Security Audit – Squads MPL Multisig

Neodyme AG

March 13th, 2023

Contents

Introduction

Squads engaged Neodyme to do a detailed security analysis of their Multisig MPL on-chain program. A thorough audit was done between the 16th of January 2023 and the 24th of January 2023.

The audit revealed no significant issues in the on-chain program. A number of possible social-engineering vectors and improvements were found. All issues were fixed by the Squads team.

The following report describes all findings in detail.

Project Overview

Squads MPL (Multisig Program Library) is a contract that provides multisig capabilities on the Solana blockchain. It allows anyone to create a new multisig, add arbitrary members to it and configure the singing threshold. Once set-up, any multisig member can propose a transaction consisting of instructions. The multisig members can then vote on that proposal. It succeeds if the threshold of required votes is reached. Once succeeded, a proposal can be executed using cross-program-invocation (CPI) via the multisig. During this execution, a program-derived address (PDA) attaches a signature unique to that multisig.

This is a flexible type of multisig, as it allows the multisig members to control anything that can otherwise be controlled by a single Solana key, such as token accounts or program upgrade authorities.

Changes to the multisig parameters are possible by calling the MPL program in a transaction, which has to be signed by the multisig itself. This makes parameter changes use the same interface as other "external" transaction proposals.

Scope

The scope of this audit is the on-chain contract of Squads MPL. The audit started on commit `7ef3ed2bab2f68fb49d5db327854b2fccc092bb0`. The commit which includes all fixes is `f2ba643daed5f660db02aac548ca6c2022efc507`, both on the main branch.

While considering social engineering vectors, we also briefly examined other components for functionality, including the Squads multisig website.

The source code of the audited squads-mpl contract is open-source at https://github.com/Squads-Protocol/squads-mpl/

Methodology

Neodyme's audit team performed a comprehensive examination of the Squads MPL contract. In addition to the usual checks outlined below, special attention was placed on the feasibility of social-engineering attacks that result out of the contracts design.

- Ruling out common classes of Solana contract vulnerabilities, such as:
  - Missing ownership checks,
  - Missing signer checks,
  - Signed invocation of unverified programs,
  - Solana account confusions,
  - Re-initiation with cross-instance confusion,
  - Missing freeze authority checks,
  - Insufficient SPL token account verification,
  - Missing rent exemption assertion,
  - Casting truncation,
  - Arithmetic over- or underflows,
  - Numerical precision errors.
- Checking for unsafe design that might lead to common vulnerabilities being introduced in the future,
- Checking for any other, as-of-yet unknown classes of vulnerabilities arising from the structure of the Solana blockchain,
- Ensuring that the contract logic correctly implements the project specifications,
- Examining the code in detail for contract-specific low-level vulnerabilities,
- Ruling out denial-of-service attacks,
- Ruling out economic attacks,
- Checking for instructions that allow front-running or sandwiching attacks,
- Checking for rug-pull mechanisms or hidden backdoors.

Findings

All findings are classified in one of four severity levels:

- **Critical**: Bugs that will likely cause loss of funds. This means that an attacker can, with little or no preparation, trigger them, or they are expected to happen accidentally. Effects are difficult to undo after they are detected.
- **High**: Bugs that can be used to set up loss of funds in a more limited capacity, or to render the contract unusable.
- **Medium**: Bugs that do not cause direct loss of funds but lead to other exploitable mechanisms.
- **Low**: Bugs that do not have a significant immediate impact and could be fixed easily after detection.

Name	Severity
MPL1: Iterative Execution has undocumented Edge Cases	Info
MPL2: Iterative Execution uses the wrong signers for the 0 Authority	Info
MPL3: Bypass of OS-SQD-SUG-01 fix via recursion	Info
Social-Engineering SE1: Squads MPL Website blindly trusts Metadata	High
Social-Engineering SE2: Reduce signer threshold via remove-then-add scheme	Low

Main Audit Findings in the MPL On-Chain Program

This section describes the three main findings in the on-chain MPL program.

