Contract Audit Results

Prepared on: Mar 26 2022

Contract: AUD461

Prepared by:

Charles Holtzkampf

Sentnlio Ltd

Prepared for:

Joe Andrews

Spilsbury Holdings Ltd

Table of Contents

Executive Summary

This document outlines any issues found during the audit of the following code

- https://github.com/AztecProtocol/barretenberg

The fuzzing corpus derived from existing Cryptofuzz-based OSS-Fuzz corpora, can be found here.

https://github.com/guidovranken/aztec-audit/tree/master/oss-fuzz

Severity Description

REMARK

Remarks are instances in the code that are worthy of attention, but in no way represent a security flaw in the code. These issues might cause problems with the user experience, confusion with new developers working on the project, or other inconveniences.

Things that would fall under remarks would include:

- Instances where best practices are not followed
- Spelling and grammar mistakes
- Inconsistencies in the code styling and structure

MINOR

Issues of Minor severity can cause problems in the code, but would not cause the code to crash unexpectedly or for funds to be lost. It might cause results that would be unexpected by users, or minor disruptions in operations. Minor problems are prone to become major problems if not addressed appropriately.

Things that would fall under minor would include:

- Logic flaws (excluding those that cause crashes or loss of funds)
- Code duplication
- Ambiguous code

MAJOR

Issues of major security can cause the code to crash unexpectedly, or lead to deadlock situations.

Things that would fall under major would include:

- Logic flaws that cause crashes
- Timeout exceptions
- Incorrect ABI file generation
- Unrestricted resource usage (for example, users can lock all RAM on contract)

CRITICAL

Critical issues cause a loss of funds or severely impact contract usage.

Things that would fall under critical would include:

- Missing checks for authorization
- Logic flaws that cause loss of funds
- Logic flaws that impact economics of system
- All known exploits (for example, on_notification fake transfer exploit)

Methodology

Differential fuzzing

During the differential fuzzingOutput of Barretenberg primitives we compared against relevant other libraries which implemented the same primitives (differential fuzzing) and which are known to be bug-free, such as Botan.

We looked for:

1. Memory bugs
2. Undefined behavior bugs
3. Incorrect results from cryptographic and mathematical primitive implementations
4. Unexpected hangs or excessive slowness
5. Other remarks that are relevant to the security and soundness of your code

Audit Results - AztecProtocol - barretenberg

uint256_t left-shift incorrect result

```
#include <numeric/uint256/uint256.hpp>

int main(void)
{
    uint256_t base(1);

    uint256_t shift((uint64_t)(-1));
    shift = shift + 1;
    /* 'shift' is now 2**64 */

    /* Shift count is so large that this ought to set 'base' to 0 */
    base <<= shift;

    /* Shouldn't abort */
    if (base) abort();

    return 0;
}
```

