ChainLight

ZKsync

zkSync Era Security Audit

：Bug Fixes Review Part 2

January 24, 2024

Revision 1.0

ChainLight@Theori

Table of Contents

Executive Summary

Starting on January 20, 2025, ChainLight of Theori reviewed the recent security bug findings and fixes in the zkSync Era circuits. The vulnerabilities were identified by Matter Labs and/or external reporters. The purpose of ChainLight's review was to identify the root cause, to analyze potential variants, and to review and suggest remediations. This review was a follow-up to a similar review performed on a different set of bugs in September 2024.

Audit Overview

Scope

Name	zkSync Era Security Audit
Target / Version	• Git Repository (matter-labs/zksync-protocol): diff `v0.150.20...protocol-v27-rc`, commits `07f7b8d8bddb7b168a8c1745c2b43e2b7b804c3e` to `d0b3ad21f4ab7474fdaba9bafb77fb9a69580fc8` • Git Repository (matter-labs/zksync-crypto): diff `v0.30.13...crypto-v27-rc`, commits `59146fc92571415abf2b62001b2bc43b1d04fe60` to `0291932b2c951d721506371519cd5aee4132fa64`
Application Type	ZK Circuit
Lang. / Platforms	ZK Circuit [Boojum]

Code Revision

N/A

Severity Categories

Severity	Description
Critical	The attack cost is low (not requiring much time or effort to succeed in the actual attack), and the vulnerability causes a high-impact issue. (e.g., Effect on service availability, Attacker taking financial gain)
High	An attacker can succeed in an attack which clearly causes problems in the service's operation. Even when the attack cost is high, the severity of the issue is considered "high" if the impact of the attack is remarkably high.
Medium	An attacker may perform an unintended action in the service, and the action may impact service operation. However, there are some restrictions for the actual attack to succeed.
Low	An attacker can perform an unintended action in the service, but the action does not cause significant impact or the success rate of the attack is remarkably low.
Informational	Any informational findings that do not directly impact the user or the protocol.
Note	Neutral information about the target that is not directly related to the project's safety and security.

Status Categories

Status	Description
Reported	ChainLight reported the issue to the client.
WIP	The client is working on the patch.
Patched	The client fully resolved the issue by patching the root cause.
Mitigated	The client resolved the issue by reducing the risk to an acceptable level by introducing mitigations.
Acknowledged	The client acknowledged the potential risk, but they will resolve it later.
Won't Fix	The client acknowledged the potential risk, but they decided to accept the risk.

Finding Breakdown by Severity

Category	Count	Findings
Critical	**0**	• N/A
High	**0**	• N/A
Medium	**2**	• `BUGFIX2-003` • `BUGFIX2-005`
Low	**1**	• `BUGFIX2-002`
Informational	**2**	• `BUGFIX2-001` • `BUGFIX2-004`
Note	**0**	• N/A

Findings

Summary

#	ID	Title	Severity	Status
1	BUGFIX2-001	Deleted the repetition in normalization	Informational	Reported
2	BUGFIX2-002	Fix overflow tracking in negated	Low	Reported
3	BUGFIX2-003	Remove EVM simulator gas stipend	Medium	Patched
4	BUGFIX2-004	Increase memory stipend for EVM calls	Informational	Patched
5	BUGFIX2-005	Fix in-circuit gas stipend passing	Medium	Patched

#1 `BUGFIX2-001` Deleted the repetition in normalization

ID	Summary	Severity
`BUGFIX2-001`	`NonNativeFieldOverU16::normalize` no longer repeats the `enforced_reduced` constraints	Informational

Description

`NonNativeFieldOverU16` provides an implementation of non-native field arithmetic where elements are represented with 16-bit limbs. To minimize unnecessary constraints, the module supports performing some arithmetic operations (e.g. `add`) without fully constraining the result to be in canonical form.

To enforce (i.e. add constraints) that an element is already in canonical form, users of the module can call `enforce_reduced`:

```
pub fn enforce_reduced<CS: ConstraintSystem<F>>(&mut self, cs: &mut CS) {
    assert_eq!(self.form, RepresentationForm::Normalized);
    if self.tracker.max_moduluses == 1 && self.form == RepresentationForm:
:Normalized {
        return;
    }
    // ...
    let modulus = self.
        .params
        .modulus
        .map(|el| cs.allocate_constant(F::from_u64_unchecked(el as u64)));
    let els_to_skip = N - self.params.modulus_limbs;
    let _ = u16_long_subtraction_noborrow_must_borrow(cs, &self.limbs, &mo
dulus, els_to_skip);
    self.tracker.max_moduluses = 1;
}
```

