TRAIL OF BITS

RoundMe: rounding analysis made simpler

Who am I?

- **Josselin Feist (@montyly)**

ToB Twitter list

- **Trail of Bits: trailofbits.com**
  - We help developers build safer software
  - R&D focused: we use the latest program analysis techniques
  - Slither, Echidna, Tealer, Caracal, solc-select, ..

Agenda

- **Rounding risks**
- **How to find and fix**
- **RoundMe: automated analysis**

Why a talk about rounding?

- **Rounding is frequently ignored or an afterthought**
  - Can be difficult to reason with
  - But can lead to theft of funds
- **Lack of recommendations and tooling**

101 on precision loss

- **Finite bit representation of number**
- **Division truncates**

$$\frac{a * b}{c} \quad <> \quad a \quad * \frac{b}{c}$$

101 on precision loss

- **Finite bit representation of number**
- **Division truncates**

$$\frac{a * b}{c} \; != \; a \; * \frac{b}{c}$$

```
uint a = 8;
uint b = 12;
uint c = 5;
uint v0 = (a*b)/c; // 19
uint v1 = a*(b/c); // 16
```

101 Fixed point arithmetic

- **Decimals is fixed**
  - 123.456789 with a decimals of 6 -> “123456789”
  - Floating repr. equivalent: “46383879161390067 31” (IEEE 754 - 64 bits)
- **“Simple” implementations**
  - DSmath, Prb-math, solmate, ..

$$mul(a,\ b) = \frac{a * b}{decimals}$$

$$div(a,\ b) = \frac{a * decimals}{b}$$

101 Fixed point arithmetic

- **Multiplication loss precision, can round up or down**

```
function mulWadDown(uint256 x, uint256 y) internal pure returns (uint256) {
    return mulDivDown(x, y, WAD); // Equivalent to (x * y) / WAD rounded down.
}

function mulWadUp(uint256 x, uint256 y) internal pure returns (uint256) {
    return mulDivUp(x, y, WAD); // Equivalent to (x * y) / WAD rounded up.
}
```

Solmate's FixedPointMathLib.sol

- **Same for pow, sqrt, …**

Does it matter?

- **Examples:**
  - JetProtocol
  - Solana Program Library (SPL)
  - Uniswap (audit)
  - Yield V2
  - PRBMath
  - Balancer

Case study

Case study: swap out

- ***~ the number of token I receive (out) depends on how much I increase the second token's supply (in)***
    - *Token out: what you receive*
    - *Token in: what you pay*

$$token_{out} = balance_{out} * \left(1 - \frac{balance_{in}}{balance_{in} + token_{in}}\right)$$

Case study: swap out

- **Ratio is less than 1**
- **~ “ratio based on how much the token supply increase”**
  - More you sent, the lower the result
- **What if it rounds toward zero?**

$$token_{out} = balance_{out} * \left(1 - \frac{balance_{in}}{balance_{in} + token_{in}}\right)$$

Case study: swap out

$$token_{out} = balance_{out} * \left(1 - \frac{balance_{in}}{balance_{in} + token_{in}}\right)$$

$$token_{out} = balance_{out} * (1 - 0)$$

$$token_{out} = balance_{out}$$

Case study: swap out

- **Token out = balance out**
- **You receive all the tokens**
  - (but you pay a lot)

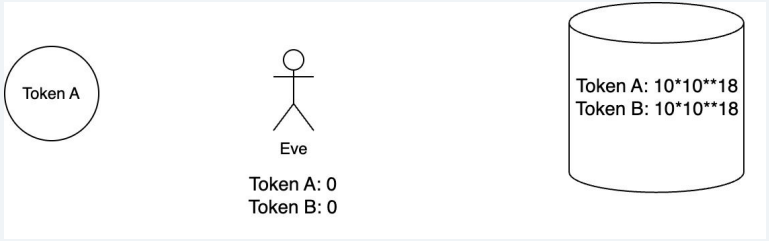
Case study: swap out

- **And so?**
  - “It’s just a trade”
  - “It will never happen”