uint512_t invmod incorrect result

```
#include <numeric/uintx/uintx.hpp>
#include <cstring>

int main(void)
{
```

```
    const uint8_t _a[] = {0x03, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x28, 0x0D, 0x6A, 0x2B, 0x19,
0x52, 0x2D, 0xF7, 0xAF, 0xC7, 0x95, 0x68, 0x22, 0xD7, 0xF2, 0x21, 0xA3, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00};

    const uint8_t _b[] = {0xFF, 0x00, 0xFF, 0xFF, 0x00, 0xFF, 0xFF, 0xFF, 0xFF, 0x5D, 0x32, 0xDA, 0x10,
0x4F, 0x1D, 0xD6, 0xCA, 0x50, 0x56, 0x11, 0x18, 0x18, 0xC2, 0xD4, 0x6C, 0x70, 0x60, 0xD9, 0xB8,
0xFA, 0x10, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0xE2, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x01, 0xFF, 0xFF,
0xFF, 0xFF};

    uint8_t _res[64];

    uint512_t a, b;

    memcpy(a.lo.data, _a, 32);
    memcpy(a.hi.data, _a + 32, 32);

    memcpy(b.lo.data, _b, 32);
    memcpy(b.hi.data, _b + 32, 32);
    const auto res = a.invmod(b);

    memcpy(_res, res.lo.data, 32);
    memcpy(_res + 32, res.hi.data, 32);

    for (size_t i = 0; i < 64; i++) {
        printf("0x%02X, ", _res[i]);
        if ( i == 31 ) printf("\n");
    }
    printf("\n");

    /* This should print:
```

```
    * 0x9F, 0x2F, 0xAA, 0x7B, 0xD7, 0x5A, 0x99, 0x56, 0x04, 0x68, 0x6C, 0x9D, 0xD8, 0x47, 0x6B,
0x52, 0xF0, 0x10, 0xD2, 0xA8, 0x62, 0x96, 0x60, 0x68, 0xBE, 0x18, 0x21, 0xA1, 0xCA, 0x6F, 0x41,
0x9C,
    * 0x37, 0x42, 0x2F, 0xA3, 0x1B, 0x41, 0x7B, 0xAA, 0xEE, 0x6D, 0x9E, 0x03, 0x78, 0x71, 0xEF,
0xCF, 0x90, 0x85, 0xEF, 0x17, 0x59, 0xC4, 0xEE, 0x24, 0x80, 0xDE, 0x7A, 0x58, 0xA5, 0x42, 0x8F,
0x97,
    *
    * This is the value
793782089846612129258981008368609936170396868496261702412847293142876897201901508577183341626972762676420213785897546421442791014271577186631976640749148
7
.
    */

   /* But it does print:
    * 0xA0, 0x2E, 0xAB, 0x7B, 0xD6, 0x5B, 0x99, 0x56, 0x04, 0x0A, 0x3A, 0xC3, 0xC7, 0xF8, 0x4D,
0x7C, 0x25, 0xC0, 0x7B, 0x97, 0x4A, 0x7E, 0x9E, 0x93, 0x51, 0xA8, 0xC0, 0xC7, 0x11, 0x75, 0x30,
0x9C,
    * 0x37, 0x42, 0x2F, 0xA3, 0x1B, 0x41, 0x7B, 0xAA, 0xEE, 0x6D, 0x9E, 0x03, 0x78, 0x8F, 0xEF,
0xCF, 0x90, 0x85, 0xEF, 0x17, 0x59, 0xC4, 0xEE, 0x24, 0x80, 0xDE, 0x7A, 0x56, 0xA6, 0x42, 0x8F,
0x97,
    *
    * This is the value
793782090156348118235645862576002786313937594940383172701887083185625667157742439915257293895383236432428719224324885111200123086599831888286953875827472
0
.
    */
   return 0;
}
```

Audit Results - AztecProtocol - barretenberg

uint256_t pow(0,0) returns 0

```
#include <numeric/uintx/uintx.hpp>

int main(void)
{
    uint256_t a, b;
    const uint256_t res = a.pow(b);
    if ( !res ) abort();
    return 0;
}
```

uint256_t/uint512_t: get_bit() out-of-bounds access

uint256_t/uint512_t: get_bit() out-of-bounds access

Audit Results - AztecProtocol - barretenberg

Point multiplication gives incorrect result

BN254 G2:

operation name: BLS_G2_Mul
A V:71614438161550543583343459123265558553665399546143750564488807827019051362 36
A W:21695569177883476650401062347755319409152628066914457580714394817998203895 1
A X:90709371284817270624447909267555783742296645428132714238482642663543536970 99
A Y:18067648048954065759777413076841546109352031352040422612871782226365138778 120
B: 34161158157884809629024386706894198681 5

Module barretenberg result:

X1: 15603546064783771975121431745144972780129806382883066895872885632002132727168
Y1: 15066491740135207838157090205891577487146437977743718736013244763554764282805
X2: 21190712983968128683531149673043778124772629956042023240752638452933620598612
Y2: 11567432026911906718739009789622509446871024167159123228431769613819033977746

Module libff result:

X1: 53551120841472885538215155252317820491579616536647941304288545708829182322 78
Y1: 82072947479056881020792533100034717342434046249864122643443819849887357736 14
X2: 33380111676712087220404568950744902688729830014712495710669253319547783804 76
Y2: 97679200563041890021387279275925162722756255753104793954453418906554359077 8

secp256k1:

operation name: ECC_Point_Mul
ecc curve: secp256k1
A X: 10920343075644590967038793131241799923271928537885362519163460169796991216889
A Y: 12999295057502890205958304945545915979655741111594184729668511776981844173 95

B: 115792089237316195423570985008687907853269984665640564039457584007908834671663

Module barretenberg result:

X: 106828507449787920475367839486311366887212143134295863212469878627934313457917
Y: 173139733909635744154100751212570873377799659533936819808977795769488441432 09

Module Botan result:

X: 189897462403993924948662591070293228666558184882645614324636166248190536671 56
Y: 213707749097770478055103722640516815028221432448532233435328378869774847089

secp256r1:

operation name: ECC_Point_Mul
ecc curve: secp256r1
A X: 37408247098253683891614493016632670971377112617664302694927749692224160831905
A Y: 99328697488960988042324003589590014826854511518270066810683193433285395680265
B: 87278832398428449337322202511838003950753763815270500558 35

Module barretenberg result:

X: 37593882894416784751810999564107912912186467273622597807602731175846965794659
Y: 34450860751163548340468118909460707102538815604199446439705209830624333980486

Module Botan result:

X: 10653045363433063640777889731134517479331704796168833315102501217190451056944 7
Y: 60336318664742709241086915231502684148752509534027212339939445824785463386468

Inadequate point validation

X: 0

Y: 69528327468847610065686496900697922508397251637412376320436699849860351814667

For this point on the secp256r1 curve, barretenberg's (point->on_curve() && !point->is_point_at_infinity()) returns false but the other libraries return true.

Does not affect secp256k1 because no point whose X = 0 exists on secp256k1.

At AztecProtocol/aztec2-internal@163f891 the following discrepancy still exists:

Operation:
operation name: ECC_ValidatePubkey
ecc curve: secp256k1
public key X: 115792089237316195423570985008687907853269984665640564039457584007913129571663
public key Y: 42421176211404854345735992948420356965137943451922558217513701442576906664822

Module barretenberg result:

true

Module secp256k1 result:

false

This is the same point as

X: 4294900000
Y: 42421176211404854345735992948420356965137943451922558217513701442576906664822

which is a valid pubkey, but because the secp256k1 prime has been added to the X coordinate, it is no longer valid.

To fix, reject coordinates which are >= curve prime.

Audit Results - AztecProtocol - barretenberg

Memory leak in Pippenger

An allocation is made here:

https://github.com/AztecProtocol/barretenberg/blob/38c8b72e633f58f4e394990da1936ed5fa2ab1ee/barretenberg/src/aztec/ecc/curves/bn254/scalar_multiplication/scalar_multiplication.cpp#L733

If subsequently an exception is thrown here:

https://github.com/AztecProtocol/barretenberg/blob/38c8b72e633f58f4e394990da1936ed5fa2ab1ee/barretenberg/src/aztec/ecc/curves/bn254/scalar_multiplication/scalar_multiplication.cpp#L318

the memory is not freed resulting in a memory leak.

std::runtime_error 'Trying to invert zero in the field' when operating on invalid points

As of https://github.com/AztecProtocol/aztec2-internal/commit/163f89156be4cb4a3a0aa8dee48ea38534c57a05 I'm finding various cases of Barretenberg throwing an std::runtime exception with the message Trying to invert zero in the field when an invalid (not on curve) point is constructed and operated on.

Here is an example which causes the ECDSA verification function to throw this exception when a particular public key is passed to it:

```
#include <crypto/ecdsa/ecdsa.hpp>

#include <ecc/curves/secp256k1/secp256k1.hpp>



static void ecdsa_verify(void) {

    const secp256k1::g1::affine_element pub(uint256_t(2), uint256_t(3));

    const crypto::ecdsa::signature sig{{}, {}};
```

```
    const std::string msg("abc");

    crypto::ecdsa::verify_signature<Sha256Hasher, secp256k1::fq, secp256k1::fr, secp256k1::g1>(msg, pub, sig);

}



int main(void)

{

    ecdsa_verify();

    return 0;

}
```

Audit Results - AztecProtocol - barretenberg

ECDSA verification succeeds when it should fail

The proof of concept below results in positive ECDSA verification, but verification should fail because R < 1. The verifier should reject all signatures whose R or S values are 0.

