



FINDING BUGS: 42 TIPS FROM 4 SECURITY RESEARCHERS

WHO?



Joran



Nat

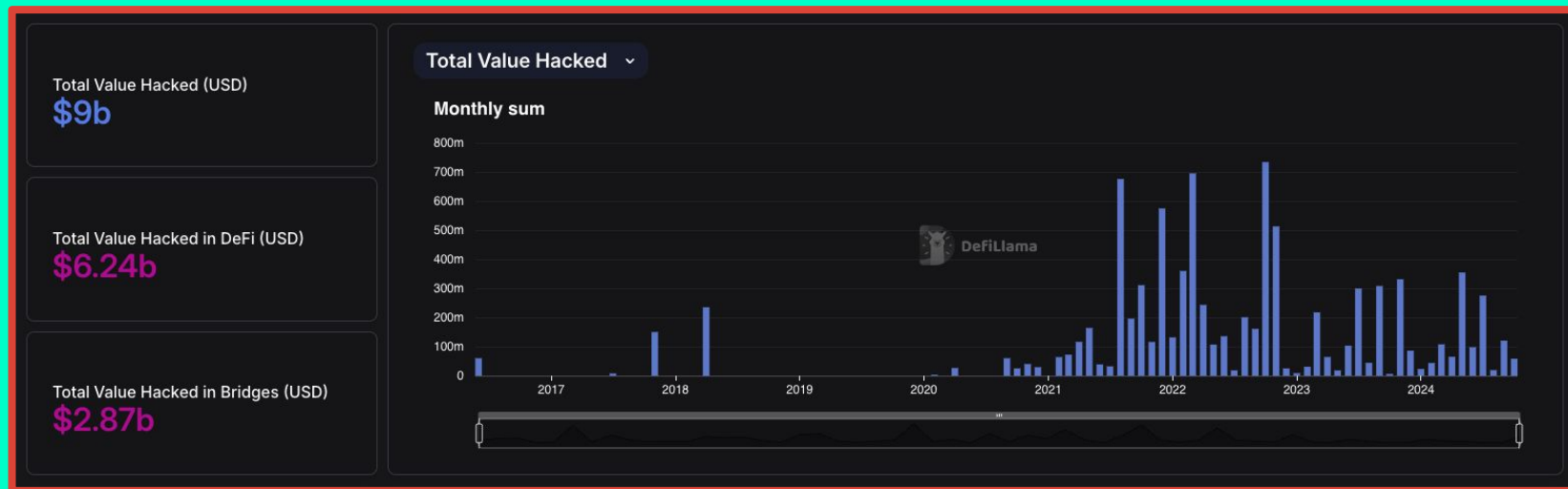


tincho



0xRajeev

WHY?



TIP #1 - PACING

the grindset is a lie

TIP #2 - SCOPING OUT TARGETS

- picking the right target is important
- type
- amounts (espec. medium)
- age of program
- size

TIP #3 – TRY TO SLEEP (SORT OF)



TIP #4 - BB DISCUSSIONS -> PREPARE FOR FALLACIES

- every bug bounty submission is different
 - everyone responds differently to having a bug submitted
 - sometimes there is disagreement
-
- not intentional
 - preparation -> productive conversation

TIP #5 - BB DISCUSSIONS -> SEE DEVS AS A BUSINESS PARTNER

- style of communication
- productive

TIP #6 - DIVERSIFY YOUR SKILLS

- competitive advantage

- Become a quant nerd
- learn about economics
- learn cryptography
- learn how provers work
- learn how compilers work