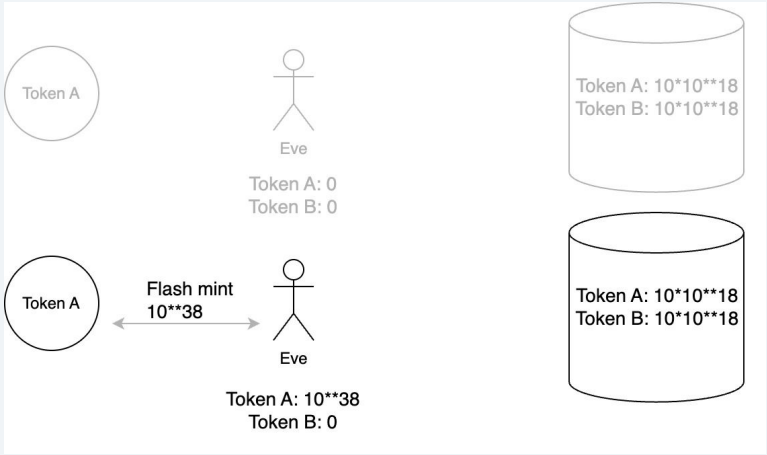
Case study: swap out

- **And so?**
  - “It’s just a trade”
  - “It will never happen”
- **Attack**
  - Force the pool to be unbalanced
  - Receiving all the balance out can be profitable

