hexens × polygon

NOV.24

SECURITY REVIEW REPORT FOR POLYGON

CONTENTS

- About Hexens
- Executive summary
  - Scope
- Auditing details
- Severity structure
  - Severity characteristics
  - Issue symbolic codes
- Findings summary
- Weaknesses
  - Inconsistency between Arithmetic SM Equation Selector Comment and Actual Values
  - ROM PIL Documentation Misses assumeFree Operation

ABOUT HEXENS

Hexens is a cybersecurity company that strives to elevate the standards of security in Web 3.0, create a safer environment for users, and ensure mass Web 3.0 adoption.

Hexens has multiple top-notch auditing teams specialized in different fields of information security, showing extreme performance in the most challenging and technically complex tasks, including but not limited to: Infrastructure Audits, Zero Knowledge Proofs / Novel Cryptography, DeFi and NFTs. Hexens not only uses widely known methodologies and flows, but focuses on discovering and introducing new ones on a day-to-day basis.

In 2022, our team announced the closure of a $4.2 million seed round led by IOSG Ventures, the leading Web 3.0 venture capital. Other investors include Delta Blockchain Fund, Chapter One, Hash Capital, ImToken Ventures, Tenzor Capital, and angels from Polygon and other blockchain projects.

Since Hexens was founded in 2021, it has had an impressive track record and recognition in the industry: Mudit Gupta - CISO of Polygon Technology - the biggest EVM Ecosystem, joined the company advisory board after completing just a single cooperation iteration. Polygon Technology, 1inch, Lido, Hats Finance, Quickswap, Layerswap, 4K, RociFi, as well as dozens of DeFi protocols and bridges, have already become our customers and taken proactive measures towards protecting their assets.

SCOPE

The analyzed resources are located on:

https://github.com/0xPolygonHermez/zkevm-proverjs/pull/361

https://github.com/0xPolygonHermez/zkasmcom/pull/105

The issues described in this report were fixed in the following commit:

https://github.com/0xPolygonHermez/zkevm-proverjs/pull/367/commits/5aba3eb2d56367d6aee6daa1077d81131a820eda

AUDITING DETAILS

STARTED

18.11.2024

DELIVERED

10.12.2024

Review Led by

HAYK ANDRIASYAN

Senior Security Researcher | Hexens

HEXENS METHODOLOGY

Hexens methodology involves 2 teams, including multiple auditors of different seniority, with at least 5 security engineers. This unique cross-checking mechanism helps us provide the best quality in the market.

SEVERITY STRUCTURE

The vulnerability severity is calculated based on two components

- Impact of the vulnerability
- Probability of the vulnerability

Impact	Probability			
	rare	unlikely	likely	very likely
Low/Info	Low/Info	Low/Info	Medium	Medium
Medium	Low/Info	Medium	Medium	High
High	Medium	Medium	High	Critical
Critical	Medium	High	Critical	Critical

SEVERITY CHARACTERISTICS

Smart contract vulnerabilities can range in severity and impact, and it's important to understand their level of severity in order to prioritize their resolution. Here are the different types of severity levels of smart contract vulnerabilities:

Critical

Vulnerabilities with this level of severity can result in significant financial losses or reputational damage. They often allow an attacker to gain complete control of a contract, directly steal or freeze funds from the contract or users, or permanently block the functionality of a protocol. Examples include infinite mints and governance manipulation.

High

Vulnerabilities with this level of severity can result in some financial losses or reputational damage. They often allow an attacker to directly steal yield from the contract or users, or temporarily freeze funds. Examples include inadequate access control integer overflow/underflow, or logic bugs.

Medium

Vulnerabilities with this level of severity can result in some damage to the protocol or users, without profit for the attacker. They often allow an attacker to exploit a contract to cause harm, but the impact may be limited, such as temporarily blocking the functionality of the protocol. Examples include uninitialized storage pointers and failure to check external calls.

Low

Vulnerabilities with this level of severity may not result in financial losses or significant harm. They may, however, impact the usability or reliability of a contract. Examples include slippage and front-running, or minor logic bugs.

Informational

Vulnerabilities with this level of severity are regarding gas optimizations and code style. They often involve issues with documentation, incorrect usage of EIP standards, best practices for saving gas, or the overall design of a contract. Examples include not conforming to ERC20, or disagreement between documentation and code.

ISSUE SYMBOLIC CODES

Every issue being identified and validated has its unique symbolic code assigned to the issue at the security research stage. Cause of the vulnerability reporting flow design, some of the rejected issues could be missing.

FINDINGS SUMMARY

Severity	Number of Findings
Critical	0
High	0
Medium	0
Low	0
Informational	2

Total: 2

Informational

Fixed

WEAKNESSES

This section contains the list of discovered weaknesses.

ZKEPOL-2

INCONSISTENCY BETWEEN ARITHMETIC SM EQUATION SELECTOR COMMENT AND ACTUAL VALUES

SEVERITY: Informational

PATH:

tools/arith/arith.ejs.pil

pil/arith.pil

REMEDIATION:

Fix selEq values in the comment for EQ5-EQ14.