MPL1: Iterative Execution has undocumented Edge Cases (Info)

Severity	Impact	Affected Component	Status
Info	Users might not understand what a transaction can do	Iterative Execution	Acknowledged

The MPL program allows proposed and approved transactions to be executed iteratively. This is needed in case one transaction would otherwise use too many compute units at once. This is a useful feature that allows users to vote on many instructions at once, which, when approved, can only be executed together.

However, the “guarantees” for iterative execution and for normal execution are different. This might trip up users, and is insufficiently documented. When looking at a proposal, a user is likely to assume that either all of the proposed instructions get executed, or none will.

To see why this can be an issue, consider failing instructions. Say you have three instructions, A, B and C. With normal execution, you will be guaranteed that either all of them succeed, or none do. With iterative execution, this is not the case. A might succeed, but if B fails, there is no way to undo A.

Reasons why an instruction might fail are numerous, here are some examples:

- If a user proposes to do a token swap using a third-party program, and while multisig members are voting that external program does an upgrade to change its swap interface, the “old” proposed instruction might no longer work. While that is unlikely to happen, as they’d essentially break compatibility of their whole ecosystem, it is *technically* possible.
- There could just be an error while creating the instruction data. Sure, that should not really happen, and every multisig member should verify before signing, but given that instruction data is a byte-blob which is created and visualized by different code than the code that will parse it on the blockchain, errors can happen.
- Also consider errors like “account erroneously not specified as writable”, which are quite hard to spot without simulation.
- The chain-state might have changed. If you transfer tokens somewhere, and the target token account is closed, the transaction will revert. This closing (and potential reopening) of a token account might be done by people not directly affiliated with the multisig if they are compensated for something.

Most of the time, a user can work around these issues by simply proposing a new, fixed transaction and voting on that. But there might be time-sensitive things, which can't be easily remedied, especially if you consider the creator of the proposal malicious.

Further, this can be dangerous if the multisig does operations on itself during iterative execution. Some combinations of "some multisig members lost their keys" and adding/removing members can create a situation where a full execution of a transaction is fine, but an iterative one permanently breaks the multisig.

A user can simply work around any issues by only adding a single instruction to a proposal. In that case, both execution methods are identical.

Suggestion Iterative execution is nice to have, but not required for the majority of transactions MPL will likely handle.

We therefore suggest to some combination of

a) Remove the iterative execution feature
b) Document the differences and potential foot guns clearly
c) Make the iterative-execution opt-in per transaction (or per multisig), warn the user when he enables it, and warn users when voting on transactions with it enabled.

Further, while it is currently impossible to change multisig members with iterative execution (see MPL2 below), it isn't an explicit ban. We recommend making that explicit.

Fix The Squads team added more comments to the code and plan to add more notes about this behavior in the readme as well.

MPL2: Iterative Execution uses the wrong signers for the 0 Authority (Low)

Severity	Impact	Affected Component	Status
Low	Transaction gets executed with unexpected signature.	Iterative Execution	Resolved

MPL allows using an `authority_index` (a `u32`) as seed for signing multisig transactions. This allows one multisig to control multiple authorities.

The `0` authority is special, as it is required to modify the multisig itself. To add a member, a user proposes a transaction with authority 0 to call MPL itself to add a member. This is implemented by special casing the 0 authority. However, it is different between iterative execution via `execute_instruction` and single-shot execution via `execute_transaction`.

In single-shot, MPL has a special-case and uses the multisig PDA authority for the 0 authority. In iterative execution, the "normal" 0 authority is used.

Suggestion There should not be two different authorities in the two different execution methods, even though this does not have any immediate implications.

Due to the issues described in MPL1, we recommend forbidding the use of the 0 authority in iterative execution. Otherwise, a "stuck" execution due to failing transactions, which modifies the multisig can in edge-cases permanently brick a multisig.

Fix The Squads team blacklisted the 0 authority from iterative execution.

MPL3: Bypass of OS-SQD-SUG-01 fix via recursion (Info)

Severity	Impact	Affected Component	Status
Info	Bypass of previous fix. No impact, except for inconsistency.	Transaction Execution	Resolved

MPL currently blacklists signing with the special "0" authority for external transactions. This was previously remarked by the OtterSec Audit, which reported `OS-SQD-SUG-01 | Enforce Signed Multisig Program` as resolved.

We have found a bypass for the employed fix.