SOME STRATEGIES

TIP #7 - FIND A PARADIGM AND SPECIALIZE

- defi
 - leverage
 - concentrated liquidity
 - ...
- zk
- go broad

TIP #8 - GO DEEP

- optimal for time constrained bb hunters
- explore
- identify
- exploit

TIP #9 - READ THE DOCUMENTATION

- design mistakes can be critical
- quick
- setup for success

TIP #10 - THE TOOLBOX MENTAL MODEL

- killchain
- tricks / gadgets

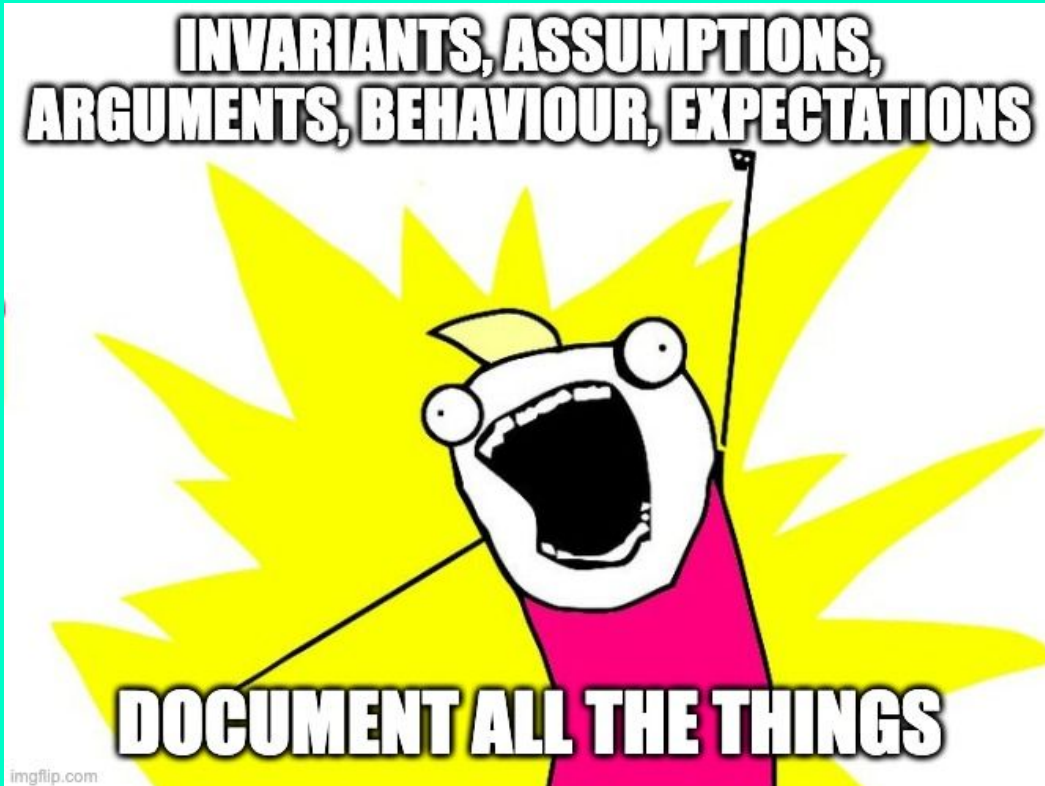
FOR DEVS

TIP #11: ARCHITECT FIRST, CODE LATER

Me trying to fix bug in my code
avoiding the other bugs to happen.



TIP #12: DOCUMENT EVERYTHING



TIP #13: SIMULTANEOUSLY TEST AND CODE



TIP #14: AUTOMATE AS MUCH AS YOU CAN; FOCUS ON REAL THINGS

-



TIP #15: PROACTIVELY CARE

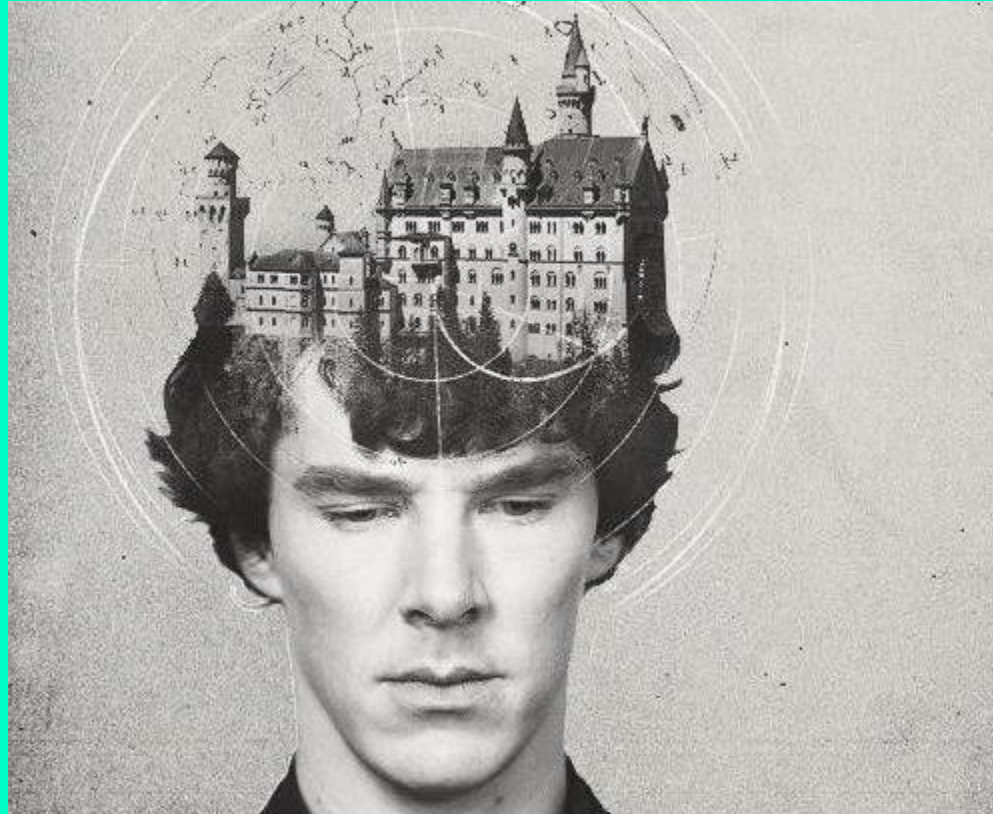
- Prevents bugs in the future
- Smaller code = bugs with less impact