STATUS: Fixed

DESCRIPTION:

The Arithmetic State Machine handles Elliptic Curve operations. Polygon zkEVM supports several curves: **Secp256k1**, **BN254**, **Secp256r1**. Arith SM has an equation selector to choose the curve:

```
pol sel_secp256k1 = selEq[1] + selEq[2];
pol sel_bn254 = selEq[3] + selEq[4] + selEq[5];
pol sel_secp256r1 = selEq[6] + selEq[7];
```

The selector equation for **BN254**, **Secp256r1** curves in the comment(**selEq** column) mismatches from the PIL one:

```
/*
    Equations:
                                                                    selEq   arithEq
    EQ0:   A(x1) * B(y1) + C(x2) = D (y2) * 2 ** 256 + op (y3)        0         1    ARITH

    EQ1:   s * x2 - s * x1 - y2 + y1 + (q0 * p1)    lambda - ADD      1         2    ARITH_ECADD_DIFFERENT
    EQ2:   2 * s * y1 - 3 * x1 * x1 + (q0 * p1)     lambda - DBL      2         3    ARITH_ECADD_SAME
    EQ3:   s * s - x1 - x2 - x3 + (q1 * p1)         x3              1+2       2,3    ARITH_ECADD_DIFFERENT, ARITH_ECADD_SAME
    EQ4:   s * x1 - s * x3 - y1 - y3 + (q2 * p1)    y3              1+2       2,3    ARITH_ECADD_DIFFERENT, ARITH_ECADD_SAME

    EQ5:   x1 * x2 - y1 * y2 - x3 + (q1 * p2)                         4         4    ARITH_BN254_MULFP2
    EQ6:   y1 * x2 + x1 * y2 - y3 + (q2 * p2)                         4         4    ARITH_BN254_MULFP2

    EQ7:   x1 + x2 - x3 + (q1 * p2)                                   5         5    ARITH_BN254_ADDFP2
    EQ8:   y1 + y2 - y3 + (q2 * p2)                                   5         5    ARITH_BN254_ADDFP2

    EQ9:   x1 - x2 - x3 + (q1 * p2)                                   6         6    ARITH_BN254_SUBFP2
    EQ10:  y1 - y2 - y3 + (q2 * p2)                                   6         6    ARITH_BN254_SUBFP2

    EQ11:  s * x2 - s * x1 - y2 + y1 + (q0 * p3)    lambda - ADD      7         7    ARITH_SECP256R1_ECADD_DIFFERENT
    EQ12:  2 * s * y1 - 3 * x1 * x1 - a + (q0 * p3) lambda - DBL      8         8    ARITH_SECP256R1_ECADD_SAME
    EQ13:  s * s - x1 - x2 - x3 + (q1 * p3)         x3              7+8       7,8    ARITH_SECP256R1_ECADD_DIFFERENT,
ARITH_SECP256R1_ECADD_SAME
    EQ14:  s * x1 - s * x3 - y1 - y3 + (q2 * p3)    y3              7+8       7,8    ARITH_SECP256R1_ECADD_DIFFERENT,
ARITH_SECP256R1_ECADD_SAME
    ...
*/
```

ROM PIL DOCUMENTATION MISSES ASSUMEFREE OPERATION

SEVERITY: Informational

PATH:

pil/rom.pil

REMEDIATION:

Include assumeFree in the operations polynomial.

STATUS: Fixed

DESCRIPTION:

Rom SM contains an **operations** constant polynomial which is constructed in the Executor. The **operations** polynomial doesn't include **assumeFree** in the generated comment.

```
/*
  comment genereated with:
  node tools/pil_pol_table/bits_compose.js
"arith,arithSame12,arithUseE,assert,bin,hashK,hashKDigest,hashKLen,hashP,hashPDigest,hashPLen,ind,indRR,isMem
,isStack,JMP,JMPC,JMPN,memAlignRD,memAlignWR,memAlignWR8,mOp,mWR,repeat,setA,setB,setC,setCTX,setD,setE,setGA
S,setHASHPOS,setPC,setRCX,setRR,setSP,setSR,sRD,sWR,useCTX,useJmpAddr,JMPZ,call,return,hashK1,hashP1,useElseA
ddr,hashS,hashSDigest,hashSLen,hashS1"

  operations =
          2**0  * arith         + 2**1  * arithSame12   + 2**2  * arithUseE     + 2**3  * assert
        + 2**4  * bin           + 2**5  * hashK         + 2**6  * hashKDigest   + 2**7  * hashKLen
        + 2**8  * hashP         + 2**9  * hashPDigest   + 2**10 * hashPLen      + 2**11 * ind
        + 2**12 * indRR         + 2**13 * isMem         + 2**14 * isStack       + 2**15 * JMP
        + 2**16 * JMPC          + 2**17 * JMPN          + 2**18 * memAlignRD    + 2**19 * memAlignWR
        + 2**20 * memAlignWR8   + 2**21 * mOp           + 2**22 * mWR           + 2**23 * repeat
        + 2**24 * setA          + 2**25 * setB          + 2**26 * setC          + 2**27 * setCTX
        + 2**28 * setD          + 2**29 * setE          + 2**30 * setGAS        + 2**31 * setHASHPOS
        + 2**32 * setPC         + 2**33 * setRCX        + 2**34 * setRR         + 2**35 * setSP
        + 2**36 * setSR         + 2**37 * sRD           + 2**38 * sWR           + 2**39 * useCTX
        + 2**40 * useJmpAddr    + 2**41 * JMPZ          + 2**42 * call          + 2**43 * return
        + 2**44 * hashK1        + 2**45 * hashP1        + 2**46 * useElseAddr   + 2**47 * hashS
        + 2**48 * hashSDigest   + 2**49 * hashSLen      + 2**50 * hashS1;

*/
```

hexens × polygon