MPL currently checks that it does not execute a foreign program with 0 authority (the PDA authority). However, a user can invoke MPL itself with another `execute_transaction` instruction recursively. In Solana, you can keep all signatures from the parent attached:

```
- AUTH: [NONE]; MPL execute_transaction(attacker_transaction_0, use
   authority 0)
    - AUTH: [0]; MPL execute_transaction(attacker_transaction_1, use
       authority 1)
            - AUTH: [0, 1]; ATTACKER_CONTRACT.

attacker_transaction_0 = IX {
    keys = [...],
    program = MPL,
    data = IX {             // attacker_transaction_1
            keys = [...],
            program = ATTACKER,
            data = "",
    },
}
```

Suggestion In our estimation, the original bug does not have any impact. Ottersec writes

> This leads to a possibility where you could drain the lamports out of the multisig account itself, causing it to become no-longer rent exempt.

However, at the time of execution, all multisig accounts are owned by the MPL program, so only the MPL program can modify their lamports. As the MPL program doesn't currently have a close instruction, lamports cannot be subtracted, so signing with the multisig PDA itself can't cause any issues.

But this bypass is still an inconsistency that should be fixed. This can be done by blacklisting the `execute_transaction` instruction from being executed with the 0 authority, same as all other external programs. Non-direct reentrancy is forbidden by Solana, so MPL does not have to worry about the case where another program calls back into it later on.

Fix The Squads team blacklisted the execute instruction/transaction endpoints for the 0 authority in commit `09470403a684b8d4d83928a46dd76d2173de5c21`. Neodyme verified this.

Off-Chain Social Engineering Vectors

With multisigs, the technical security of the contract is only one side of the coin. The other is the ability of multisig members to confidently vote on proposals. A multisig therefore inherently relies on all multisig members being able to verify the proposal they want to vote on.

In such situations, you always have to consider the possibility of social engineering. If a user can create a proposal, and successfully lie to the other members of the multisig what it does, he can cause unwanted effects to occur. In the case of MPL, where only trusted members of the multisig can propose transactions, the surface of these kinds of attacks is limited to insider attacks.

The provided tools should help users see what they are actually voting on, without relying on the proposers' commentary. Squads MPL provides a website that parses the proposal information and shows it to the user.

We have carefully considered possible ways to make social engineering attacks work against MPL, and have some ideas where the user-interface of both contract and web can be improved to make them more difficult.

SE1: Squads MPL Website blindly trusts Metadata (High)

Severity	Impact	Affected Component	Status
High	Social-Engineering Attack. Show fake transaction details.	Squads Multisig Website	Resolved

While the MPL website isn't the main target of this audit, we briefly looked at the whole multisig flow.

For "unknown" transactions, the website visualizes a fairly basic but complete view of the proposed instructions. This allows users to verify what is happening, although they have to be a bit technical to understand the information.

