Argent smart contracts audit

- **Repositories**:
  github.com/argentlabs/argent-contracts-zksync
  (commit 0ed59fe4e94fbab15ecf8b90677a8bb00143a568)
- **Dates**:
  03.04.2023 - 25.04.2023

Findings

Scope:
argent-contracts-zksync/contracts
(except argent-contracts-zksync/contracts/test)

- **H-01: No system checks of transaction signed fields in the “executeTransactionFromOutside” function**

Severity: high
Context: ArgentAccount.sol#L296-L302
Description
It is possible to execute a standard L2 transaction through the “executeTransactionFromOutside” function flow by using the same authorization method as for account abstraction transactions. However, this creates a vulnerability that allows front-running of the execution of any L2 transaction with incorrect values for gasLimit, gasPerPubdataByte, feePerGas, and priorityFeePerGas, as well as unexpected tx.origin address, unprocessed paymaster flow, and bytecodes of the factoryDeps array not marked as known. This vulnerability allows an attacker to completely alter the execution flow of a signed transaction.
Recommendation
Use a unique authorization method for validating transactions in the “executeTransactionFromOutside” function. Additionally, it is required to include the allowed transaction executor address as part of the fields that must be signed by the user. This approach would provide protection against the execution of transactions with unexpected context.
Argent: Addressed.
For extra safety, we made it impossible to reuse regular transactions using `executeTransactionFromOutside`. This was done by requiring a gas limit of `0` for transactions coming from outside. We also check that other transaction fields comply with the expected values.
Additionally, we are now requiring the `msg.sender` to be included in the signed data along with the actual transaction.

0xRetrospective: Checked at commit f2f1ff05f670e0d7b6d52b4071ca6491e64e67e0.

- ## H-02: The reuse of default L2 transactions signatures for the execution of argent transactions and vice versa

Severity: high
Context: ArgentAccount.sol#L561-L563
Description
Argent transactions can be executed using the owner's signatures of default L2 transactions of LEGACY_TX, EIP_2930_TX, and EIP_1559_TX transaction types, and vice versa. This is because the "from" field is not part of the preimage of the messages that needs to be signed for such transactions.
Also, if an account is the owner of multiple argent accounts, the transaction signatures can be reused as well.
Also, it is possible to process argent transactions using the owner's signatures of default L2 transactions of EIP_712_TX transaction type through the "executeTransactionFromOutside" function.
Recommendation
Include to the message that is to be signed some project-specific magic value and address of the argent account to which signature refers.
Another possible mitigation is to restrict executions to only EIP_712_TX transactions and implement a specific check on the "_transaction.from" value in the "_validateTransaction" function.
Argent: Addressed.
Modified to allow only ERC712 transactions and to check `_transaction.from`.
0xRetrospective: Checked at commit f2f1ff05f670e0d7b6d52b4071ca6491e64e67e0.

- ## M-01: Unchecked gasPerPubdataByte value in the logic of protection of assets against the case of stolen one of the owner/guardian keys

Severity: medium
Context: ArgentAccount.sol#L496-L556
Description
There is protection against asset theft when access is limited to just one key held by the owner or the guardian. It is based on the fact that the operator does not directly choose the value of basefee, and there is an upper limit on the value of maxPriorityFeePerGas for calls that require only one of the owner/guardian keys. However, this protection is not complete, as there is no check on the gasPerPubdataByte value. It is possible to "spend" a significant amount of gas for the storage write using this value, creating a MEV opportunity for the described scenario.
Recommendation
Allow the execution of calls that require only one of the owner/guardian keys only through the "executeTransactionFromOutside" function.
Another possible mitigation is to add a check on the gasPerPubdataByte value that is similar to the check on the maxPriorityFeePerGas value for the aforementioned calls. However, this solution is less stable as it may not be possible to accurately predict "normal" values for this parameter.

Argent: Addressed.
We added a limit to the `gasPerPubdataByteLimit` to have a ceiling on the maximum amount that can be consumed.
0xRetrospective: Checked at commit f2f1ff05f670e0d7b6d52b4071ca6491e64e67e0.

- ### M-02: Unchecked gasLimit and paymasterInput values in the logic of protection of assets against the case of stolen one of the owner/guardian keys

Severity: medium
Context: ArgentAccount.sol#L496-L556
Description
While there is protection against asset theft when access is limited to just one key held by the owner or the guardian, this protection is not complete. It is so because there are no checks on the gasLimit and paymasterInput values. This creates a vulnerability where it is possible to “spend” a significant amount of gas for the execution of a transaction due to the potential for a very long paymasterInput bytes array, which is creation of MEV opportunity for the described scenario.
Recommendation
Allow the execution of calls that require only one of the owner/guardian keys only through the “executeTransactionFromOutside” function.
Another possible mitigation is to enforce that there is no redundant gas consumption except the minimum required for such calls – you can create an upper bound on the length of the paymasterInput bytes array in such transactions.
Argent: Addressed.
We added checks forbidding the use of any paymaster in the escape methods.
We didn’t add a limit on the gas limit as it can fluctuate. Submitting a high gas limit can increase the gas usage slightly, but it increases logarithmically, so there is a reasonable upper bound to the gas being used. And in the future, MatterLabs wants to make the gasUsed independent of the gas limit.
0xRetrospective: Checked at commit f2f1ff05f670e0d7b6d52b4071ca6491e64e67e0.

