Using `find_program_address` instead of `create_program_address`

Impact:	Low	Likelihood:	Low
Target:	cancel_payment.rs, withdraw.rs	Type:	Compute budget

Description

With `find_program_address` there is a risk of exceeding the instruction compute budget. The processes of finding a valid program address is by trial and error, and even though it is deterministic given a set of inputs it can take a variable amount of time to succeed across different inputs. This means that when called from an on-chain program it may incur a variable amount of the program's compute budget.

Recommendation

As all account addresses accessed by an on-chain Solana program must be explicitly passed to the program, it is typical for the PDAs to be derived in off-chain client programs, avoiding the compute cost of generating the address on-chain. The address may or may not then be verified by re-deriving it on-chain, depending on the requirements of the program. This verification may be performed without the overhead of re-searching for the bump key by using the `create_program_address` function.

Go back to Findings Summary

I1: `PaymentAccount` struct has unused fields

Impact:	Informational	Likelihood:	N/A
Target:	payment.rs	Type:	Code smell

Description

The `id` and `nr_payments` fields of the `PaymentAccount` struct are never used.

Recommendation

Explain or delete these fields. By removing the fields, one can effectively reduce the size of a payment account, thus reducing the fees paid as rent.

Go back to Findings Summary

I2: Unnecessary mutable modifier

Impact:	Informational	Likelihood:	N/A
Target:	create_payment.rs	Type:	Code smell

Description

In structure `CreatePayment`, accounts `recipient` and `recipient_token_account` do not have to be mutable.

Recommendation

Remove the `mut` modifier.

Go back to Findings Summary

Appendix A: Non-Security-Related Recommendations

A.1. Don't use explicit `require!(…)` when not necessary

It is not a good practice to check the accounts explicitly inside the instruction body (loc: 22).

Recommendation

Use `Program<'info, Token>` constraint instead of explicit require.

A.2. Avoid using `#account(signer)`

If you use the `#[account(signer)]` constraint, the Anchor's linter will raise an error that the given account is unsafe and should be documented (using the `CHECK` doc comment).

Recommendation

Using the `Signer<'info>` constraint, you can avoid this error.

A.3. Associated token accounts

When working with tokens, always use the associated token accounts. Then it is easy to check if a user passed the correct account into your program.

7. Appendix B

7.1. How to cite

Please cite this document as:

Ackee Blockchain, ”Helio Protocol Audit”, May 20, 2022.

If an individual issue is referenced, please use the following identifier:

`ABCH-{project_identifer}-{finding_id}`,

where `{project_identifier}` for this project is `HELIO-PROTOCOL-AUDIT` and `{finding_id}` is the id which can be found in Summary of Findings. For example, to cite C1 issue, we would use `HELIO-PROTOCOL-AUDIT-C1`.

8. Appendix C: Fix Review

On July 22, 2022, Ackee Blockchain reviewed Helio's fixes for the issues identified in this report. The following table summarizes the fix review.

Fix log

Id		Type	Impact	Likelihood	Status
C1F	C1: `withdraw_payment` and `cancel_payment` instructions will not work after the pay stream ends	Arithmetic	High	High	**Fixed**
C2F	C2: Possibility of stealing tokens from escrow token account	PDA sharing	High	High	**Fixed**
C3F	C3: Possibility of stuck tokens	Arithmetic	High	High	**Fixed**
C4F	C4: Using the same struct for SOL payments as for token payments results in the possibility of a tokens lock attack	Type cosplay	High	High	**Fixed**

Id		Type	Impact	Likelihood	Status
M1F	M1: Hanging `payment_token_account(s)`	Accounts management	Low	High	**Fixed**
L1F	L1: Using `find_program_address` instead of `create_program_address`	Compute budget	Low	Low	**Fixed**
I1F	I1: `PaymentAccount` struct has unused fields	Code smell	Informational	N/A	**Fixed**
I2F	I2: Unnecessary mutable modifier	Code smell	Informational	N/A	**Acknowledged**
I3	Anchor version mismatch	Version mismatch	Informational	N/A	**Reported**
I4	Impossible to build and test with a newer anchor version	Developer experience	Informational	N/A	**Reported**
I5	A missing `CHECK` doc comment	Unsafe struct field	Informational	N/A	**Reported**

Table 2. Table of Findings

8.2. C1F: `withdraw_payment` and `cancel_payment` instructions will not work after the pay stream ends

Impact:	High	Likelihood:	High
Target:	C1: `withdraw_payment` and `cancel_payment` instructions will not work after the pay stream ends	Type:	Arithmetic

Description

There is now a limit for the `due_amount` variable in the `amount_available` method. If the due_amount (calculated with `nr_intervals * interval_amount`) is larger than the total amount, it is set to the total amount. That way, overflow is avoided.