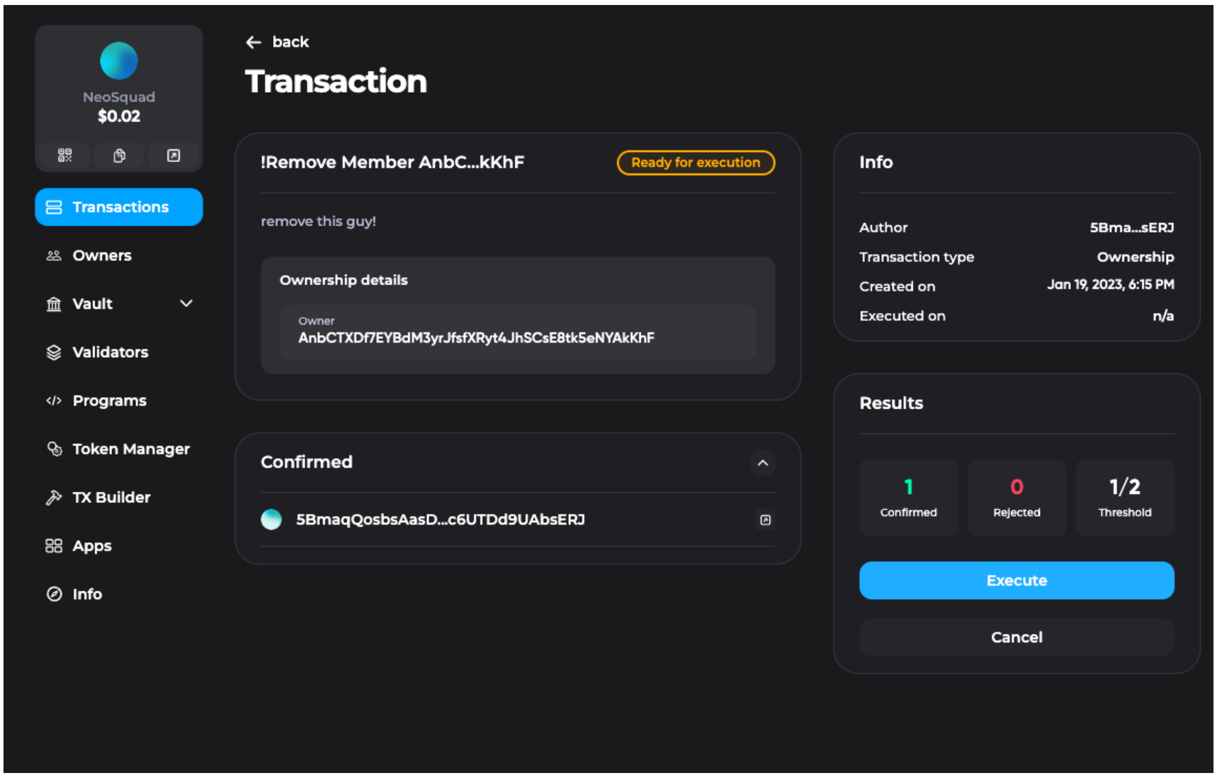
To help users understand the most important operations like add- and remove-member, the website parses these kinds of transactions more. It does this by relying on metadata pushed to the chain during proposal creation.

That is where we found an issue: The website blindly trusts the meta info pushed on-chain by the proposal-creator.

Whenever the user creates a new transaction proposal via the website, the web interface stores meta information about the transaction with the TxMeta contract. This includes Title, Description, Action

Type etc. Relevant information such as action type is not checked to be correct on-chain. That in itself is not a problem, and would be rather difficult to do. The problem lies in the fact that the web interface treats this information as trusted and uses the provided metadata when rendering a transaction.

Essentially, an attacker can create proposals that do the opposite of what they seem to do in the interface. As a proof-of-concept we have created two transactions, both adding a new multisig member, but one claiming to be removing one:



Transaction

!Remove Member AnbC...kKhF — Ready for execution

remove this guy!

