Security Engineering:

Zcash Ecosystem Security and Learning From Safety-Critical Disciplines

Taylor Hornby

Part 1: The ZecSec Project

Funded by a Zcash Community Grant, I...

- Provide security audits to community projects.
- Offer security advice through open office-hours
- Help projects respond to security bugs
- (I think about scalable protocol design too.)

more info - https://zecsec.com

Contributions

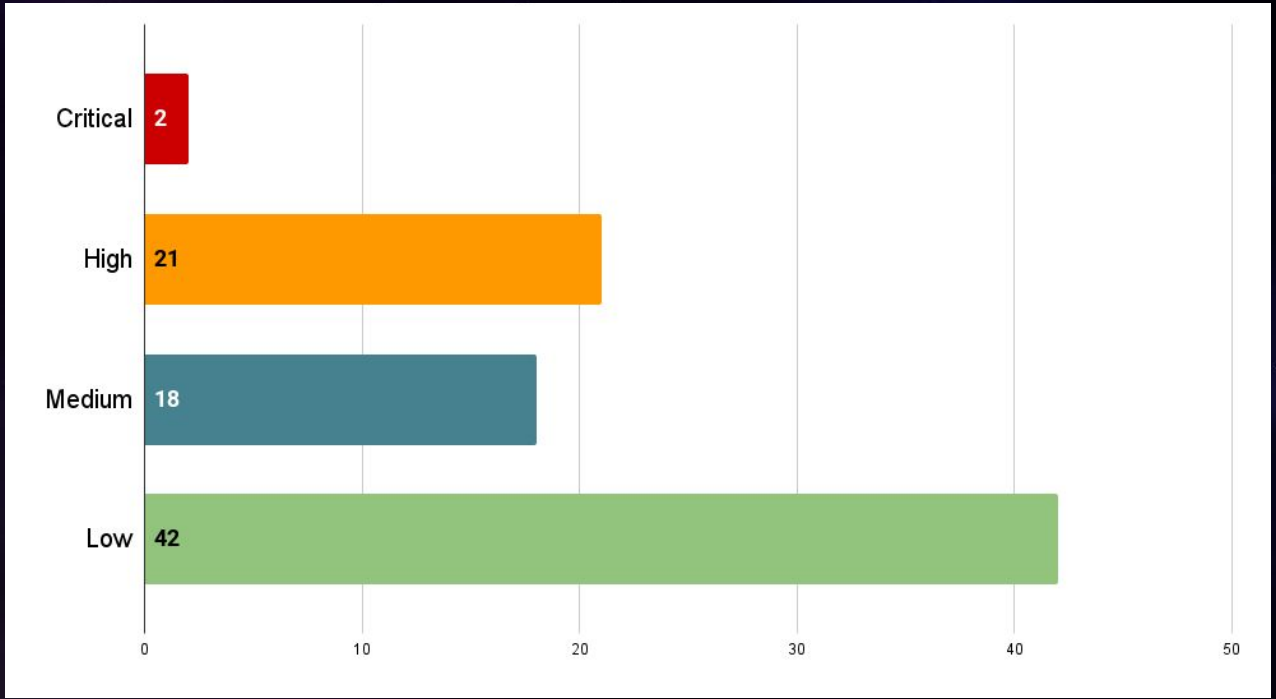
Audits:

- zecwallet-lite-cli
- Ywallet
- lightwalletd
- Nighthawk’s lightwalletd deployment infrastructure
- Free2z
- zGO
- Ledger hardware wallets: Zondax’s & Hanh’s
- ZIPs for Zcash Shielded Assets

Other:

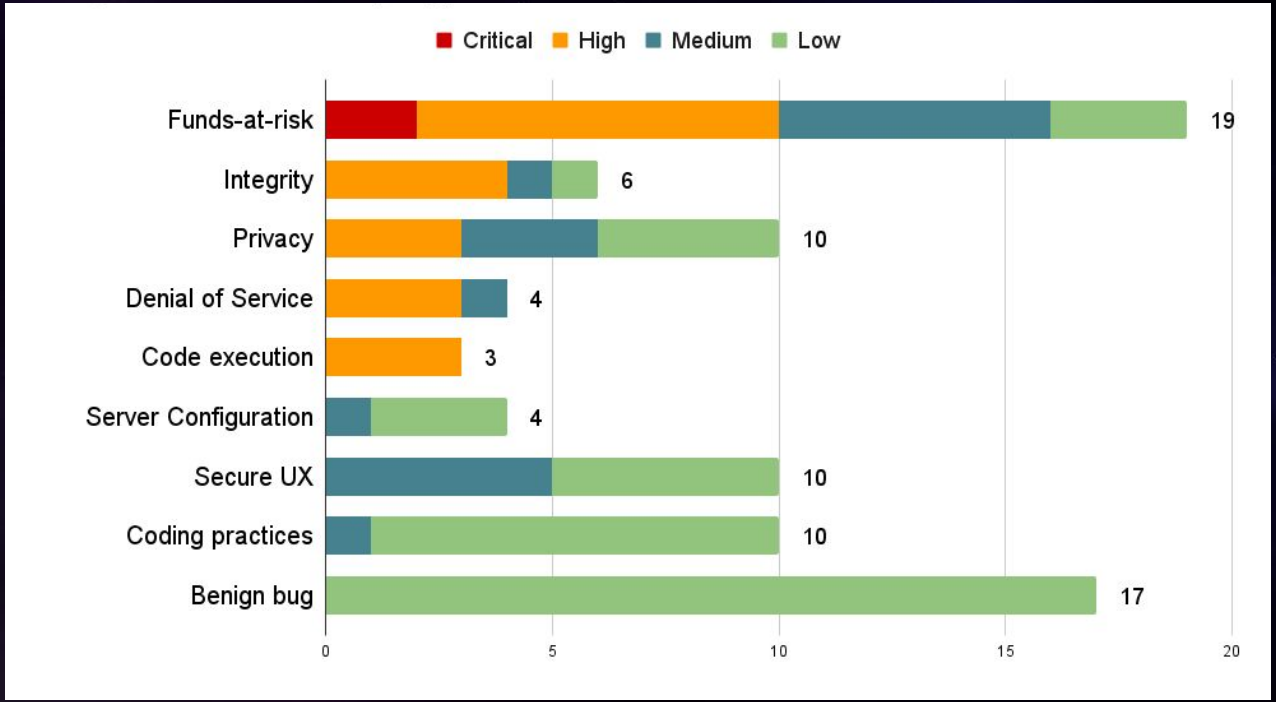
- Survey alternatives to transaction scanning.
- Misc. bug investigation / security response help.
- A lot of forum discussion :-)

Bugs Discovered, by Severity



83 bugs total, 41 medium-or-worse

Bugs Discovered, by Type



Conclusions So Far

Security review is effective, *especially* in code that handles funds.

Most bugs were found by manual review + a creative bug discovery process.

I *missed* some bugs, e.g. in the Zcash Shielded Assets ZIPs.

Most projects respond to bug reports quickly and thoroughly.

Some projects don’t; not enough resources. Should fund bug-fix engineers.

Building security review/fixes into the grant process itself would help.

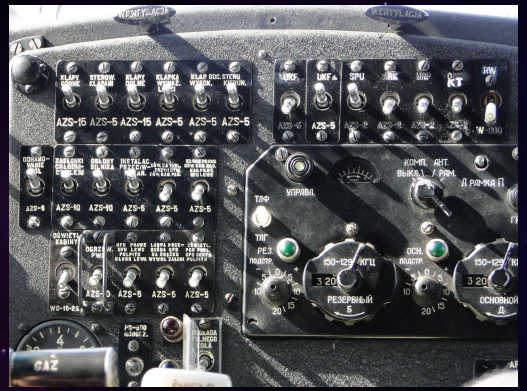
Part 2:

Security Engineering: Learning From Safety-Critical Disciplines

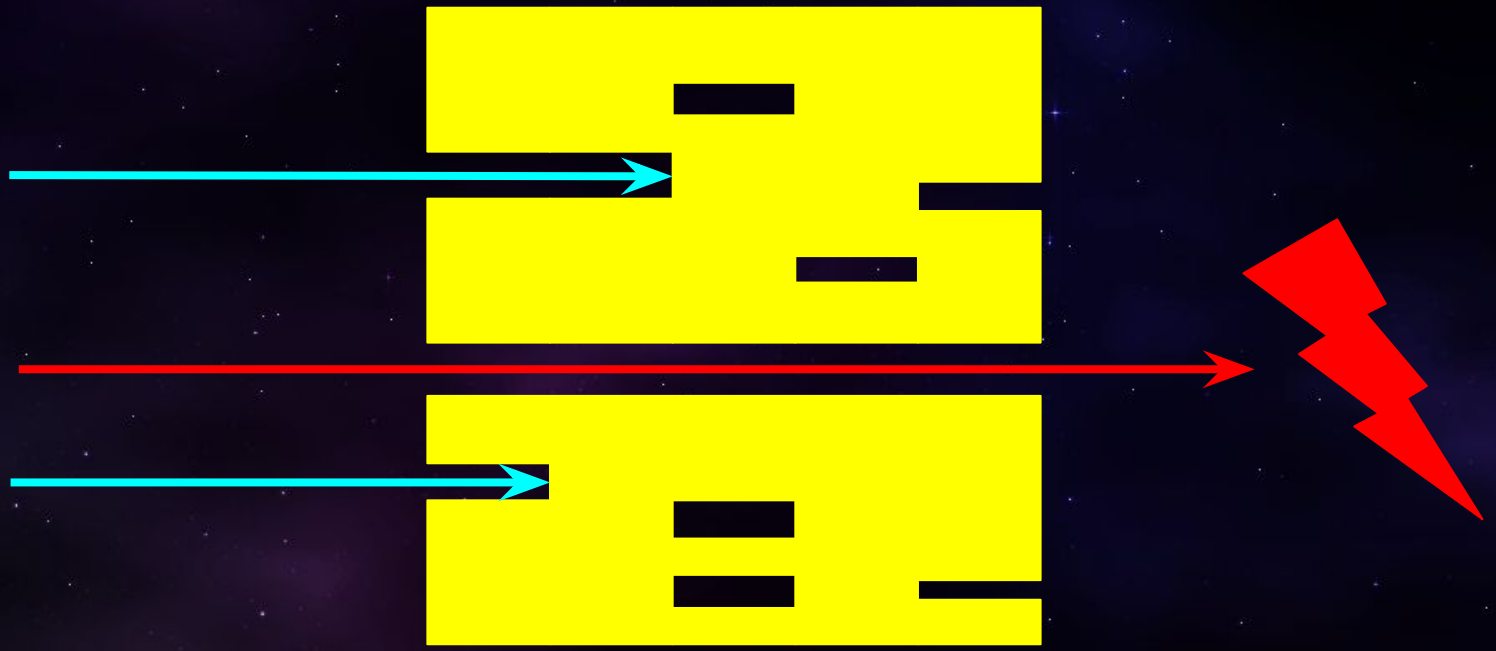
Tenet 1: Learning from Failure

We owe our safety to the people who have died in disasters, and to the investigators who worked tirelessly to find out why.

Tenet 2: “Human Error”



Swiss Cheese Model



Tenet 3: Blameless Investigation

“The sole objective of the investigation and the Final Report is the prevention of accidents.” - ICAO