If the element is not yet in canonical form, users of the module can call `normalize` to produce a normalized and reduced representation:

```
pub fn normalize<CS: ConstraintSystem<F>>(&mut self, cs: &mut CS)
where
    [(); N + 1]:,
{
    if self.tracker.max_moduluses == 1 && self.form == RepresentationForm:
:Normalized {
        return;
    }
    // well, we just mul by 1
    let mut one: NonNativeFieldOverU16<F, T, N> =
        Self::allocated_constant(cs, T::one(), &self.params);
    let mut normalized = self.mul(cs, &mut one);
    // ...
    let modulus = self
        .params
        .modulus
        .map(|el| cs.allocate_constant(F::from_u64_unchecked(el as u64)));
    // for rare case when our modulus is exactly 16 * K bits, but we use l
arger representation
    let els_to_skip = N - self.params.modulus_limbs;
    let _ =
        u16_long_subtraction_noborrow_must_borrow(cs, &normalized.limbs, &
modulus, els_to_skip);
    assert!(normalized.form == RepresentationForm::Normalized);
    normalized.tracker.max_moduluses = 1;

    // update self to normalized one
    *self = normalized;
}
```

Notice that the full `enforced_reduce` logic is duplicated in `normalize`. The bulk of the logic to actually produce a normalized representation is in `mul`, which `normalize` leverages by multiplying the element by one. However, the `mul` function always calls `enforce_reduced` on the result before returning:

```
pub fn mul<CS: ConstraintSystem<F>>(&mut self, cs: &mut CS, other: &mut Se
lf) -> Self
where
```

```
    [(); N + 1]:,
{
    // ...
    let mut new = Self {
        limbs: r,
        non_zero_limbs: self.params.modulus_limbs,
        tracker: OverflowTracker {
            max_moduluses: self.params.max_mods_in_allocation,
        },
        form: RepresentationForm::Normalized,
        params: self.params.clone(),
        _marker: std::marker::PhantomData,
    };

    // enforce that r is canonical
    new.enforce_reduced(cs);

    new
}
```

As a result, the additional reduction checking constraints added by `normalize` are unnecessary and redundant.

Impact

Informational

This change does not fix a security issue. It simply removes some redundant constraints.

Recommendation

The change removes the additional constraints and simply returns the result after multiplying by one. This relies on the implementation of `mul` always returning fully reduced values. We suggest a more resilient change, which is to substitute the additional constraints with a call to `new.enforce_reduced(cs)`. This call will be a no-op on reduced values, but will be robust against future edits to `mul`.

References

N/A

Remediation

Reported

A patch removing the additional constraints from `normalize` was applied in commit `b03cc4fc67a84efe46104f9c885b8ad3cc3134cd` of matter-labs/zksync-crypto.

Our recommended fix has been reported to Matter Labs.

#2 `BUGFIX2-002` Fix overflow tracking in negated

ID	Summary	Severity
`BUGFIX2-002`	`NonNativeFieldOverU16::negated` incorrectly implements overflow tracking	Low

Description

`NonNativeFieldOverU16` performs static range tracking for each element, which is used both for runtime asserts and constraint optimizations. For instance, adding two elements should only require a gate count proportional to their number of nonzero limbs, so statically tracking the maximum value of each element allows omitting those unnecessary gates.

The static range tracking is implemented using `OverflowTracker`, which tracks the maximum value of element in terms of multiples of the modulus:

```
pub struct OverflowTracker {
    pub max_moduluses: u32,
}
```

`NonNativeFieldOverU16::negated` returns the additive inverse of a field element.

```
pub fn negated<CS: ConstraintSystem<F>>(&mut self, cs: &mut CS) -> Self
where
    [(); N + 1]:,
{
    let new = if self.form == RepresentationForm::Normalized {
        // ...
        // lazy path is not possible in this case
        let modulus_shifted = self
            .params
            .modulus_u1024
            .wrapping_mul(&U1024::from_word(self.tracker.max_moduluses as
u64));
        let used_words = self
            .tracker
```

```
            .used_words_if_normalized(self.params.modulus_u1024.as_ref());
        debug_assert!(used_words <= N);

        let modulus_words = u1024_to_u16_words::<N>(&modulus_shifted);
        let modulus_words =
            modulus_words.map(|el| cs.allocate_constant(F::from_u64_unchec
ked(el as u64)));
        let limbs =
            u16_long_subtraction_noborrow(cs, &modulus_words, &self.limbs,
N - used_words);

        let new = Self {
            limbs,
            non_zero_limbs: used_words,
            tracker: OverflowTracker { max_moduluses: 2 }, // NOTE: if sel
f == 0, then limbs will be == modulus, so use 2
            form: RepresentationForm::Normalized,
            params: self.params.clone(),
            _marker: std::marker::PhantomData,
        };

        new
    } else {
        // ...
    };
    // ...

    new
}
```

The above code returns an `OverflowTracker` with `max_moduluses` hardcoded to 2, which is only correct if the input was fully reduced. In reality, `negated` computes the result as

```
self.tracker.max_moduluses * self.params.modulus - self
```

which may be as large as `self.tracker.max_moduluses * self.params.modulus` if the input was the zero element.