Ownership details

Owner
AnbCTXDf7EYBdM3yrJfsfXRyt4JhSCsE8tk5eNYAkKhF

Confirmed

5BmaqQosbsAasD...c6UTDd9UAbsERJ

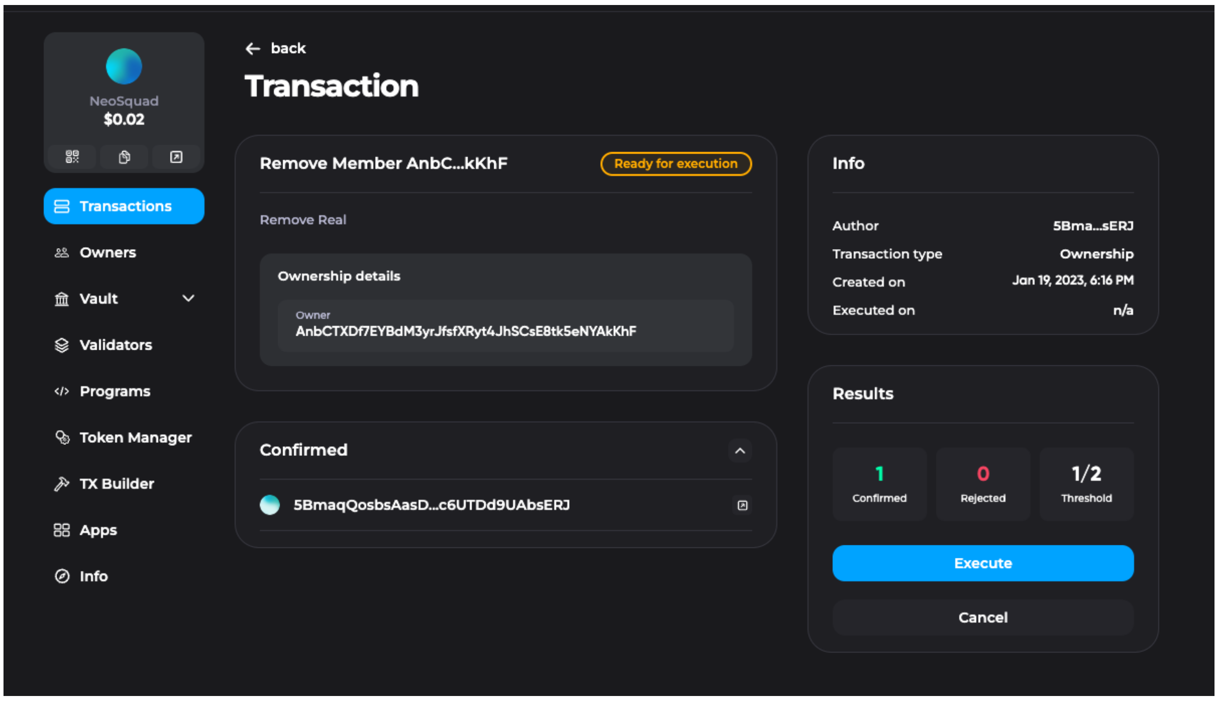
Info

Author	5Bma...sERJ
Transaction type	Ownership
Created on	Jan 19, 2023, 6:15 PM
Executed on	n/a

Results

1	0	1/2
Confirmed	Rejected	Threshold

Figure 1: fake proposal showing remove-member, but would actually add a member to the multisig. Exclamation mark added for clarity.



Transaction

Remove Member AnbC...kKhF — Ready for execution

Remove Real

Ownership details

Owner
AnbCTXDf7EYBdM3yrJfsfXRyt4JhSCsE8tk5eNYAkKhF

Confirmed

5BmaqQosbsAasD...c6UTDd9UAbsERJ

Info

Author	5Bma...sERJ
Transaction type	Ownership
Created on	Jan 19, 2023, 6:16 PM
Executed on	n/a

Results

1	0	1/2
Confirmed	Rejected	Threshold

Figure 2: real proposal to perform remove-member

- Real Proposal, showing remove-member
- Fake Proposal, showing remove-member instead of add-member

Suggestion Add a check that verifies the meta info matches the IDL when you render.

Fix The Squads website was promptly updated to include an “expand” button that now reveals all instructions, accounts and data for all instructions, whether they get special parsing or not. This allows anyone to verify the actual contents without relying on the metadata. The metadata is still shown as before, but Squads is working on API and client fortification to add more checks for the metadata.

SE2 Social Engineering Idea: Reduce signer threshold via remove-then-add scheme (Low)

Severity	Impact	Affected Component	Status
Low	Social Engineering Attack. Threshold of a multisig can be lowered when a user might not expect it	Multisig Member Management	Acknowledged

At any time, a multisig can add and remove members. If you have a 2-out-of-2 multisig, one member could try to sneakily change it to a 1/2 multisig by creating a transaction where they claim that they need to change their key, including a transaction with the two instructions `remove_user(old_key)` and `add_user(new_key)`. However, `remove_user` would change the threshold to 1/1, and `add_user` would then change it to 1/2.

This isn't obvious and might not be caught by the other multisig party. It doesn't necessarily have to be a 2/2 multisig, any "full-threshold" multisig will show this behavior.

Suggestion This isn't a technical issue, but a social one that can be improved with documentation and naming. For example, MPL could force the use of `remove_member_and_change_threshold` whenever the threshold would change, by simply adding a check that aborts if `current threshold == member_count`. MPL could also add a flag to remove user, something along the lines of `allow_decrease_threshold`. Only if this is set, allow a decrease of the threshold.