FOR SECURITY RESEARCHERS

TIP #16: PRE-EMPTIVELY ASK



TIP #17: BUILD A MENTAL MODEL OF THE CODEBASE

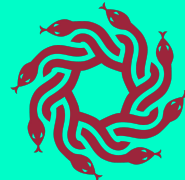


TIP #18: LEARN YOUR TOOLS



Hardhat

kontrol



Medusa

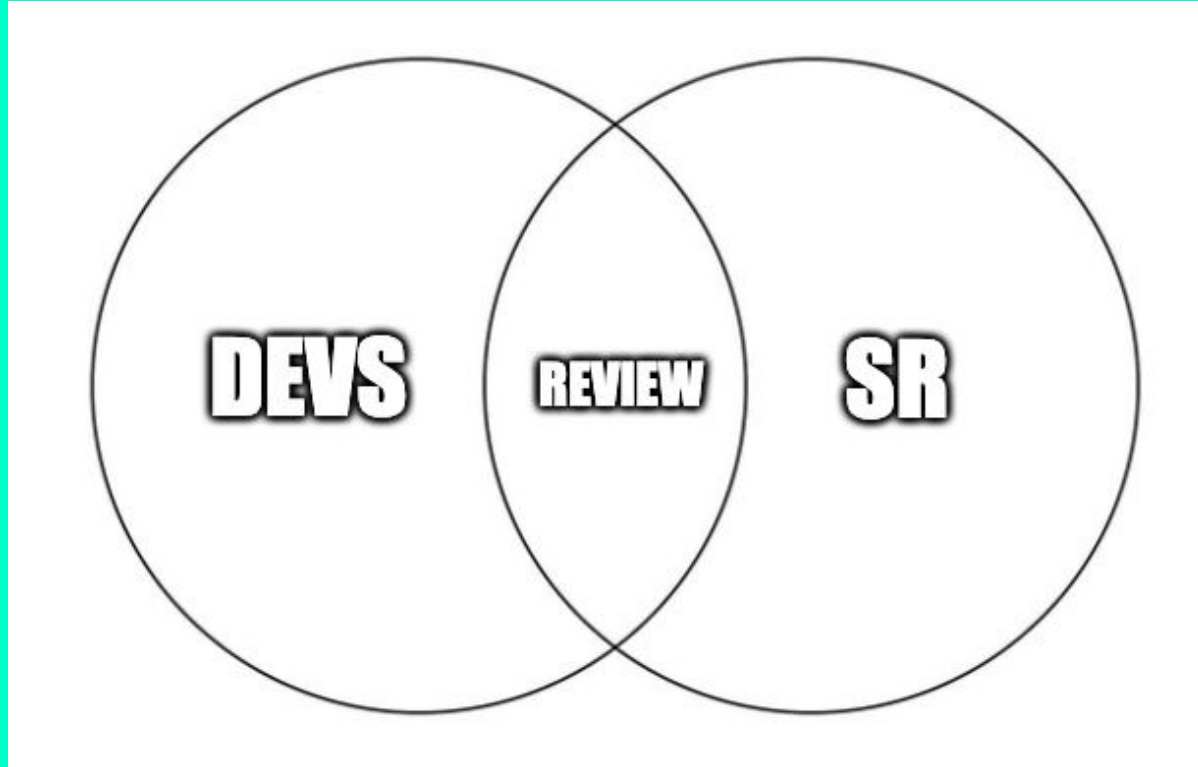
TIP #19: LEARN HOW USERS USE THE SYSTEM

How developers want users to use their system

!=

How users are intended to use the system

TIP #20: LEAN ON EACH OTHERS STRENGTHS



TIP #21 - LEARN FROM THE PAST

The screenshot displays the Solodit website interface. The top navigation bar includes links for Findings, Checklist, Audits, Bug Bounties, and Leaderboard. The left sidebar contains filters for findings, including a search bar, report tags (ALL), source (ALL), impact (HIGH, MEDIUM), user (searchable), protocol name (searchable), protocol category (ALL), and number of finders (At Least 1). The main content area shows a list of findings, with the first two being 'Transaction DOS via `permit()` front-running' and 'Potential loss of funds if `msg.value` exceeds `amountIn + executionPrice`', both by MixBytes. The third finding is '[H-02] `newLeverage` wrongly calculated inside `requestIncreasePositionSize`' by Pashov Audit Group. The fourth is 'Successful transactions are not stored, causing a replay attack on `redeemDepositsAndInternalBalances`' by Beanstalk: The Finale. The fifth is 'Ignoring the Well Function logic for a ratio of reserves calculation' by Beanstalk: Dive Into Basin. The right panel provides a detailed view of the first finding, 'Transaction DOS via `permit()` front-running', including its date (10/10/2024), author (EYWA), severity (MED), rarity (4 stars), quality (5 stars), categories (Dexes), and tags (Grief Attack, Front-Running, DOS). The description explains that the `permit()` data is publicly accessible and can be replicated to execute the permit, potentially reverting the transaction. A reference link is provided: <https://github.com/eywa-protocol/eywa-clp/blob/d68ba027ff19e927d64de123b2b02f15a43f8214/contracts/RouterV2.sol#L100-L105>. The recommendation section is also visible.

Solodit

Findings Checklist Audits Bug Bounties Leaderboard

14111 findings found

Search for keywords (title, content)

Search for keywords

Report Tag

ALL

Source

ALL

Impact

HIGH, MEDIUM

User

Search for user

Protocol Name

Search for protocol

Protocol Category

ALL

Number of finders

At Least 1

Bookmarked Read All Unread

Transaction DOS via `permit()` front-running

10/10/2024 - EYWA

Author(s): MixBytes

Potential loss of funds if `msg.value` exceeds `amountIn + executionPrice`

10/10/2024 - EYWA

Author(s): MixBytes

[H-02] `newLeverage` wrongly calculated inside `requestIncreasePositionSize`

22/07/2024 - Gainsnetwork May

Author(s): Pashov Audit Group

Successful transactions are not stored, causing a replay attack on `redeemDepositsAndInternalBalances`

08/07/2024 - Beanstalk: The Finale

Author(s): deadrosesxyz, asefewwexa, T1MOH, PutraLaksmana

Ignoring the Well Function logic for a ratio of reserves calculation

29/04/2024 - Beanstalk: Dive Into Basin

Author(s): pontifex

Transaction DOS via `permit()` front-running

10/10/2024 - EYWA at

Author(s): MixBytes

Rarity ★★★★★ Quality ★★★★★

Categories: Dexes

Tags: Grief Attack Front-Running DOS

<> Details Notes

Description

The `permit()` data, once submitted, is publicly accessible in the mempool. An attacker can execute the permit by replicating the transaction arguments. Once the permit is executed, the second call with identical parameters will revert.