Impact

Low

Since the overflow tracking is used for gate optimizations and overflow avoidance, this issue may have security impact in particular circumstances.

Recommendation

The following patch was provided for us to review:

```
@@ -764,7 +764,7 @@ where
            let new = Self {
                limbs,
                non_zero_limbs: used_words,
-               tracker: OverflowTracker { max_moduluses: 2 }, // NOTE: if
self == 0, then limbs will be == modulus, so use 2
+               tracker: self.tracker,
                form: RepresentationForm::Normalized,
                params: self.params.clone(),
                _marker: std::marker::PhantomData,
```

However, we suggest using the following patch instead:

```
@@ -764,7 +764,7 @@ where
            let new = Self {
                limbs,
                non_zero_limbs: used_words,
-               tracker: OverflowTracker { max_moduluses: 2 }, // NOTE: if
self == 0, then limbs will be == modulus, so use 2
+               tracker: OverflowTracker { max_moduluses: self.tracker.max_
moduluses + 1 },
                form: RepresentationForm::Normalized,
                params: self.params.clone(),
                _marker: std::marker::PhantomData,
```

The reason is that other code assumes that a normalized value with `max_moduluses = 1` means the value is reduced. For instance, `enforce_reduced` will not produce any constraints if these conditions are met:

```rust
pub fn enforce_reduced<CS: ConstraintSystem<F>>(&mut self, cs: &mut CS) {
    assert_eq!(self.form, RepresentationForm::Normalized);
    if self.tracker.max_moduluses == 1 && self.form == RepresentationForm:
:Normalized {
        return;
    }
    // ...
}
```

However, calling `negate` on the zero element will return the modulus, which is not reduced and thus should not have `max_moduluses == 1`.

References

N/A

Remediation

Reported

The change which returns `self.tracker` was applied in commit `59146fc92571415abf2b62001b2bc43b1d04fe60` of `matter-labs/zksync-crypto`.

Our recommended fix has been reported to Matter Labs.

#3 BUGFIX2-003 Remove EVM simulator gas stipend

ID	Summary	Severity
BUGFIX2-003	Removes the large gas stipend when calling the EVM simulator	Medium

Description

The EraVM supports configuring gas stipends which provide free gas to a callee that can not be returned to the caller if unused. These are primarily used to offset overhead introduced by EraVM system contracts, but were also briefly used for the EVM simulator.

The gas stipends are typically configured through a lookup table on the system contract address, but since the EVM simulator is not called via a system contract, its stipend required special logic to apply.

```
let evm_simulator_stipend =
    UInt32::allocated_constant(cs, zkevm_opcode_defs::system_params::EVM_S
IMULATOR_STIPEND);
let callee_stipend = UInt32::conditionally_select(
    cs,
    can_call_evm_simulator_without_masking,
    &evm_simulator_stipend,
    &callee_stipend,
);
```

The stipend was also unusually large, granting `2**30` gas to the EVM simulator contract.

```
pub const EVM_SIMULATOR_STIPEND: u32 = 1u32 << 30;
```

Combined with BUGFIX2-005, this large stipend could be abused by a malicious prover.

A commit was pushed which entirely removes the gas stipend for the EVM simulator. Note that it is replaced with a memory stipend in BUGFIX2-004.

Impact

Medium

Due to the large stipend, users could spam expensive EVM simulator calls in their transactions with minimal gas cost, potentially leading to denial of service. When combined with BUGFIX2-005, a malicious prover could use the large stipend to generate infinite gas within a transaction.

Recommendation

None

References

N/A

Remediation

Patched

The EVM simulator gas stipend was removed in commit `965841d3912d14a13600b2f399a661e0c1826b67` of matter-labs/zksync-protocol.

#4 BUGFIX2-004 Increase memory stipend for EVM calls

ID	Summary	Severity
BUGFIX2-004	Grants the EVM simulator a memory stipend when called	Informational

Description

The EraVM has two types of stipends: gas stipends and memory stipends. BUGFIX2-003 removed the gas stipend for the EVM simulator, while this change adds a memory stipend to EVM simulator calls.