- ### M-03: Reentrant execution of account transactions using the “executeTransactionFromOutside” function

Severity: medium
Context: ArgentAccount.sol#L296-L302
Description
The “executeTransactionFromOutside” function allows for the execution of an account transaction inside another transaction execution from the same account. This violates the invariant that account transactions will be executed sequentially.
Recommendation
To maintain the invariant that account transactions are executed sequentially, disallow the execution of transactions through the “executeTransactionFromOutside” function if an account transaction is currently running.
Argent:
We added checks to document this behaviour.

We think it's not a security issue, as even if nested transaction are allowed, the transactions are all signed by owner (and guardian if needed) and the nonces have to be respected. Other smart contract wallets with billions of assets (such as Safe or Argent Vault) have the same behaviour and we are not aware of any issue with it.

- **L-01: Unchecked factoryDeps value in the logic of protection of assets against the case of stolen one of the owner/guardian keys**

Severity: low
Context: ArgentAccount.sol#L496-L556
Description
While there is protection against asset theft when access is limited to just one key held by the owner or the guardian, this protection is not complete. It is so because there is no check on the factoryDeps value. This creates a vulnerability where it is possible to "spend" a significant amount of gas for the publishing of the bytecodes, which is creation of MEV opportunity for the described scenario.
Recommendation
Allow the execution of calls that require only one of the owner/guardian keys only through the "executeTransactionFromOutside" function.
Another possible mitigation is to enforce that there is no redundant gas consumption except the minimum required for such calls – you can enforce that the factoryDeps array is empty in such transactions.
Argent: Addressed.
We added checks to avoid using `factoryDeps` in escape methods.
0xRetrospective: Checked at commit f2f1ff05f670e0d7b6d52b4071ca6491e64e67e0.

- **L-02: Trivial encoding of the message that is to be signed for the "_validateNewOwner" function**

Severity: low
Context: ArgentAccount.sol#L600-L602
Description
The message that is to be signed for the "_validateNewOwner" function is currently encoded as a simple array of bytes, which increases the risk of this message coinciding with messages used in other applications.
Recommendation
To mitigate the aforementioned risk, build the message using the EIP-712 standard.
Argent:
No changes were performed.
We believe the simple encoding is good enough for its purpose, which is preventing accidental mistakes when changing the owner to an address you don't control.
The signed data includes the method selector, which makes the signature unusable for other purposes.

- **L-03: Unsafe selector variable definition in the "_validateTransaction" function**

Severity: low
Context: ArgentAccount.sol#L472
Description
In the “_validateTransaction” function, the value of the selector variable is derived by casting the _transaction.data bytes array into a bytes4 type. However, if the length of _transaction.data is less than 4, the selector variable may have an incorrect value. Although the transaction can still be executed, this may result in unexpected behavior and potential security vulnerabilities.
Recommendation
Define the selector variable as zero in case the length of _transaction.data bytes array is smaller than 4.
Argent: Addressed.
Using the audit’s recommendation.
0xRetrospective: Checked at commit f2f1ff05f670e0d7b6d52b4071ca6491e64e67e0.

- ### L-04: No validation of the new owner in the “initialize” and “escapeOwner” functions

Severity: low
Context: ArgentAccount.sol#L188, ArgentAccount.sol#L408
Description
The “changeOwner” function includes validation of the new owner, but this check is not present in the “initialize” and “escapeOwner” functions. This violates the invariant that the owner must always be validated according to the logic of the “_validateNewOwner” function.
Argent:
No changes were performed.
The signature avoids bricking your account in case you don’t have a guardian. Because you can accidentally change your owner to a wrong address you don’t control.
Check during escape:
When you have a guardian, you can use it to fix your account. For that reason, it’s ok if it’s not used for escaping, as this use case already requires a guardian.
Check during initialization:
Even if there was a check during deployment, it won’t prevent users from depositing to the account, as they can deposit before the constructor is called.
The check can only run during deployment, but deployment can be delayed until the first transaction (that’s the normal scenario). Therefore, adding the check will only surface the issue when doing the first transaction, the same as if there was no check at all.

- ### L-05: Violation of the invariant that the guardianBackup is not defined in cases when the guardian is not defined

Severity: low
Context: ArgentAccount.sol#L423
Description
The “changeGuardian” and “changeGuardianBackup” functions maintain the invariant that the guardianBackup is not defined in cases when the guardian is not defined. But in the

“escapeGuardian” function this invariant is violated – it is possible to remove guardian without removing its backup.
Argent: Addressed.
Added checks enforcing the invariant when triggering the guardian escape.
0xRetrospective: Checked at commit f2f1ff05f670e0d7b6d52b4071ca6491e64e67e0.