Miscellaneous Notes and Security Unrelated Nitpicks

As a multisig is a highly trusted program, we also have some ideas how to improve the code and make it more easily understood by other parties. We also want to note down some ideas we had that don't have any immediate consequences, but might come into play if either Squads or another 3rd party adjusts the MPL program.

This section therefore contains a number of ideas and nitpicks we have encountered while auditing. They are not directly security relevant, but can in places improve the current or future code.

NP1 Maximum Instruction Size

MPL currently has a maximum instruction size of 1280 bytes. We assume this limit is based on the previous value of `max_cpi_instruction_size`, and used to ensure that iterative execution does not fail in the middle due to instruction size too large errors. This limit of 1280 was a bit arbitrary in the Solana client. It is the maximum packet size Solana currently uses for transactions, but when you account for headers, you are left with only 1232 bytes for serialized transactions (Docs). Solana is currently changing the limit to allow for up to 10kB instruction data: (Pull-Request):

```
/// Maximum CPI instruction data size. 10 KiB was chosen to ensure that CPI
/// instructions are not more limited than transaction instructions if the size
/// of transactions is doubled in the future.
pub const MAX_CPI_INSTRUCTION_DATA_LEN: u64 = 10 * 1024;
```

That new 10kB limit matches the current maximum you can one-shot allocate for a new PDA. MPL could drop the limit altogether or raise it to 10kB, but keeping as-is is also fine, just unnecessarily limiting. Note that the feature that increases the CPI size (`GDH5TVdbTPUpRnXaRyQqiKUa7uZAbZ28Q2N9bhbKoMLm`) is not yet activated on mainnet. In any case, there should be a comment as to why that limit is there because it is not immediately obvious.

Fix The Squads team removed the limit.

NP2 MPL only allows for a total of u32 transactions

MPL only has a u32 for transaction indices available. Once that is saturated, the multisig can never execute instructions again.

While it is unlikely that this limit will be reached, it is *technically* possible. The following examines the cost and time constraints.

Currently, all proposed transactions have to stick around. Making `2**32` rent-exempt accounts is prohibitively expensive. Assuming each TX account has 0 instructions and 1 multisig member, it would cost 7.4M SOL and would definitely be noticed due to the increased storage requirements validators will encounter.

Should MPL ever allow closing of old transaction proposals, this changes. You still have to uniquely advance the index by one for each potential transaction. `2**32` times transaction-fee of 0.000005 SOL is around 20kSOL, so around 400k€ at current prices. If an attacker can open and close say 10 proposals per transaction, this lowers to ~40k€. Such an attack would take forever (at 10 creations per transaction, 10 transactions per slot, each 0.4s, this would be ~200 days of constant transactions). Due to this volume, this might be noticed before it becomes an issue.

MPL could then bump the index up to 64-bit which avoids this problem.

Response Squads noted that they'll be monitoring metrics for all transactions on MPL, therefore noticing such a hugely increased transaction volume early.

NP3 No rent refunds

Related to the issue above, there are no rent refunds for executed transactions. It can be costly to keep them around. We know some users have been complaining about this for spl-governance. However, MPL has to be careful to not introduce edge cases when allowing to close transactions. Having old transactions around can also be helpful in keeping multisig members accountable and in visualizing who signed what.

Response

> We're opting to maintain all accounts, we'd rather not introduce issues around making it easier to attack as mentioned in NP2, as well as provide transparency and history on-chain for our users.

NP4 execute_external allows for sandwich attacks

Sandwich attacks are attacks more commonly known with mempool based chains. A 'victim' transaction (for example one that is doing some kind of swap) is sandwiched between two attacker transactions (which for example manipulate the price of the asset in question by swapping a large amount beforehand, and to avoid losses swap back afterward). Feasibility of such attacks always depends on the exact circumstances, which are hard to predict given that nobody knows which kind of transaction might be put in the multisig.

This is fairly obvious if one thinks about it, but is a foot gun users should be warned about when enabling external execute, and maybe even warned about on each TX proposal/vote when `external_execute` is enabled.

Fix As no users had the external execution feature enabled, the Squads team has deprecated and removed it. All executions must now be performed by a member of the multisig.

