The Definitive Guide to Rounding in DeFi

WhoAmI

- Josselin Feist (@Montyly)
- Independent security researcher
  - Trail of Bits: 2017 - 2025
  - Reviewed DeFi/L1/L2
  - Created Slither
- *“I am a hacker and a scholar”*

Why a talk about rounding?

- Difficult to reason with, yet critical:
  - JetProtocol
  - Solana Program Library (SPL)
  - Yield V2
  - PRBMath
  - Bunny
  - Balancer (twice)
- Lack of systematic approach
  - “Just round in favor of the protocol” -> not that easy

Agenda

- Rounding 101
- Case study
- Favor the protocol?
- Recommendations

Rounding 101

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

- 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, .

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

101 fixed point arithmetic

- Multiplication loss precision, can round up or down
- Same for pow, sqrt, …

```
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

Case Study

Case study: Balancer (2021)

- 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
- You receive all the tokens
  - But you pay a lot

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

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

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

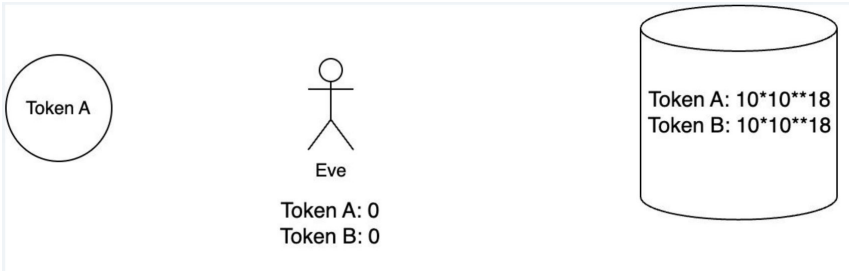
Case study: swap out

- And so?
  - “It’s just a trade”
  - “It won’t happen”

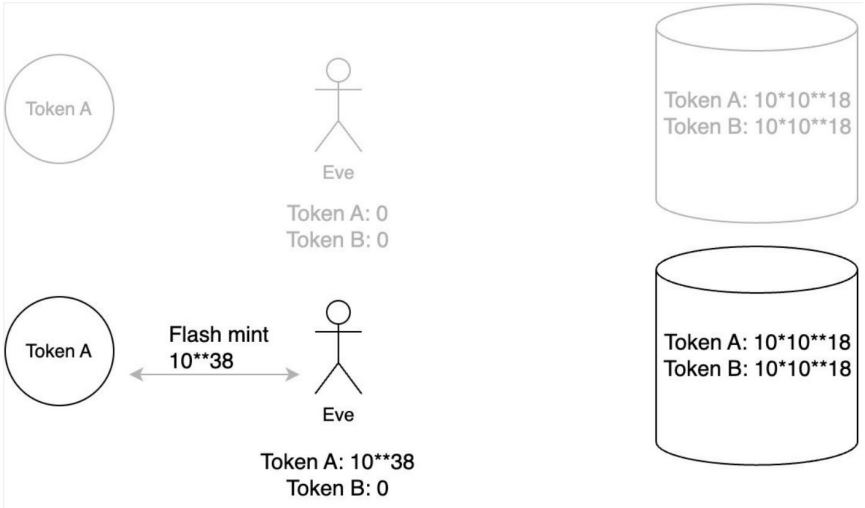
Case study: swap out

- And so?
  - “It’s just a trade”
  - “It won’t happen”
- Attack
  - Force the pool to be unbalanced
  - Make a profit from receiving all the balance out

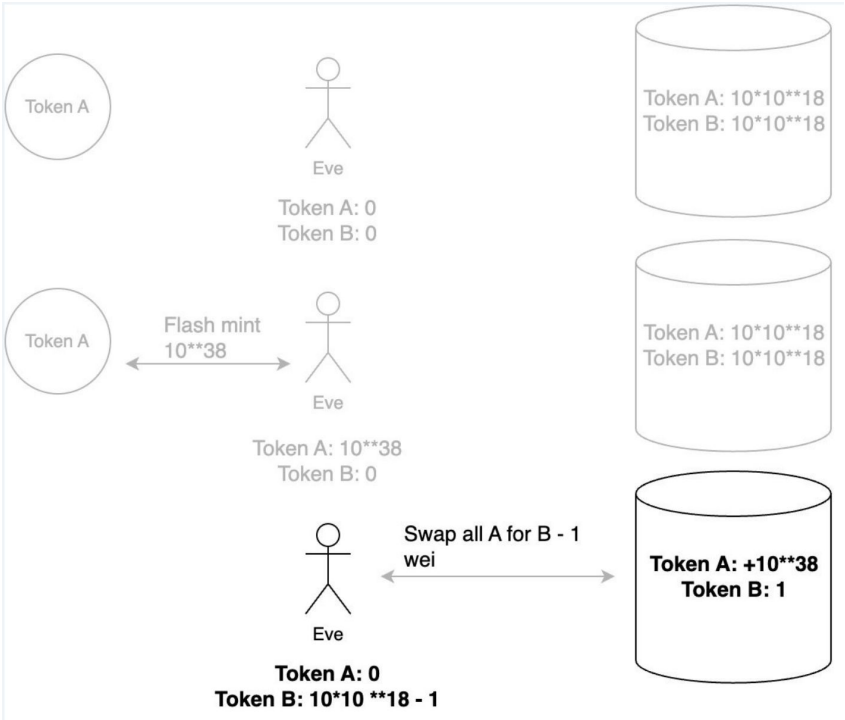
Case study: attack



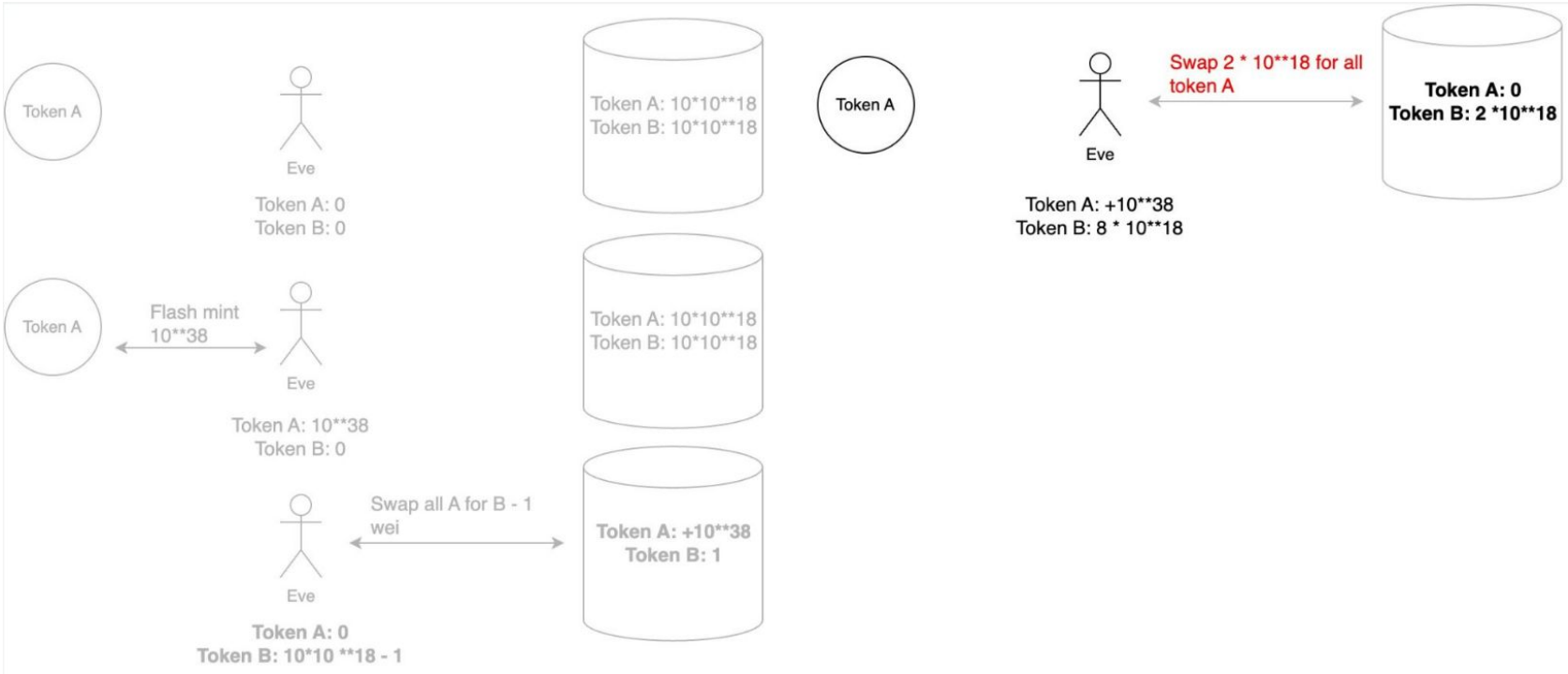
Case study: attack



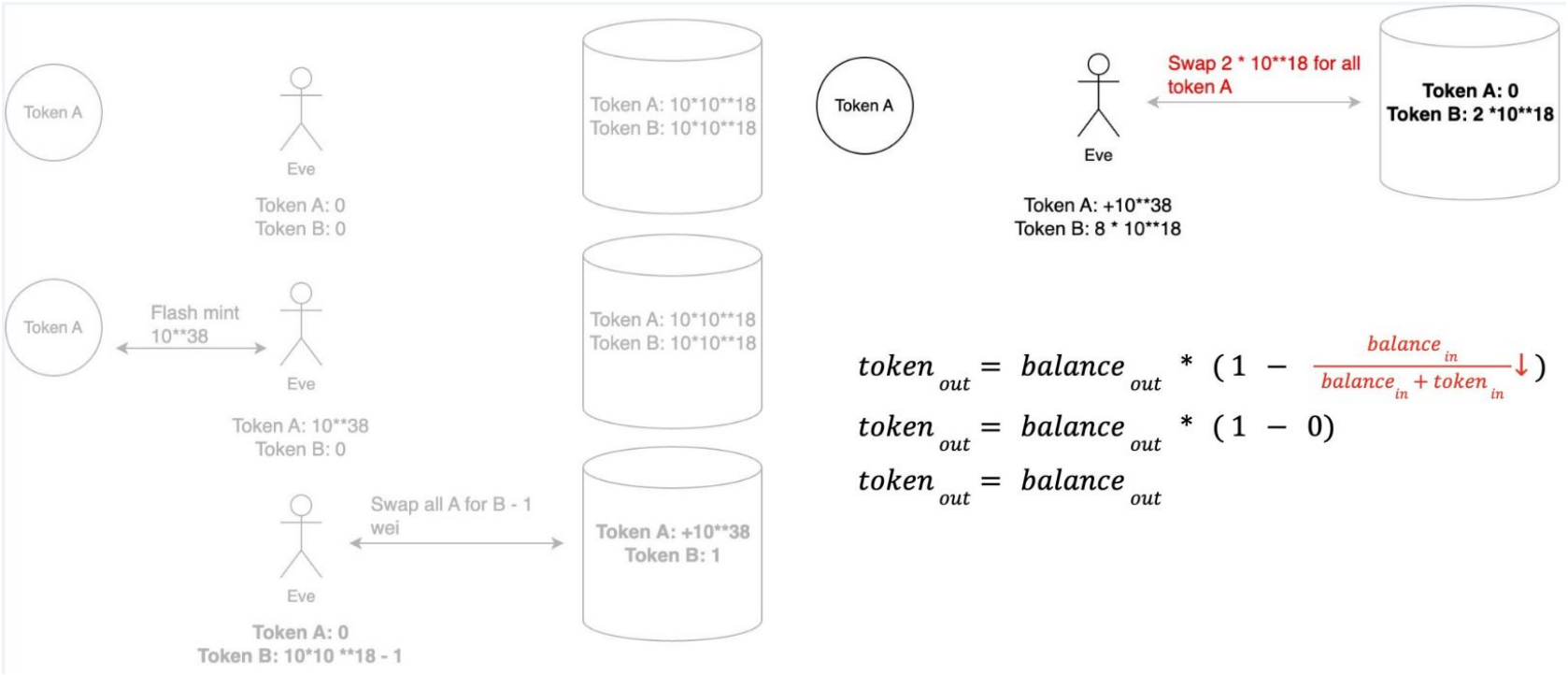
Case study: attack



Case study: attack



Case study: attack

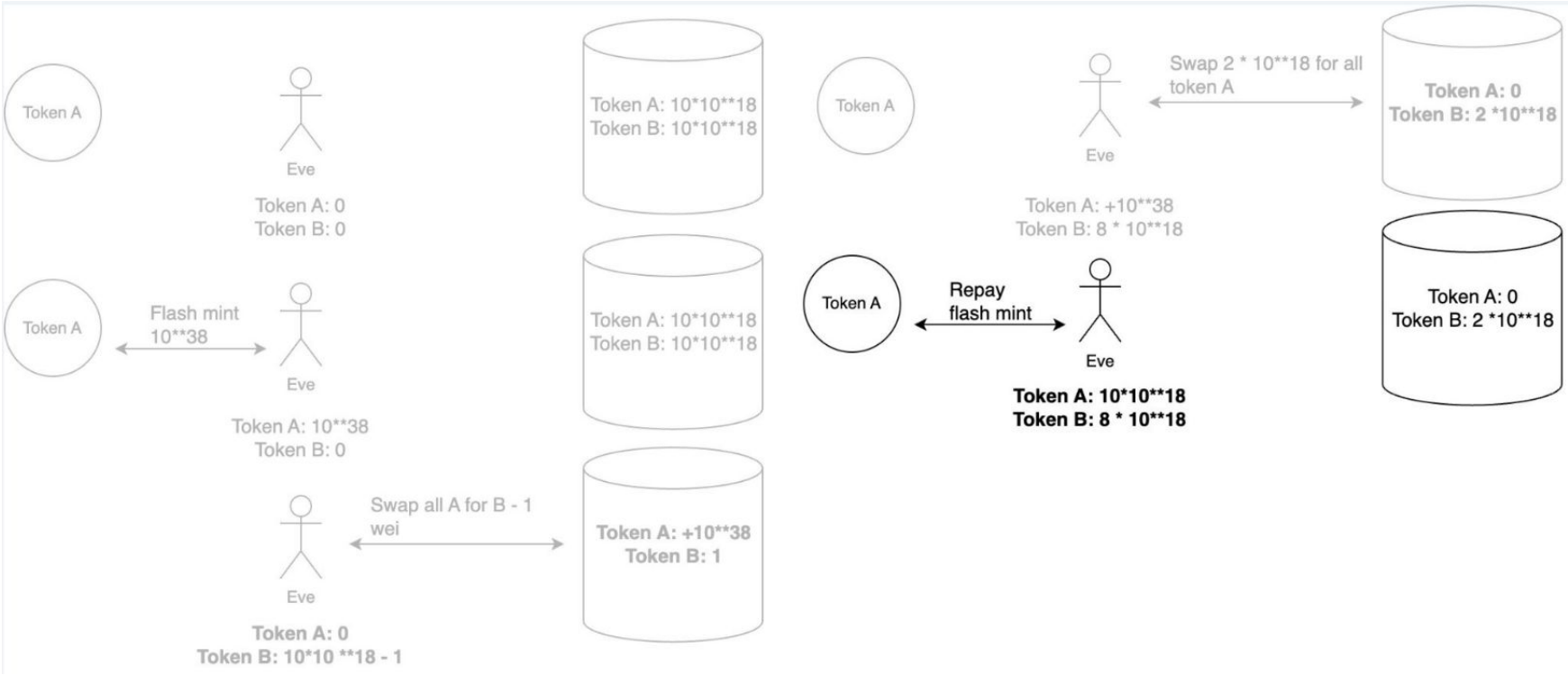


$$token_{out} = balance_{out} * (1 - \frac{balance_{in}}{balance_{in} + token_{in}}\downarrow)$$