In a scenario where a signed transaction includes `PERMIT_CODE`, a malicious user can "activate" this permit, bypassing the router's `start()` function. As a result, the user's `start()` transaction would fail:

- <https://github.com/eywa-protocol/eywa-clp/blob/d68ba027ff19e927d64de123b2b02f15a43f8214/contracts/RouterV2.sol#L100-L105>

Reference:

- <https://www.trust-security.xyz/post/permission-denied>

Recommendation

TIP #22 - DEPTH OR WIDTH?

In **depth** you can find
special / rare issues



Pointless without
context

MUST **zoom out!**

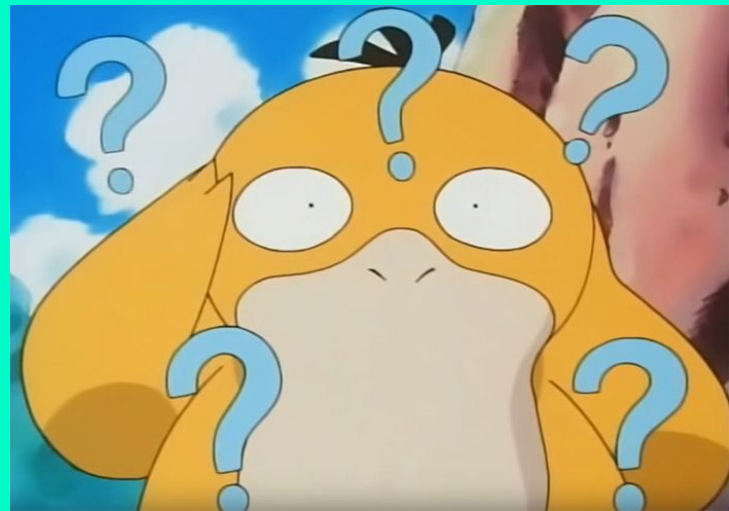
Understand expected use cases,
relationships, cross-contract dynamics

TIP #23 - BEWARE OF DIMINISHING RETURNS

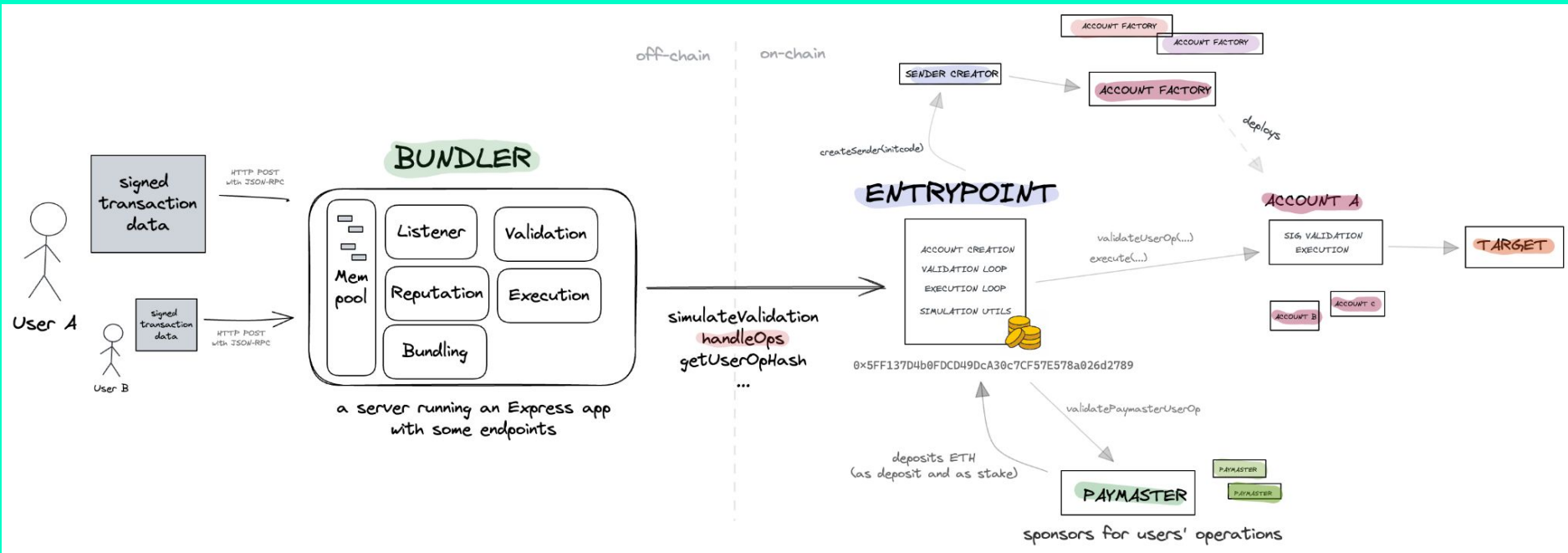
How long should I chase
down this bug?

How much detail should I include
in this report with PoC?