Go back to Fix log

8.3. C2F: Possibility of stealing tokens from escrow token account

Impact:	High	Likelihood:	High
Target:	C2: Possibility of stealing tokens from escrow token account	Type:	PDA sharing

Description

PDA account is now derived from the payment account public key as recommended. If the attacker provides a wrong `payment_token_account`, the PDA authority `HelioError::InvalidPDASigner` is returned.

Go back to Fix log

8.4. C3F: Possibility of stuck tokens

Impact:	High	Likelihood:	High
Target:	C3: Possibility of stuck tokens	Type:	Arithmetic

Description

The `interval_amount` is zero when less than one smallest currency unit is due to the truncation. `interval_amount` is now set to 1 (smallest allowed interval amount) in these cases.

Go back to Fix log

8.5. C4F: Using the same struct for SOL payments as for token payments results in the possibility of a tokens lock attack

Impact:	High	Likelihood:	High
Target:	C4: Using the same struct for SOL payments as for token payments results in the possibility of a tokens lock attack	Type:	Arithmetic

Description

SOL and SPL tokens do not share the same payment account structure anymore. Code is separated, and sending SPL `PaymentStructure` in SOL functions is impossible or vice versa.

Go back to Fix log

8.6. M1F: Hanging `payment_token_account`(s)

Impact:	Low	Likelihood:	High
Target:	M1: Hanging `payment_token_account`(s)	Type:	Accounts management

Description

The `cancel_payment` function, which stops the stream, now closes the `payment_token_account` and refunds the rent to the sender.

Go back to Fix log

8.7. L1F: Using `find_program_address` instead of `create_program_address`

Impact:	Low	Likelihood:	Low
Target:	L1: Using `find_program_address` instead of `create_program_address`	Type:	Compute budget

Description

Now the program takes the `payment_account` key and uses `create_program_address` function to generate the PDA address. The bump is derived on an off-chain client program to save the compute budget.

Go back to Fix log

8.8. I1F: `PaymentAccount` struct has unused fields

Impact:	Informational	Likelihood:	N/A
Target:	I1: `PaymentAccount` struct has unused fields	Type:	Code smell

Description

The `id` and `nr_payments` fields of the `PaymentAccount` are removed from the struct.

Go back to Fix log

I3: Anchor version mismatch

Impact:	Informational	Likelihood:	N/A
Target:		Type:	Version mismatch

Description

The version of the Anchor framework is different for program (`v0.23.0`), tests (`v0.21.0`), and CLI (`v0.18.2`).

Recommendation

Use the exact version of the Anchor framework for program, tests, and CLI. Preferably one of the versions `v0.24.2` or `v0.25.0`.

Go back to Fix log

I4: Impossible to build and test with a newer anchor version

Impact:	Informational	Likelihood:	N/A
Target:		Type:	Developer experience

Description

When building and testing the program with the Anchor CLI version newer than `v0.18.2`, it is impossible to build the program and run the tests.

Recommendation

Specify the CLI version and the step-by-step process of building and testing the program in the readme file.

Go back to Fix log

I5: A missing `CHECK` doc comment

Impact:	Informational	Likelihood:	N/A
Target:		Type:	Unsafe struct field

Description

In newer versions of Anchor, the linter will raise an error that the given account is unsafe and should be documented (using the `CHECK` doc comment).

Recommendation

Use the `CHECK` comment on all necessary places in the structs and describe why the account is not unsafe.

Go back to Fix log

ackee blockchain

Thank You

Ackee Blockchain a.s.

Prague, Czech Republic

hello@ackeeblockchain.com

https://discord.gg/wpM77gR7en