This uses the "NullHasher" which is not used in your library, but which I'm using in my fuzzer to make it easier to find corner cases like this one.

```
#include <crypto/ecdsa/ecdsa.hpp>
#include <ecc/curves/secp256r1/secp256r1.hpp>

struct NullHasher {
    static std::vector<uint8_t> hash(const std::vector<uint8_t>& message) {
        return message;
    }
};

int main(void)
{
    uint256_t x, y;
    const uint8_t x_bytes[] = {0x2A, 0xE5, 0xCF, 0x50, 0xA9, 0x31, 0x64, 0x23, 0xE1, 0xD0, 0x66, 0x32, 0x65, 0x32,
0xF6, 0xF7, 0xEE, 0xEA, 0x6C, 0x46, 0x19, 0x84, 0xC5, 0xA3, 0x39, 0xC3, 0x3D, 0xA6, 0xFE, 0x68, 0xE1, 0x1A};
    memcpy(x.data, x_bytes, 32);

    const uint8_t y_bytes[] = {0xD1, 0x73, 0x78, 0x22, 0x9D, 0xB7, 0x04, 0x9E, 0x29, 0x82, 0xE9, 0x3C, 0xE6, 0xAD,
0x7D, 0xBA, 0xDB, 0x30, 0x74, 0x9F, 0xC6, 0x9A, 0x3D, 0x29, 0x40, 0xD0, 0x8E, 0xDB, 0x10, 0x55, 0x77, 0x07};
    memcpy(y.data, y_bytes, 32);

    const secp256r1::g1::affine_element pub(x, y);

    const std::array<uint8_t, 32> r{0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
```

```
0x00};
    const std::array<uint8_t, 32> s{0xF2, 0x1A, 0x23, 0x9F, 0x35, 0x7B, 0x47, 0x5C, 0x84, 0x2E, 0x7D, 0x22, 0xFB,
0x24, 0xFF, 0xC2, 0xD2, 0x59, 0xF1, 0xB7, 0x19, 0x72, 0x12, 0x28, 0xAF, 0x4E, 0x6E, 0xCA, 0x0C, 0xD5, 0x68,
0x7A};

    const crypto::ecdsa::signature sig{r, s};

    const std::string msg(32, 0);
    printf("Verified: %d\n", crypto::ecdsa::verify_signature<NullHasher, secp256r1::fq, secp256r1::fr,
secp256r1::g1>(msg, pub, sig));
    return 0;
}
```

Here is a more readable rendition of this bug, with decimal coordinate values:

Operation:
operation name: ECDSA_Verify
ecc curve: secp256r1
public key X: 121583992996938303229678086127133986361553678870416281767988719547883716539 30
public key Y: 337703184371225825922371145149145259808867551975154856711245809463549758356 9
msg: {} (this is padded to 32 zero bytes)
signature R: 0
signature S: 109505893234220627122504643420391275575463182862784447225831800231103487633530
digest: NULL

Module barretenberg result:

true

Module Botan result:

false

Incorrect field square root computation with assembly language optimizations

The square root of *6647964875202594977810710751927464325147118738437648596507388215053719738179* is *8047886942884762300879823309598152162393387115659347484374289332482298920315*, but barretenberg reports that it is *120498597390623964243429023232219200937421838935101237872635641703574590328 1*.

Python code which proves that barretenberg is wrong:

```
mod = 21888242871839275222246405745257275088696311157297823662689037894645226208583
sqr = 6647964875202594977810710751927464325147118738437648596507388215053719738179
sqrt_botan = 8047886942884762300879823309598152162393387115659347484374289332482298920315
sqrt_barretenberg =
120498597390623964243429023232219200937421838935101237872635641703574590328 1

# This succeeds
assert(
    ((sqrt_botan*sqrt_botan) % mod) == sqr
)

# This fails
assert(
    ((sqrt_barretenberg*sqrt_barretenberg) % mod) == sqr
)
```

Reproducer:

```
#include <ecc/curves/bn254/g1.hpp>
#include <iostream>

int main(void)
{
   const uint8_t v_bytes[] = {0x43, 0x97, 0xF8, 0xB4, 0x0F, 0xDA, 0x8D, 0xEA, 0x5E, 0x53, 0xF8, 0x55, 0x06, 0xFE,
0x1C, 0x66, 0xF3, 0x57, 0x4A, 0x43, 0xE1, 0x52, 0x9D, 0x11, 0xAF, 0xF6, 0xEA, 0x0B, 0x62, 0x9D, 0xB2, 0x0E};
   uint256_t v;
   memcpy(v.data, v_bytes, 32);
```

```
    ::barretenberg::fq fq(v);
    std::cout << fq.sqrt() << std::endl;
    return 0;
}
```

Compile the reproducer with -D__BMI2__

It will print
0x02a9ff75dbeac3a24839f6ce4c67f8e8ce03853e41e0015b65241d0c296042b1

which is hex for
120498597390623964243429023232219200937421838935101237872635641703574590328 1

When compiled without -D__BMI2__, it prints
0x11caf14d7c60c355c12a090e491a811416c0058789751628f6feceb6bb00bd7b

which is hex for
804788694288476230087982330959815216239338711565934748437428933248229892031 5