- **L-06: Unstable value of MAX_ESCAPE_PRIORITY_FEE constant**

Severity: low
Context: ArgentAccount.sol#L67
Description
Enforcing an upper bound on the gas price that can be used in the escape function calls by checking the maxPriorityFeePerGas value against a hardcoded constant is not a reliable approach. This is due to the fact that this parameter can change unpredictably for normal transaction execution.
Recommendation
Allow calls of escape functions only through the “executeTransactionFromOutside” function.
Argent:
No changes were performed.
If a higher priority fee is needed for some reason, users can still escape submitting transactions from the outside.

- **L-07: Unchecked gasLimit value leads to unacknowledged paymaster refund in the logic of protection of assets against the case of stolen one of the owner/guardian keys**

Severity: low
Context: ArgentAccount.sol#L496-L556, ArgentAccount.sol#L233-L248, bootloader.yul#L1465-L1472
Description
While there is protection against asset theft when access is limited to just one key held by the owner or the guardian, this protection is not complete. It is so because there is no check on the gasLimit value. This creates a vulnerability where it is possible to force the paymaster to pay on behalf of the user for a great gasLimit, while the paymaster will not recognise the refund of unused reserved gas due to a lack of such logic in the bootloader.
Recommendation
Allow the execution of calls that require only one of the owner/guardian keys only through the “executeTransactionFromOutside” function.
Argent: Addressed.
We added checks that forbid using any paymaster in the escape methods.
0xRetrospective: Checked at commit f2f1ff05f670e0d7b6d52b4071ca6491e64e67e0.

- **I-01: Signatures of the message used in the “_validateNewOwner” function are reusable**

Severity: informational
Context: ArgentAccount.sol#L603-L604

Description
The signatures used in the "_validateNewOwner" function can be reused since they have no associated nonce or timestamp-related deadline.
Argent:
No changes were performed.
We believe the simple encoding is good enough for its purpose, which is preventing accidental mistakes when changing the owner to an address you don't control. The signature is not supposed to protect against bad actors.

- ### I-02: No possibility of a call with the isSystem flag to a contract different from the ContractDeployer

Severity: informational
Context: ArgentAccount.sol#L613-L624
Description
There is no possibility to call any system contract with the isSystemCall flag except the special calls checked in the "_execute" function. Such a possibility can be needed, for example, in the cases when some pre-signed transaction signatures were leaked and the account has a need to increment nonce by making a call to the NonceHolder contract.
Argent:
No changes were performed.
If we need to allow for that possibility we can ship a contract upgrade. Today we prefer to keep the functionality reduced for simplicity.
Regarding the example of leaked transaction signatures, the current contract allows us to deal with that. One way would be to submit an empty transaction to increase the nonce. Or alternatively, rotate the owner key to invalidate all older signatures.

- ### I-03: Incorrect check on the number of signatures in the "prepareForPaymaster" function

Severity: informational
Context: ArgentAccount.sol#L242-L245
Description
It is forbidden to use "approvalBased" paymaster flow to pay for execution of the transaction that requires only one signature. However, if the token allowance is already greater than or equal to the required value before calling the "prepareForPaymaster" function, this restriction should not apply.
Argent: Addressed.
We disabled using any kind of paymasters in escape transactions. We believe that even if the paymaster had an approved amount, the paymaster shouldn't be able to use that allowance for transactions submitted by one party alone.
0xRetrospective: Checked at commit f2f1ff05f670e0d7b6d52b4071ca6491e64e67e0.

- ### I-04: Phantom fallback function could be destructive for future versions of account abstraction

Severity: informational

Context: ArgentAccount.sol#L440-L445
Description
Any calls to the argent account that do not match the selectors of implemented functions are handled by the fallback function, which does not revert unless the bootloader is the caller. If future versions of account abstraction in Era introduce functions that are not called from the bootloader and do not expect any return data, this fallback function will be used. The success of calls of it may be incorrectly interpreted by the system, leading to potential issues.
Recommendation
Remove fallback function from the account implementation to rely solely on the receive function for accepting ETH.
Argent: Addressed.
Modified with the provided recommendation.
0xRetrospective: Checked at commit f2f1ff05f670e0d7b6d52b4071ca6491e64e67e0.

- ### I-05: Hardcoded amount of gas for calls of the “supportsInterface” function in the ERC165Checker library

Severity: informational
Context: ArgentAccount.sol#L26, ERC165Checker.sol#L119
Description
The ERC165Checker library uses a predefined amount of gas for calls of the “supportsInterface” function. Taking into account specific gas-related logic in Era, it will be safer to provide all available gas for such calls. Not providing enough gas could result in the argent account becoming unupgradeable, as the “upgrade” function relies on the stability of this functionality.
Recommendation
Provide all available gas for calls of the “supportsInterface” function.
Argent: Addressed.
Modified with the provided recommendation.
0xRetrospective: Checked at commit f2f1ff05f670e0d7b6d52b4071ca6491e64e67e0.