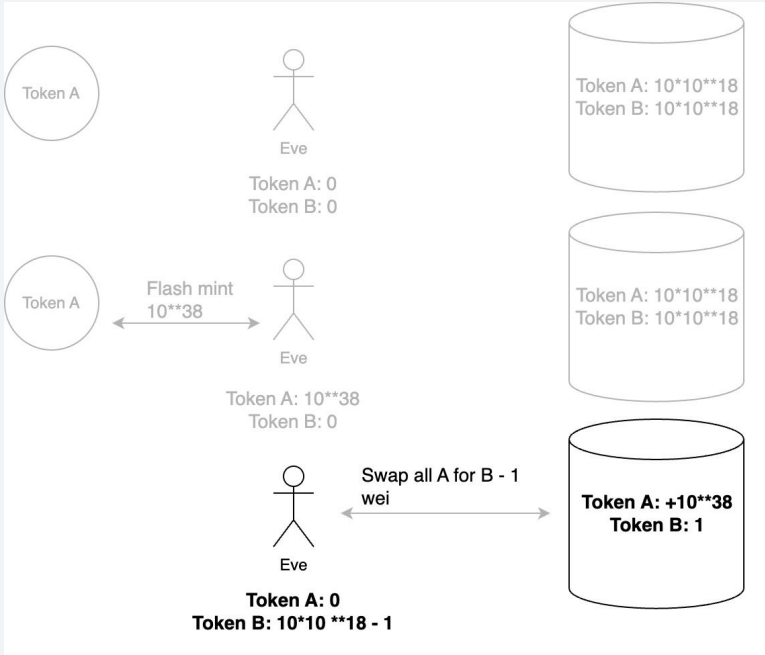
Case study: attack



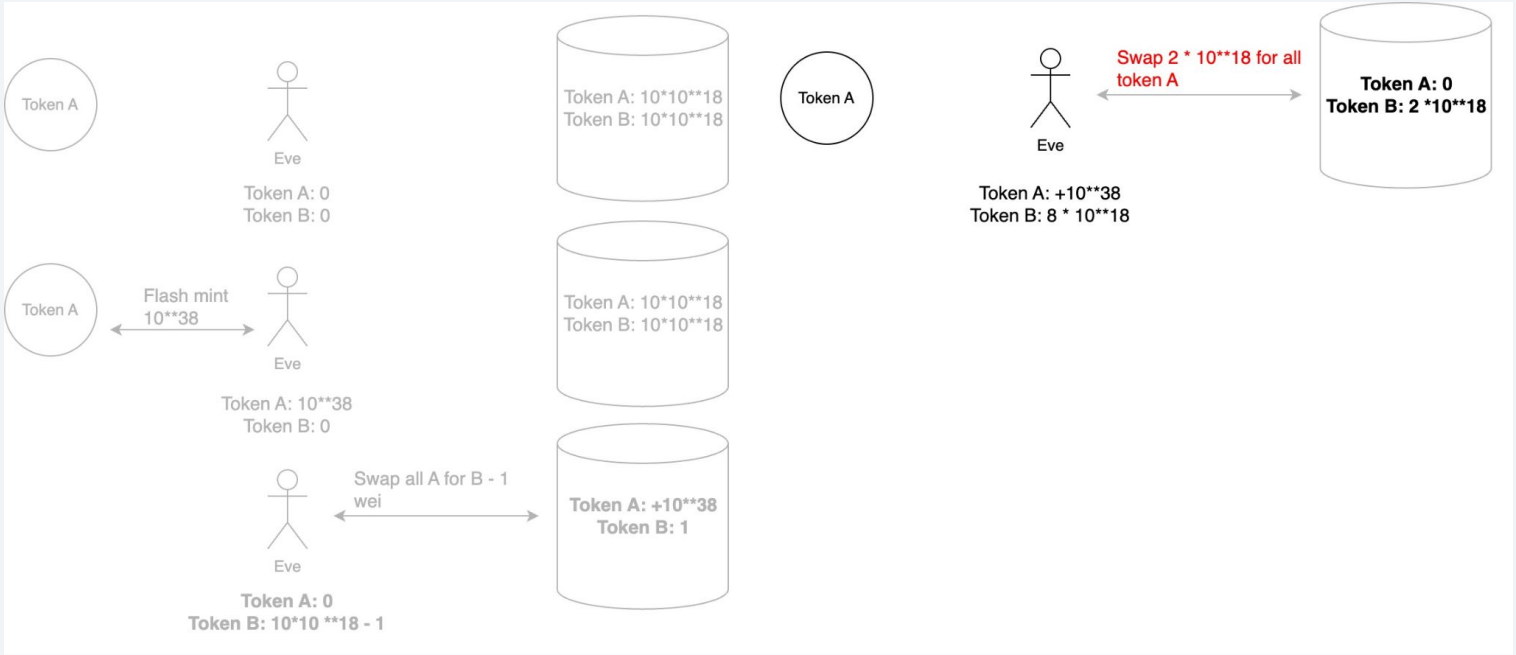
Case study: attack



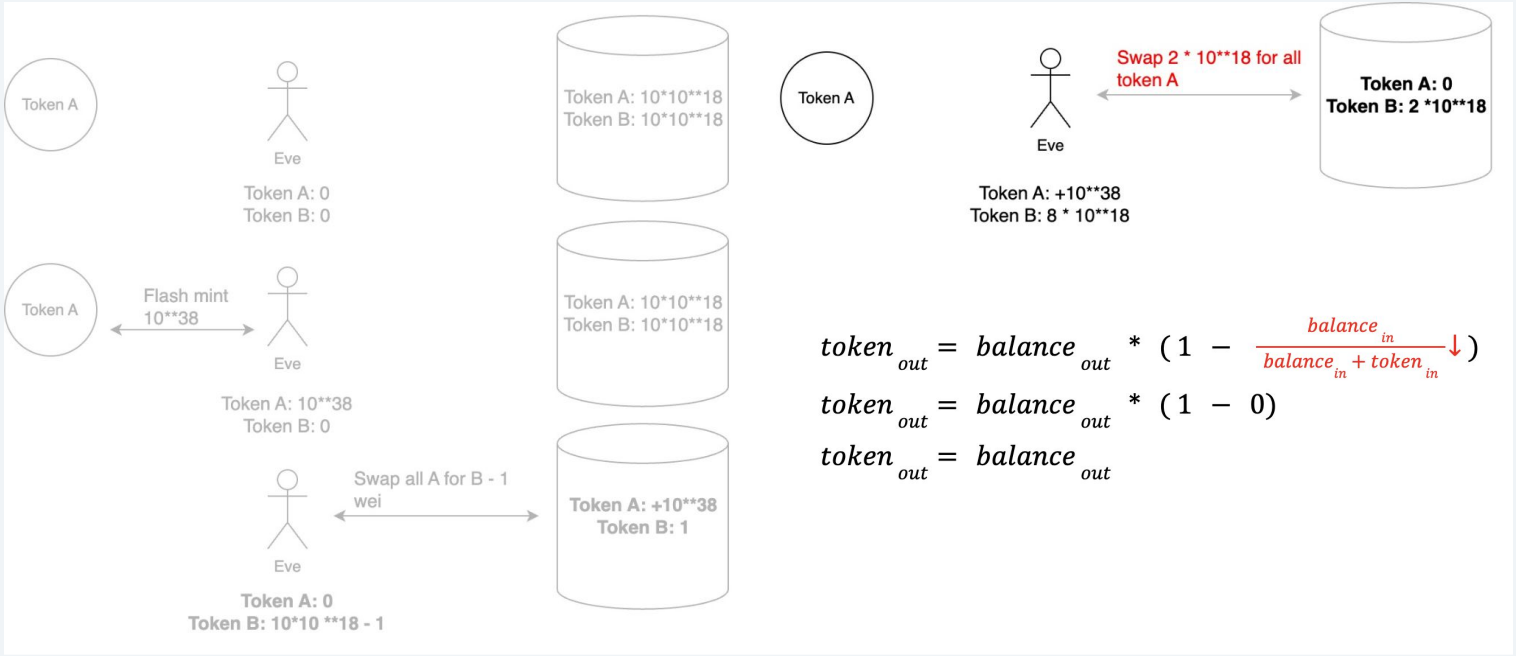
Case study: attack



Case study: attack



Case study: attack

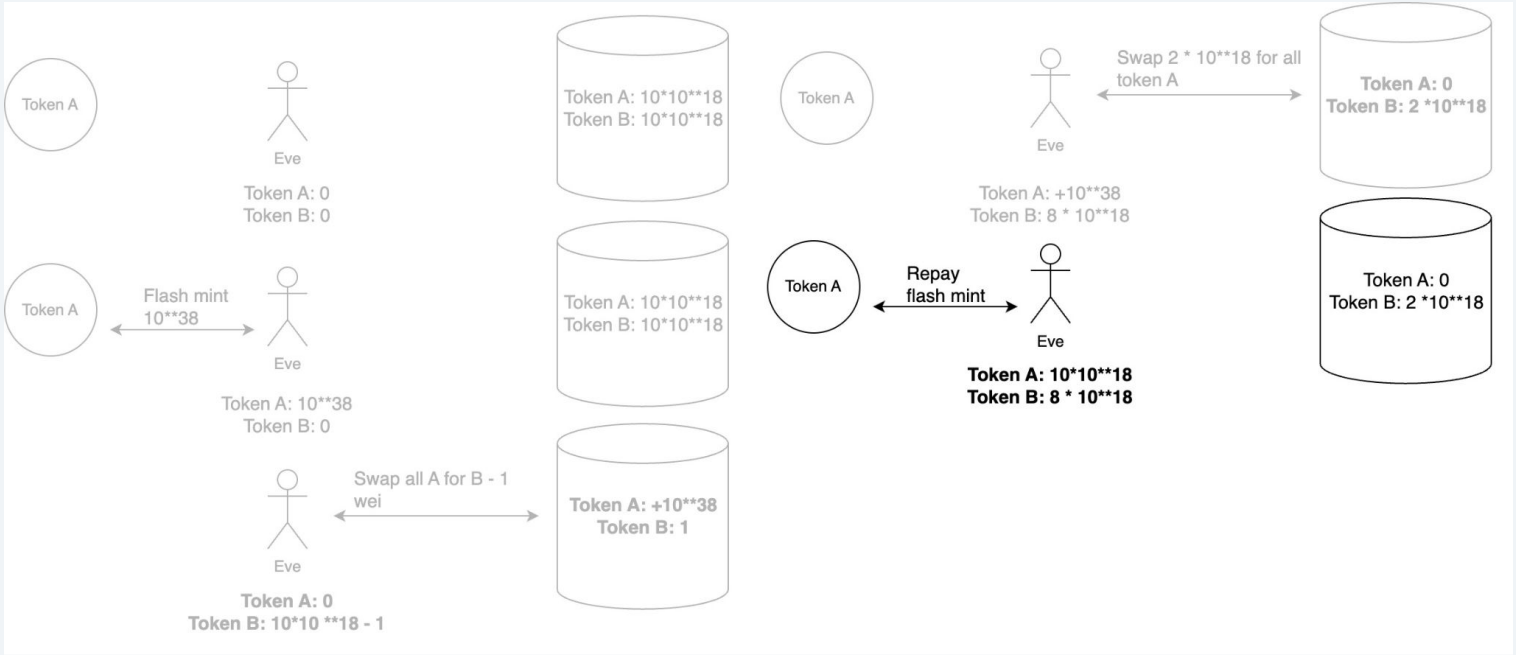


$$token_{out} = balance_{out} * (1 - \frac{balance_{in}}{balance_{in} + token_{in}}\downarrow)$$

$$token_{out} = balance_{out} * (1 - 0)$$