Audit Results - AztecProtocol - barretenberg

Valid and invalid points regarded as equal

```
#include <crypto/ecdsa/ecdsa.hpp>
#include <ecc/curves/secp256k1/secp256k1.hpp>

int main(void)
{
    const secp256k1::g1::element point = secp256k1::g1::one;

    const auto point_mul_0 = point * 0;

    if ( point == point_mul_0 ) {
        std::cout << "points are regarded as equivalent despite being different:" << std::endl;
        std::cout << point << std::endl;
        std::cout << point_mul_0 << std::endl;
    }

    return 0;
}
```

This prints:

```
points are regarded as equivalent despite being different:
{ 0x79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798,
0x483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8,
0x0000000000000000000000000000000000000000000000000000000000000001 }
{ 0x0000000000000000000000000000000000000000000000000000000000000000,
0x0000000000000000000000000000000000000000000000000000000000000000,
0x0000000000000000000000000000000000000000000000000000000000000000 }
```

Tested at: https://github.com/AztecProtocol/aztec2-internal/commit/163f89156be4cb4a3a0aa8dee48ea38534c57a05

I think this happens because in element<Fq, Fr, T>::operator==, other.z is zero, and by using it as a multiplier, zeroes are propagated to the other values, leading to all 4 Fq values being zero at this line:

https://github.com/AztecProtocol/aztec2-internal/blob/e7281fad63b1a717a6d938c3422d567399b020f3/barretenberg/src/aztec/ecc/groups/element_impl.hpp#L550

hence returning true.

Addition of invalid points creates valid point

```
#include <ecc/curves/secp256k1/secp256k1.hpp>

int main(void)
{
    const secp256k1::g1::affine_element p1(0, 0);
    const secp256k1::g1::affine_element p2(0, 0);
    const auto r = static_cast<secp256k1::g1::element>(p1) + static_cast<secp256k1::g1::element>(p2);

    std::cout << "P1: " << p1.on_curve() << ", " << p1.is_point_at_infinity() << std::endl;
    std::cout << "P2: " << p2.on_curve() << ", " << p2.is_point_at_infinity() << std::endl;
    std::cout << "r: " <<  r.on_curve()  << ", " << r.is_point_at_infinity()  << std::endl;
    std::cout << r << std::endl;

    return 0;
}
```

Prints

```
P1: 0, 0
P2: 0, 0
r: 1, 0
```

Results

{ 0x0000000000000000000000000000000000000000000000000000000000000000,
0x0000000000000000000000000000000000000000000000000000000000000000,
0x0000000000000000000000000000000000000000000000000000000000000000 }

Audit Results - AztecProtocol - barretenberg

conditional_negate_affine does not negate point if predicate is any value other than 1

```
#include <ecc/curves/bn254/g1.hpp>
#include <iostream>

int main(void)
{

    barretenberg::g1::affine_element p(
        barretenberg::fq(uint256_t(1)),
        barretenberg::fq(uint256_t(2))
    );

    const auto expected = -(barretenberg::g1::affine_element(
        barretenberg::fq(uint256_t(1)),
        barretenberg::fq(uint256_t(2))
    ));

    ::barretenberg::g1::conditional_negate_affine(&p, &p, 2);

    assert(p == expected);

    return 0;
}
```

The expected behavior is that it inverts the point for every non-zero predicate.

Out-of-bounds access in mul_with_endomorphism

To reproduce:

1. Get clang 14: https://apt.llvm.org/
2. Clone https://github.com/AztecProtocol/aztec2-internal/ and checkout defi-bridge-project
3. Compile the reproducer (below) using: clang++-14 -std=c++20 -O3 -I aztec2-internal/barretenberg/src/aztec mul_with_endomorphism_oob.cpp
4. Run the executable
5. Output is: Bus error (core dumped)

```
#include <ecc/curves/bn254/g1.hpp>int main(void)
{
    const uint8_t v = {0x8E, 0xCB, 0xFA, 0x5F, 0xD5, 0x52, 0xE1, 0x27, 0x7E, 0xDB, 0xED, 0xC8, 0xC4, 0x37, 0x7B, 0xB0, 0x41, 0x49, 0xBA, 0x85, 0x04, 0x7A, 0x30, 0x44, 0x2D, 0x40, 0x3E, 0x37, 0xCB, 0x16, 0xCB, 0x1F};

    uint256_t r;
    memcpy(r.data, v, 32);
    ::barretenberg::fr fr(r);

    const ::barretenberg::g1::affine_element res = ::barretenberg::g1::one * fr;
    return 0;
}
```