NP5 `change_index` is undocumented

MPL currently implements a feature that invalidates all in-progress proposals whenever a change to the multisig config is made. This is especially relevant when adding and removing members. There are no issues with the technical implementation, but documentation can be improved:

- MPL should document what `change_index` does, and when it changes. Both on a technical level in the code, and in user documentation. Currently, the "make it impossible to vote on TX when a multisig changes, but ready TX can still be executed" behavior is not obvious.
- it also does not update when changing `external_execute`. That could be an issue if you consider sandwich attacks. Other than that, it does not change much about the security.

Fix External execution was removed (see NP4). Further, some inline comments were added to the MPL source-code describing the feature.

NP6 Be aware that automated execution (via external execution) can be dangerous

We don't know if/how any automated execution of multisig transactions will be implemented via the external execute feature. But be aware that this kind of pattern can be used to steal the funds of the operator running the external-exeuction crank. If an attacker can guess an operator's public key, they can specify it as signer in the multisig transaction and call a malicious "drain everything" program, which will then have a valid operator signature once automatically executed.

Fix External execution was removed.

NP7 `set_executed` of instructions is never read and sometimes stale

When iteratively executing, MPL marks each instruction as executed. MPL does not do this when a user executes single-shot. This isn't a problem as MPL marks the tx as executed, but it will leave instructions behind that are wrongly marked unexecuted. This is technically “invalid state”, which sometimes leads to bugs on protocol extensions, even though at the time of implementation all edge-cases are fine.

MPL could add a comment in the struct notifying other devs about this, so they don’t make invalid assumptions in future changes or custom frontends.

IN addition, MPL never actually reads the `set_executed()` field from instructions. The replay-protection is based on the instruction’s index. MPL thus could entirely omit the field.

Fix As the field wasn’t required, the Squads team removed it in favor of the `executed_index`.

NP8 Payer special-case in add-member execution is technically not required

The special handling in transaction execution of `add_member`, where MPL swaps the “payer” for the currently executing operator is only needed for the rent increase, that can happen if the account is too small to store the new member and needs to be resized. MPL could have the client transfer the needed rent when proposing the transaction, then this special-casing wouldn’t be necessary. There aren’t any real issues with the current method, but it increases on-chain code-complexity.

There should also be a comment describing why this special casing is there in the code.

Fix The Squads team has removed the special casing, opting to move the logic to the client as suggested. A new error now tells that the account does not have enough lamports for the resize operation. This makes the most crucial part of the program a lot more intuitive to understand.

NP9 Inconsistent checks of multisig seeds

MPL always checks the address of the multisig against the expected seeds, in addition to the Anchor Type check. Except in `MsAuth` and `MsAuthRealloc`, where MPL omits the seed check. This isn’t an issue, as one of the two checks would be sufficient. Just an odd pattern-break, that makes it harder to reason about the program.

Fix Together with MP10, the Squads team made `MsAuth` and `MsAuthRealloc` more concise.

NP10 MsAuth has duplicate account

In `MsAuth`, MPL currently specifies both `multisig` and `multisig_auth`. In all cases, these two will be the same account, as the `create_key` is unchangeable, and the `_auth` account has the same seeds as the multisig. MPL can thus drop the auth, and put all constraints on the multisig account. This will make transactions a bit smaller and the code simpler.

Fix Together with MP9, the Squads team made `MsAuth` and `MsAuthRealloc` more concise.

NP11 Unchecked create_key

When a user creates a new multisig, they provide a `create_key`. This is any arbitrary, unchecked Solana account key. It is used to seed the multisig account, so only one multisig can exist for every `create_key`.

The fact that "create_key" in create can be arbitrary is slightly dangerous, as it allows an attacker to recreate a multisig (with his own members) should a multisig ever "vanish". It can't vanish without another bug, though! MPL could add a check that this `create_key` has to sign.

NP12 _meta is undocumented

The unused parameter `_meta` in `create()` is undocumented. The Squads website is currently using it to store information about the creation of the multisig on-chain, MPL could add a comment describing that so people looking at the code know what it is for.

Fix The Squads team added a small note about the `_meta` field.

Neodyme AG

Dirnismaning 55
Halle 13
85748 Garching
E-Mail: contact@neodyme.io

https://neodyme.io