$$token_{out} = balance_{out}$$

Case study: attack



Case study: attack

- **Round down (↓)**
  - Swap 2* 10**18 B for ~10**38
- **If round up (↑)**
  - Swap 2* 10**18 B for ~10**38 **- 100*10**18**

$$token_{out} = balance_{out} * (1 - \frac{balance_{in}}{balance_{in} + token_{in}}\downarrow)$$

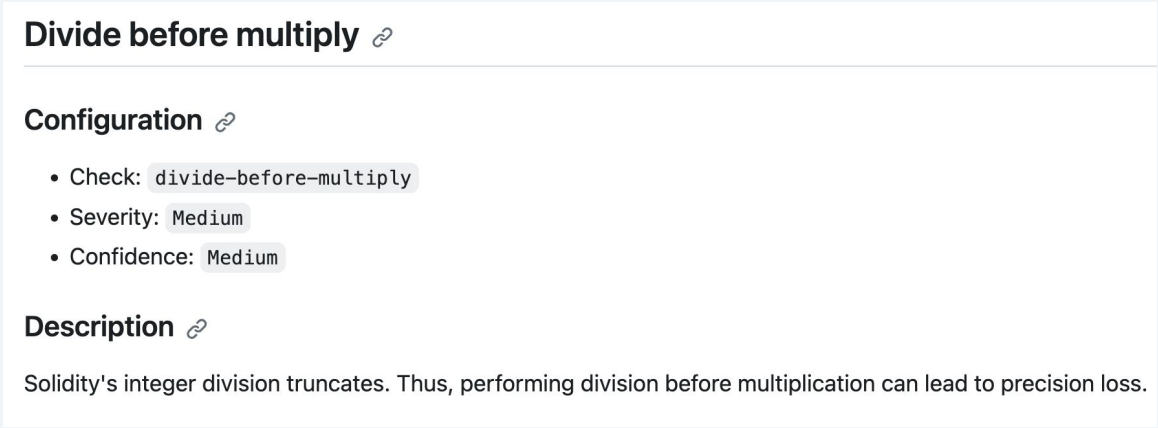
$$token_{out} = balance_{out} * (1 - 0)$$

$$token_{out} = balance_{out}$$

Recommendations

Operation order

- **Multiply first, divide after**
  - (a * b) / c instead of a * (b / c)
- **Use slither's** divide-before-multiply **detector**



Divide before multiply

Configuration

- Check: `divide-before-multiply`
- Severity: `Medium`
- Confidence: `Medium`

Description

Solidity's integer division truncates. Thus, performing division before multiplication can lead to precision loss.