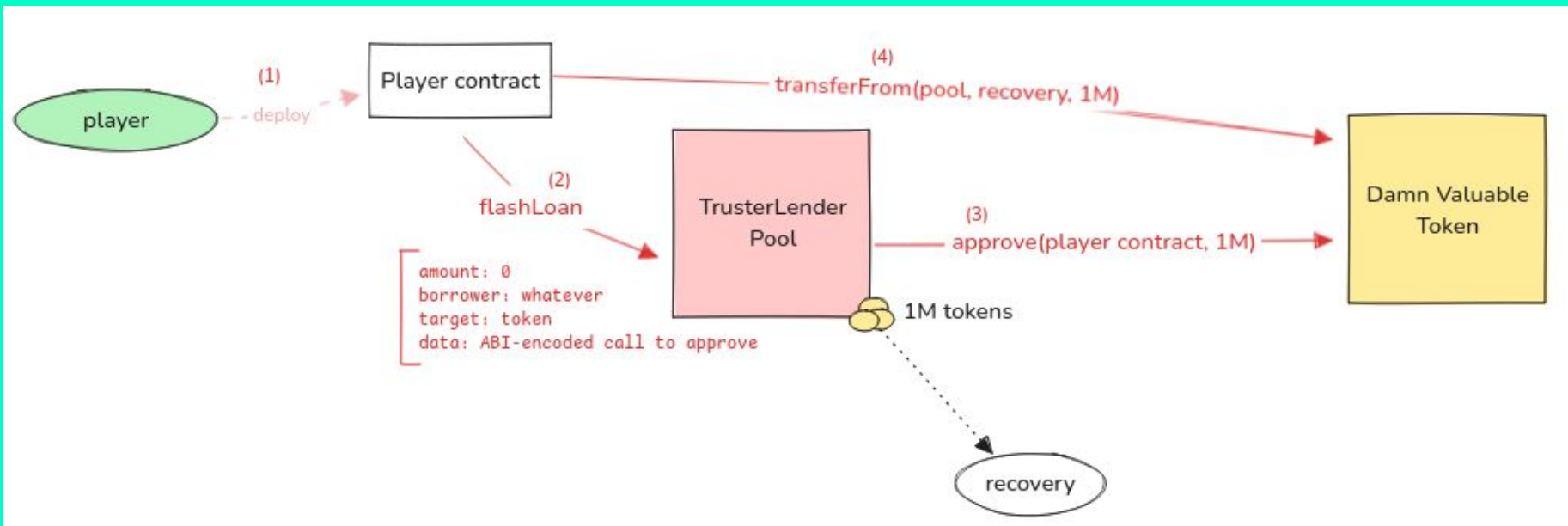
How much should I fight back in
this endless escalation thread?