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

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

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**

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

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

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

Favor the protocol

“Round to favor the protocol”

- Formula can become complex

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

“Round to favor the protocol”

- Formula can become complex
  - Start from the outer result
    - Token out => round down (↓)

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

“Round to favor the protocol”

- Formula can become complex
  - Start from the outer result
    - Token out => round down (↓)
  - Decompose

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

“Round to favor the protocol”

- Formula can become complex
  - Start from the outer result
    - Token out => round down (↓)
  - Decompose
  - $a^{(\frac{c}{d})}$ needs to round down

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

“Round to favor the protocol”

- $a^{(\frac{c}{d})}$ needs to round down
  - If a >= 1
    - to ↓ , c/d needs to ↓
  - If a < 1
    - to↓, c/d needs to ↑

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

“Round to favor the protocol”

- $a^{(\frac{c}{d})}$ 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**

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

“Round to favor the protocol”

- Intermediate steps can be tricky
  - B needs to round down
  - C needs to round up
  - What to do with A?

$$B = A * (....)$$

$$C = A * (....)$$

Rounding - In practice

- For security researchers
  - Hard to find
  - Hard to demonstrate impact
- For developers
  - Afterthought for most of the teams
  - Can require a lot of refactoring to fix

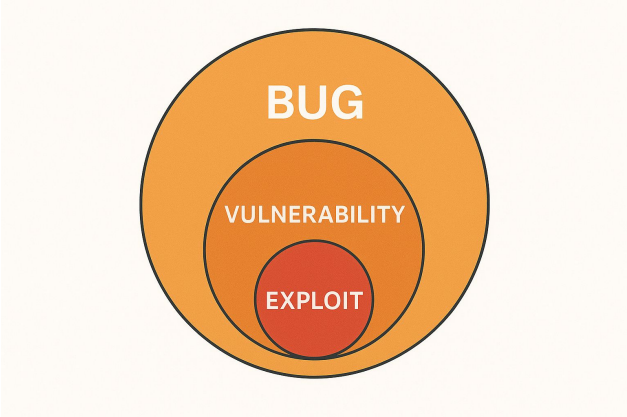
Recommendations

Rounding - What we need

1. Paradigm shift
2. Design
3. Implementation

Recommendations - Paradigm shift

- Bug -> vulnerability -> exploit
  - All incorrect roundings are bugs
  - Some roundings bugs are vulnerabilities
  - Some vulnerabilities can lead to exploit
- **Consider all every incorrect rounding as a bug**



Recommendations - Design

- **“Round to favor the protocol”**
- Think ahead how the code will be structured
  - Rounding of intermediate steps -> might contradict
- Look for math property
- Consider “trapping” paths
  - Ex: rounding in favor of the protocol that blocks liquidation
- Customized vuln matrix
  - Is rounding down on fees acceptable?

Recommendations - Implementation

- Use explicit name
  - mul_down(...)
  - Or mul(..., rounding)
- Document & test every decision
- Use fuzzing or formal verification

Conclusion

- Rounding is hard
- Tldr:
  - Every incorrect rounding is a bug
  - Always round to favor the protocol