Memory stipends are an amount of memory for which the callee can use before being charged memory growth costs. Before this change, all calls received a small memory stipend, while calls to system contracts received an extra memory stipend:

```
// 4 KB for new frames is "free"
pub const NEW_FRAME_MEMORY_STIPEND: u32 = 1u32 << 12;
// 2 MB for kernel frames, where we can be sure about the behavior.
// Note, that this number should high enough to allow any bytecode for `de
commit` opcode.
pub const NEW_KERNEL_FRAME_MEMORY_STIPEND: u32 = 1u32 << 21;
```

This change adds a new constant

```
// 56 KB for new EVM frames is "free"
pub const NEW_EVM_FRAME_MEMORY_STIPEND: u32 = 56 * 1u32 << 10;
```

which is used as the memory stipend when calling the EVM simulator.

The logic for selecting the memory stipend in the out-of-circuit `zk_evm` is now as follows:

```
let memory_stipend = if address_is_kernel(&address_for_next) {
    zkevm_opcode_defs::system_params::NEW_KERNEL_FRAME_MEMORY_STIPEND
} else {
    if call_to_evm_simulator {
```

```
            zkevm_opcode_defs::system_params::NEW_EVM_FRAME_MEMORY_STIPEND
        } else {
            zkevm_opcode_defs::system_params::NEW_FRAME_MEMORY_STIPEND
        }
    };
```

Equivalent logic is implemented in the circuits to select the appropriate memory stipend.

Impact

Informational

This bug adds additional memory stipend for EVM simulator calls, but the amount is still relatively small and thus does not introduce significant risk of abuse.

Recommendation

None

References

N/A

Remediation

Patched

This change was applied in commit `965841d3912d14a13600b2f399a661e0c1826b67` of matter-labs/zksync-protocol.

#5 BUGFIX2-005 Fix in-circuit gas stipend passing

ID	Summary	Severity
BUGFIX2-005	Gas stipends were not being assigned to the new callee frame in the circuits	Medium

Description

The EraVM supports configuring gas stipends which provide free gas to a callee that can not be returned to the caller if unused. These are primarily used to offset overhead introduced by EraVM system contracts or the EVM simulator. The stipend logic happens in two parts:

1. During the call, the stipend amount is computed and added to the gas amount to be passed to the callee
2. During the return, the stipend is subtracted from the remaining gas before being returned to the caller, masking it to zero if the result was negative.

After BUGFIX2-003, the EVM simulator stipend is removed, so the circuits once again always determine gas stipends by performing a lookup on the callee address:

```
let [callee_stipend, extra_ergs_from_caller_to_callee] =
cs.perform_lookup::<1, 2>(table_id, &[address_low_masked.get_variable()]);
// ...
let callee_stipend = unsafe { UInt32::from_variable_unchecked(callee_stipe
nd) };
// ...
let passed_ergs_if_pass = passed_ergs_if_pass.add_no_overflow(cs, callee_s
tipend);
```

Although the stipend was correctly increasing the gas passed to the callee, the stipend was never assigned to the callee's stack frame object. As a result, the leftover stipend would incorrectly be returned to the caller. An attacker could abuse this to generate infinite gas by repeatedly extracting the leftover stipend.

The issue was fixed by simply assigning the stipend into the new stack frame during a far call:

```
// stipend
new_callstack_entry.stipend = callee_stipend;
```

Impact

Medium

The impact of this bug is limited because the out-of-circuit implementation correctly tracked the stipend. As a result, the witness data for an exploit transaction would have to be specially crafted and proved by a malicious prover.

The impact also depends on the size of the stipends. If the full stipends are guaranteed to be consumed by the callee, it would be impossible to extract gas through this bug. Before BUGFIX2-003, the large EVM simulator stipend could likely be partially extracted, which increases the severity of this bug.

Recommendation

The provided fix fully addresses this bug. We have no additional recommendations.

References

N/A

Remediation

Patched

The issue was fixed in commit `965841d3912d14a13600b2f399a661e0c1826b67` of matter-labs/zksync-protocol.

Revision History

Version	Date	Description
.0	January 24, 2024	Initial version

Theori, Inc. (“We”) is acting solely for the client and is not responsible to any other party. Deliverables are valid for and should be used solely in connection with the purpose for which they were prepared as set out in our engagement agreement. You should not refer to or use our name or advice for any other purpose. The information (where appropriate) has not been verified. No representation or warranty is given as to accuracy, completeness or correctness of information in the Deliverables, any document, or any other information made available. Deliverables are for the internal use of the client and may not be used or relied upon by any person or entity other than the client. Deliverables are confidential and are not to be provided, without our authorization (preferably written), to entities or representatives of entities (including employees) that are not the client, including affiliates or representatives of affiliates of the client.