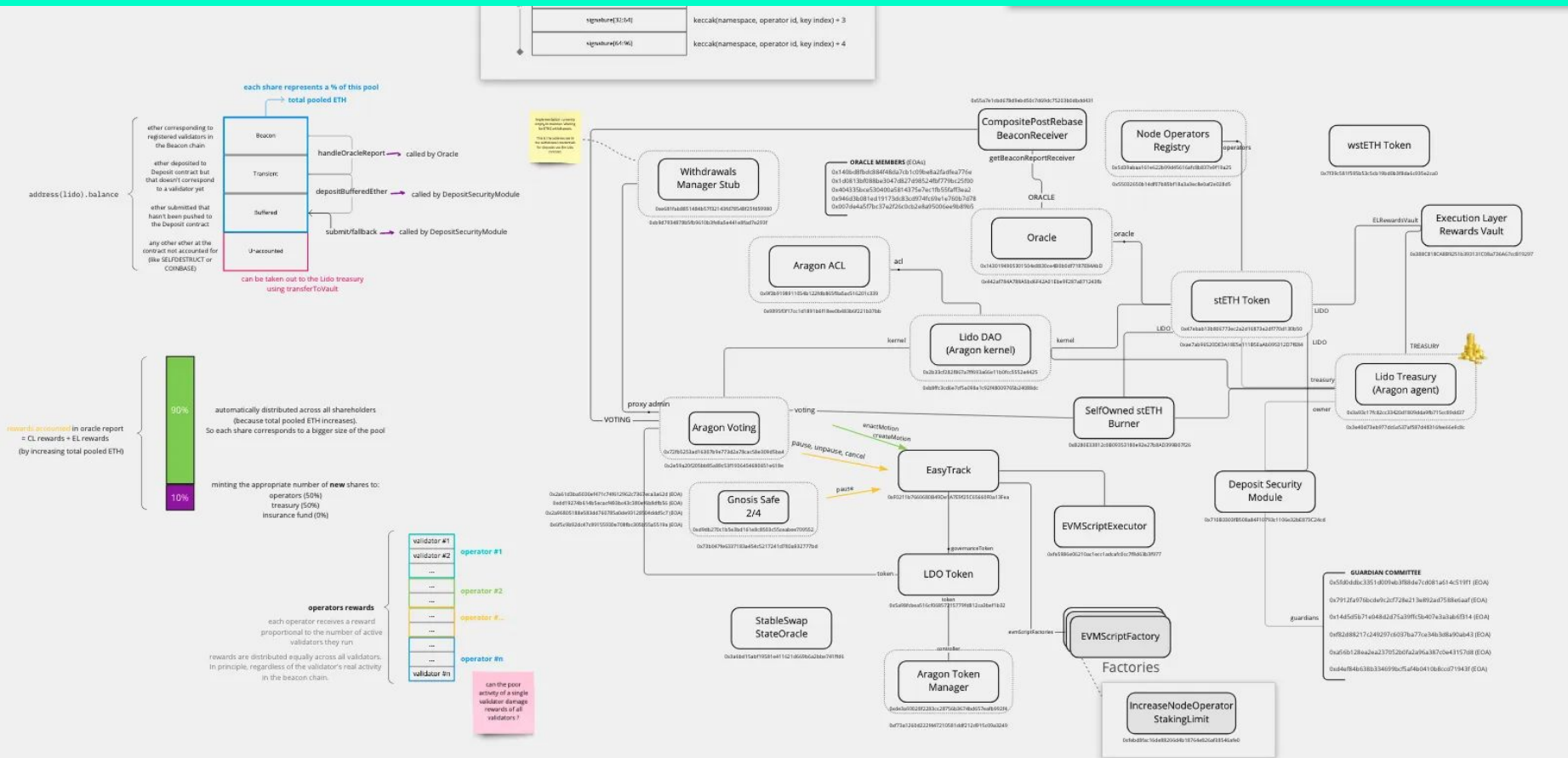
TIP #24 - DIAGRAMS FTW



TIP #24 - DIAGRAMS FTW



TIP #24 - DIAGRAMS FTW



TIP #25 - OUT OF THE COMFORT ZONE

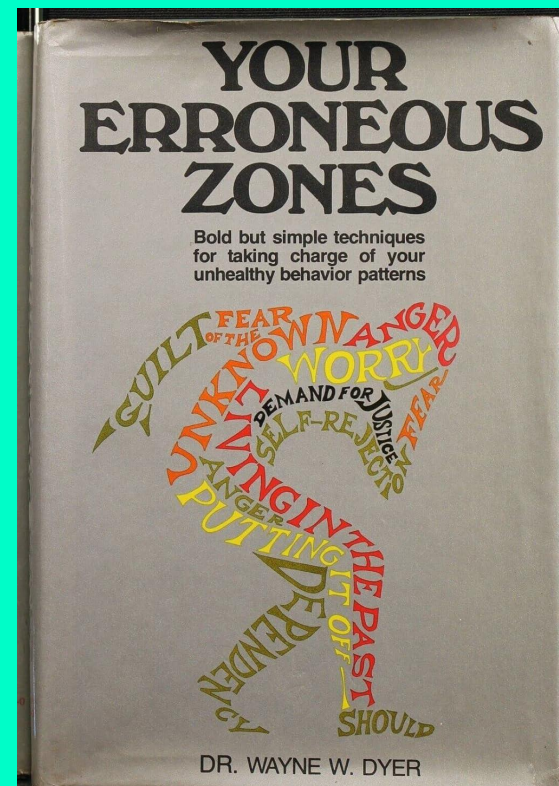
What's most people going
to review?



Can I go somewhere else?

What skills do I need?

Can I learn during the review?



TIP #26 - WRITING FTW

Writing to think ☒

Writing to not forget ☒

Writing to communicate ☒

TIP #27 - WHAT'S NOT TESTED?

Less tests, more chances for bugs.

Review, run and rework test suites to find holes.

TIP #28 - CHOOSE YOUR GROUND

- **Auditing** is not bug hunting
- **Bug hunting** is not contests
- **Contests** are not auditing

TIP #29 - WHATEVER YOU DO, BE PROFESSIONAL

→ Assertiveness

→ Thoughtfulness

→ Attention to detail

→ Respect

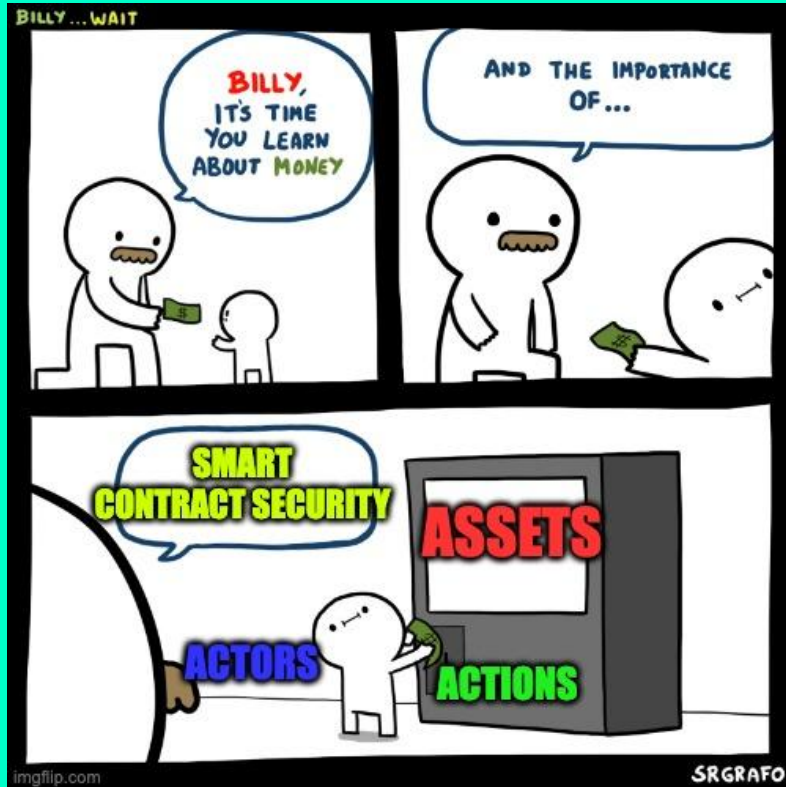
and **please** avoid
bulls**t reports
on Twitter,
thanks

TIP #30 - PATIENCE

- Finding cool bugs **takes time**
- Reporting **takes time**
- Triaging **takes time**
- Judging **takes time**
- Escalations **take time**
- Payments **take time**



TIP #31: ASSESS - ASSETS, ACTORS & ACTIONS



Assets: What? Where?



Actors: Who?



Actions: What? When?
How? Why?