Finding rounding direction

- **Formulas can become really complex**

$$token_{out} = \left(a^{\left(\frac{c}{d}\right)} * \left(1 - \left(\frac{e}{e+f+g}^{\left(\frac{h*k}{j}\right)}\right)\right)\right)$$

Finding rounding direction

- **Start from the outer result**
  - Token out => round down (↓)

$$token_{out} = \left(a^{\left(\frac{c}{d}\right)} * \left(1 - \left(\frac{e}{e+f+g}^{\left(\frac{h*k}{j}\right)}\right)\right)\right)$$

Finding rounding direction

- **Start from the outer result**
  - Token out => round down (↓)

$$token_{out} = \left(a^{\left(\frac{c}{d}\right)} * \left(1 - \left(\frac{e}{e+f+g}^{\left(\frac{h*k}{j}\right)}\right)\right)\right)$$

$$token_{out} = \left(a^{\left(\frac{c}{d}\right)} * \left(1 - \left(\frac{e}{e+f+g}^{\left(\frac{h*k}{j}\right)}\right)\right)\right)$$

Finding rounding direction

- $a^{(\frac{c}{d})}$ **needs to round down (↓)**
  - If a >= 1
    - to ↓ , c/d needs to ↓
  - If a < 1
    - to↓, c/d needs to ↑
- **The rounding direction depends on the value's context**

$$token_{out} = \left(a^{(\frac{c}{d})} * \left(1 - \left(\frac{e}{e+f+g} \right)^{(\frac{h*k}{j})} \right) \right)$$

Finding rounding direction

- **Right side needs to round down (↓)**
  - 1 - X needs to round down (↓)
  - X needs to round up (↑)
  - etc..

$$token_{out} = \left(a^{\left(\frac{c}{d}\right)} * \left(1 - \left(\frac{e}{e+f+g}^{\left(\frac{h*k}{j}\right)}\right)\right)\right)$$

General recommendations

- **Analyze the arithmetics step by step**
  - Always round to benefit the protocol
  - Start from the outer component toward the inner components
- **Use tools**
  - Fuzzing
  - RoundMe (see next)

General recommendations - Developers

- **Create primitives to round up / down**
  - mul_up, mul_down, ...
  - Make every rounding explicit
- **Rewrite the formula**
  - Multiply before divide
  - Reduce number of operation
  - Avoid expression that can be positive and negative
- **Document and test known precision loss**

Introducing roundme

RoundMe

- **Human-assisted rounding analyzer**
  - User provides the formula
  - Roundme automatizes the step by step process
- **https://github.com/crytic/roundme**
  - Rust
- **Early stage**
  - 6 rounding rules - more to come
  - Only unsigned fixed point integers

RoundMe

```
$ roundme config
Formula to analyze:
(a**(c/d) * (1 - ((e/(e + f + g)) ** ((h * k)/j))))
Should the formula round up? Y/N (yes, no)
n
Is a greater than 1? Y/N (yes, no)
y
Is (e / ((e + f) + g)) greater than 1? Y/N (yes, no)
n
config.yaml generated.
```

RoundMe

```
 ---
formula: (a**(c/d) * (1 - ((e/(e + f + g)) ** ((h * k)/j))))
round_up: false
less_than_one:
  - (e / ((e + f) + g))
greater_than_one:
  - a
```

RoundMe

- **Generate reports in latex/PDF**

$$((a^{(\frac{c}{d}\downarrow)}) *_{\downarrow} (1 - ((\frac{e}{((e+f)+g)}\uparrow)^{(\frac{(h *_{\downarrow} k)}{j}\downarrow)})))$$

Conclusion

Conclusion

- **Pay attention to the roundings**
- **Make all roundings explicit**
  - https://github.com/crytic/roundme will help
- **Use a fuzzer**

- **Interested in fuzzing?**
  - “How to fuzz like a pro” at 6pm (sponsor zone)
  - Invariant as a service