TIP #32: BEHAVIOR - EXPECTED VS UNEXPECTED

EXPECTATION



REALITY



Normally vs Anomaly

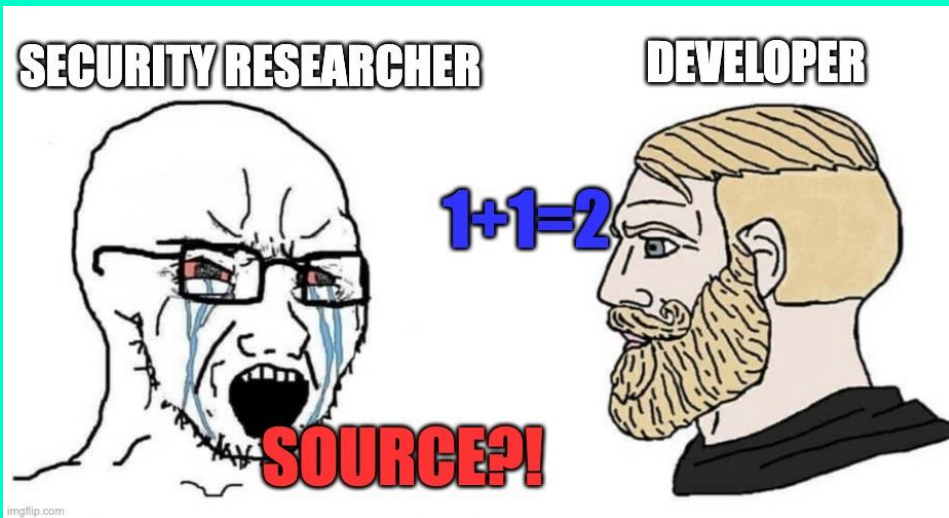


Bug \subset Anomaly



Bug \ll Vulnerability

TIP #33 CHALLENGE: ASSUMPTIONS



Root of all Bugs!

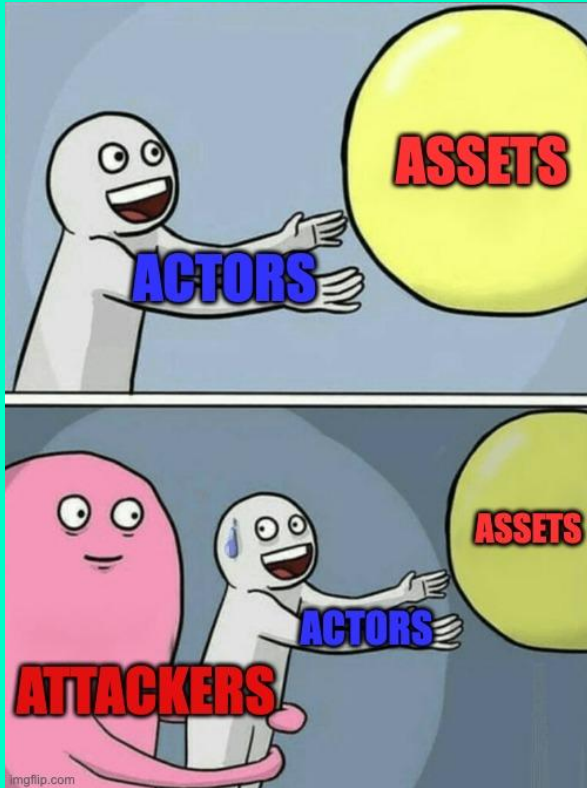


Question everything



5 W's: Who? What? Why?
When? Where?

TIP #34: DISCOVER - MISUSE SCENARIOS



Use vs Misuse



Abuse \subset Misuse



Attack surface

TIP #35: EVALUATE - SEVERITY



Impact x Likelihood

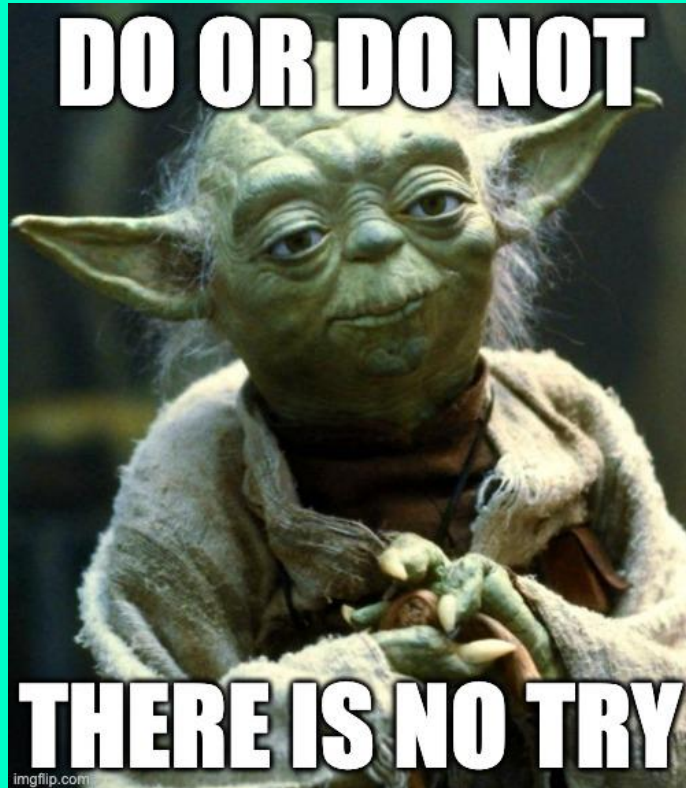


Impact: Loss/Lock, Temp/Perm,
All/Few



Likelihood: Always/Sometimes,
Unconditional/Conditional

TIP #36: FIND - INCONSISTENCIES



Inconsistent checks

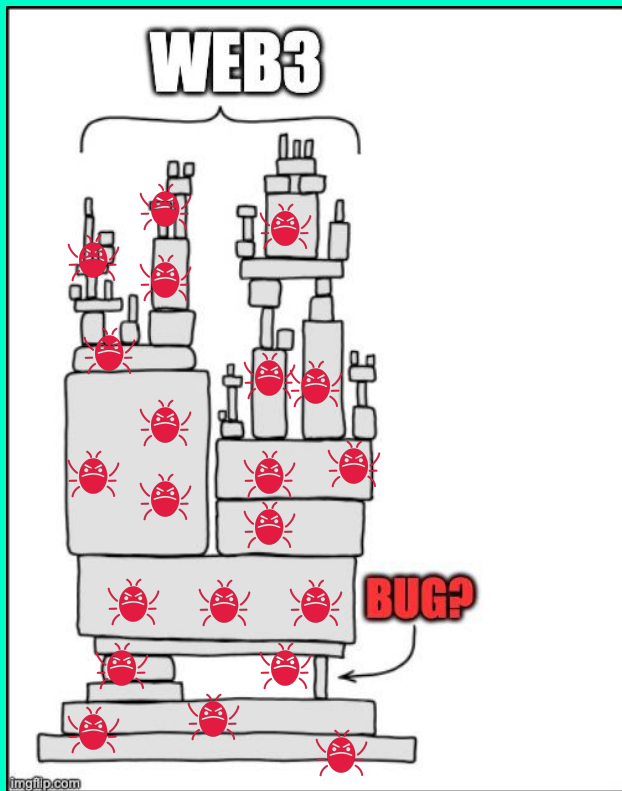


Inconsistent transitions



Inconsistent state

TIP #37: GROK EVERY LINE



Every line matters!



Read between the lines



Read beyond the lines

TIP #38: HACKER MENTALITY



Look: What's there &
What's missing



Think: Attacker



Act: Whitehat

TIP #39: INFERRED INVARIANTS



Properties

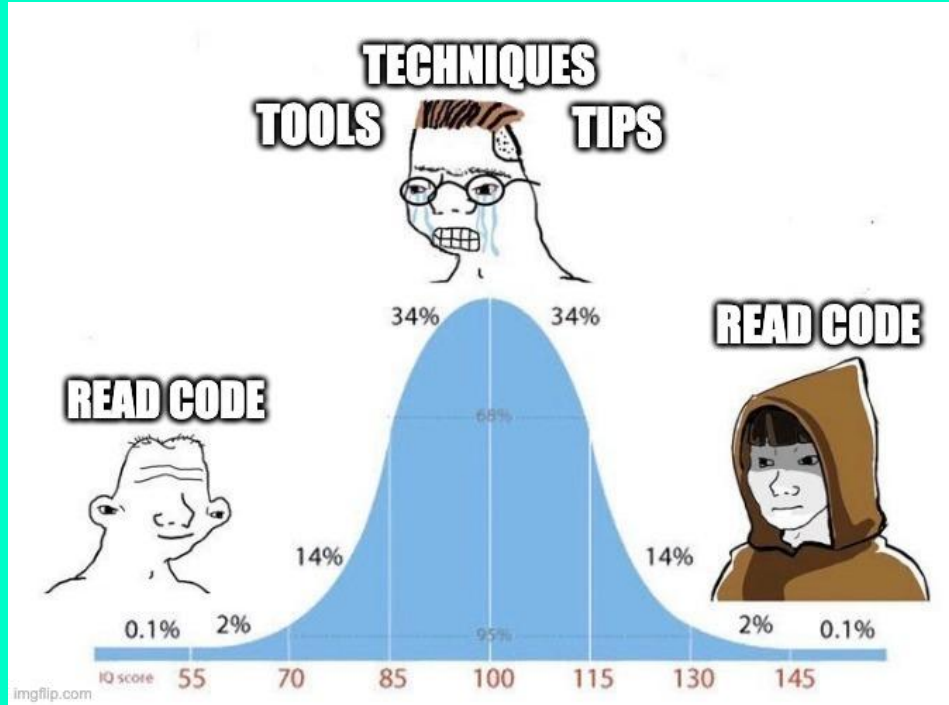


Spec/Doc/Comment/Comms



Infer: Clues in code

TIP #40: ~~JUST DO~~ FIND IT ✓



No bug-free code



Read code → Find bugs



Prepare → Practice →
Patience

TIP #41 (DEVS)

- 🔧 Specify -> Design -> Code -> Test
- 🔧 Assets - Actors - Actions
- 🔧 Security Mindset -> Shift Left
- 🔧 Testing Properties/Invariants
- 🔧 Frameworks & Extensions
- 🔧 Find Your Tools/Techniques
- 🔧 Use <> Abuse
- 🔧 Adversarial Environment -> Defensive Programming
- 🔧 Audits vs Contests vs Bug Bounties
- 🔧 Code - Coverage - Communications

TIP #42 (SRS)



Read Code



Challenge Assumptions



Documentation & Intuition



Pre-conditions & Post-conditions



Coverage & Leverage



Scoping & Pacing



Dive-in & Step-out



Past -> Learn



Prepare-Practice-Patience



TLC: Code & Yourself

RACE



Short (16 minutes) quiz



8 MCQs



Password: TICKER-IS-ETH



Submit by tomorrow



Top 2: Win DSS'25 tickets