Tenets of Safety Engineering

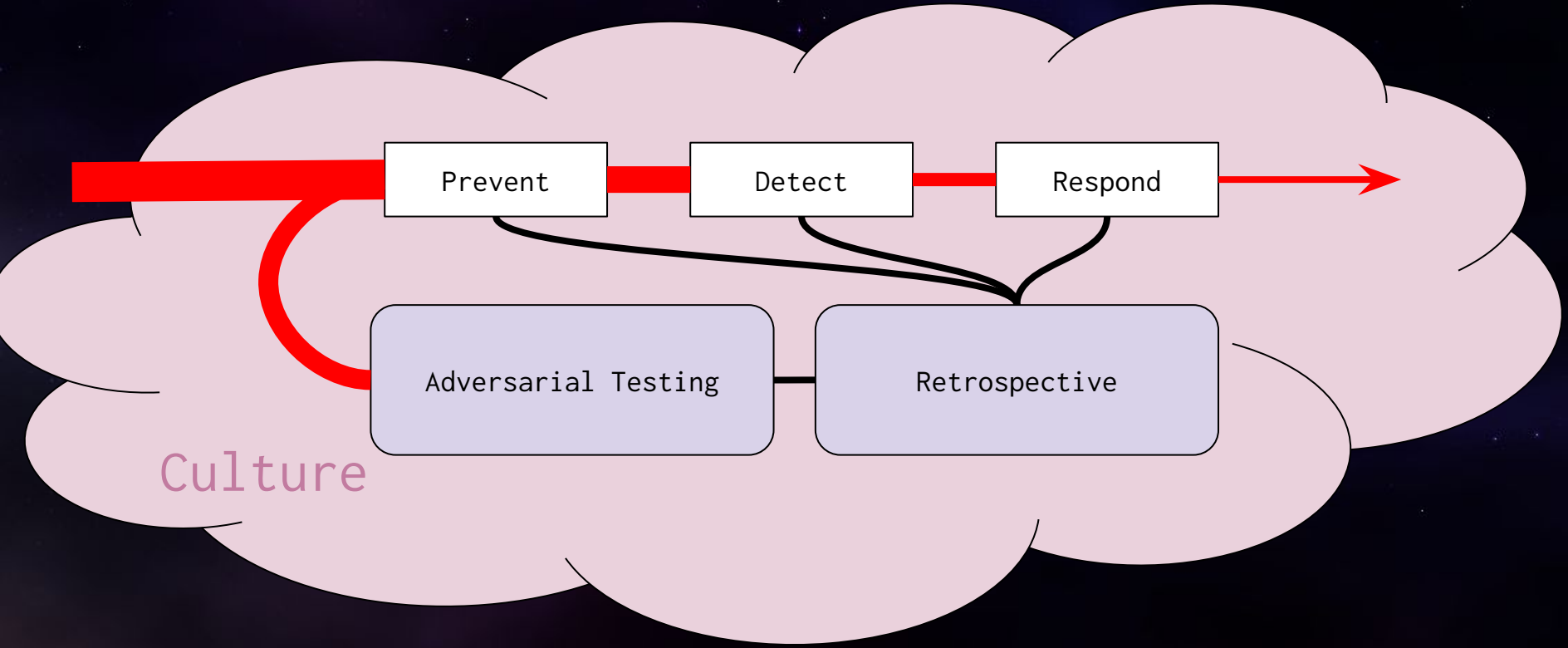
Learn from failure

Design for human error

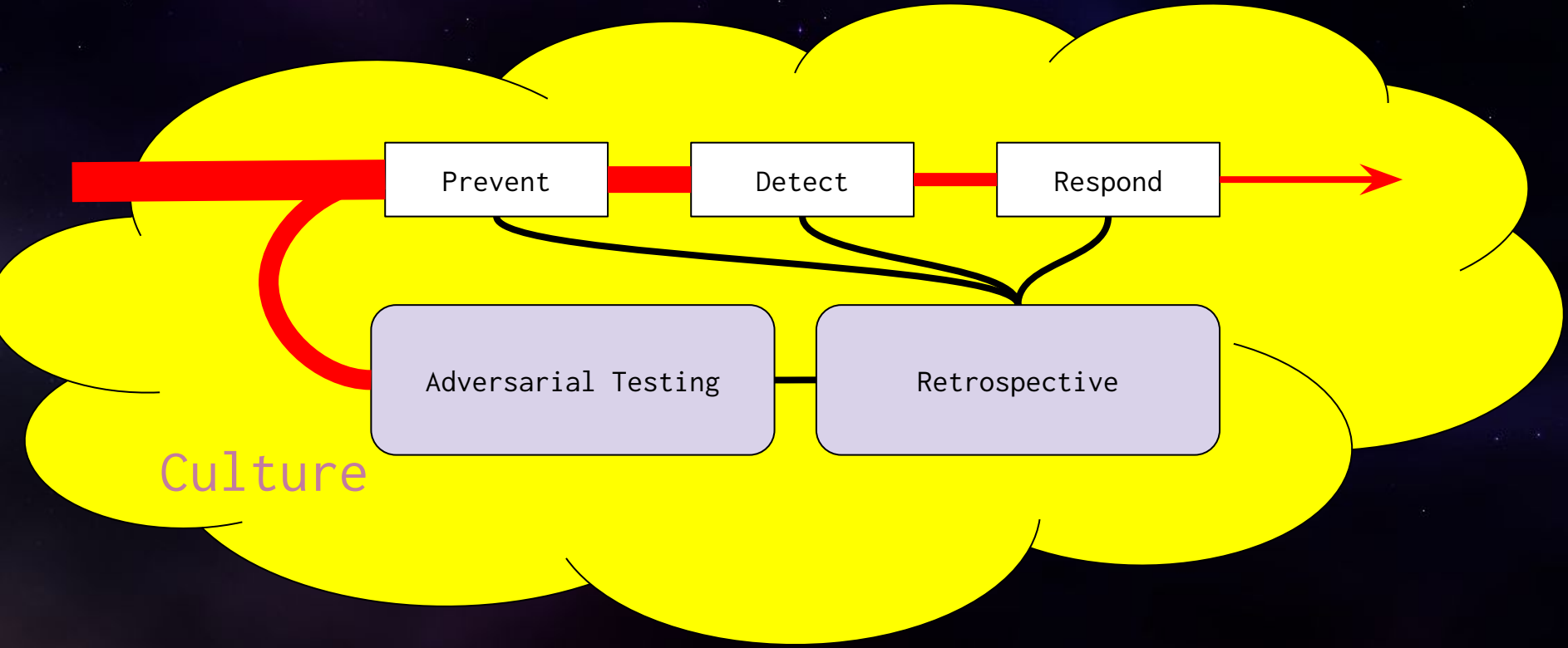
Create a blameless environment

Safety vs. Security

Security Engineering



Security Engineering



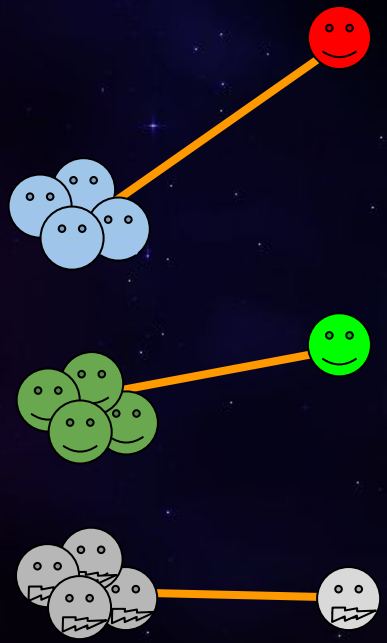
Culture: Hazards

Blameful Environment

Judicial Mindset

Scholarly Mindset

Authority Gradients



“15:30:21 HOT-1: got any ideas?”

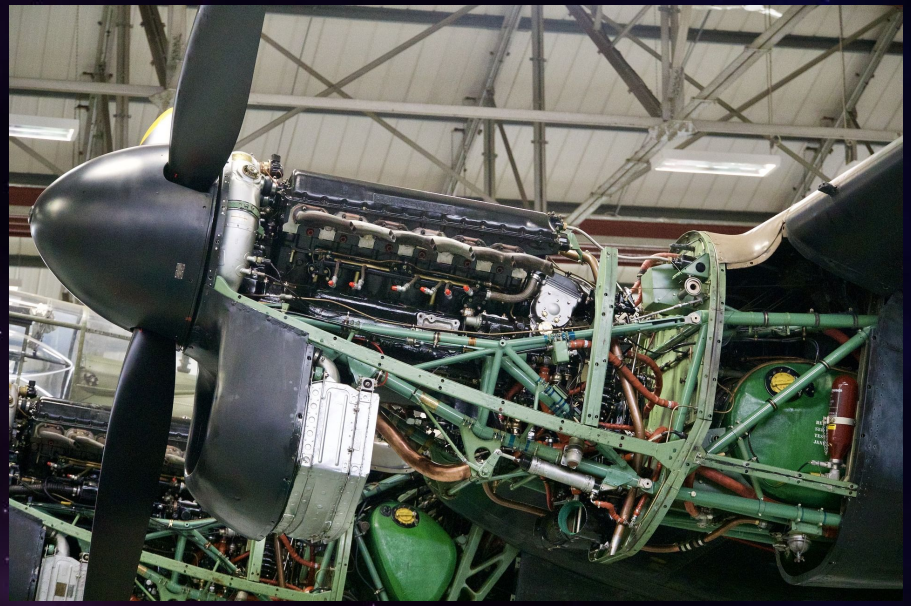
Get-there-itis

Habituation / Normalization of Deviance

Task Saturation

Alarm Fatigue

Condition-Based Maintenance



vs. Periodic Maintenance

Culture: Tools

Value Blamelessness and Fallibility

"I don't know"

"I made a mistake, when I *X*, *Y* happened;
next time I will Z."

Make sure everyone is comfortable saying these things!

“Hey That’s Weird”

“Hey is this email phishing?”

“I got this weird error on my computer.”

“Am I really supposed to transfer these funds?”

“I don’t recognize this script on our website.”

“My personal email was involved in a data breach.”

“I can’t log in.”

Time Out

A systematic time-out in the operating room just before incision has been introduced the last two decades to help prevent wrong site surgeries and other surgical never events.

Positive Transfer of Control

“Your aircraft.”

“My aircraft.”

Pilot Flying vs. Pilot Monitoring

Checklists

1. Read
2. Execute
3. Say aloud
4. Confirm

Checklists: Beware of Rhyming

- Auxiliary fuel pump
- Flight controls
- Instruments and radios
- Landing gear position lights
- Altimeter
- Directional gyro
- Fuel gauges

… … …

a. …Off
a. …Free and correct
a. …Checked and set
a. …Checked
a. …Set
a. …Set
a. …Checked

Pointing and Calling

Culture: Summary

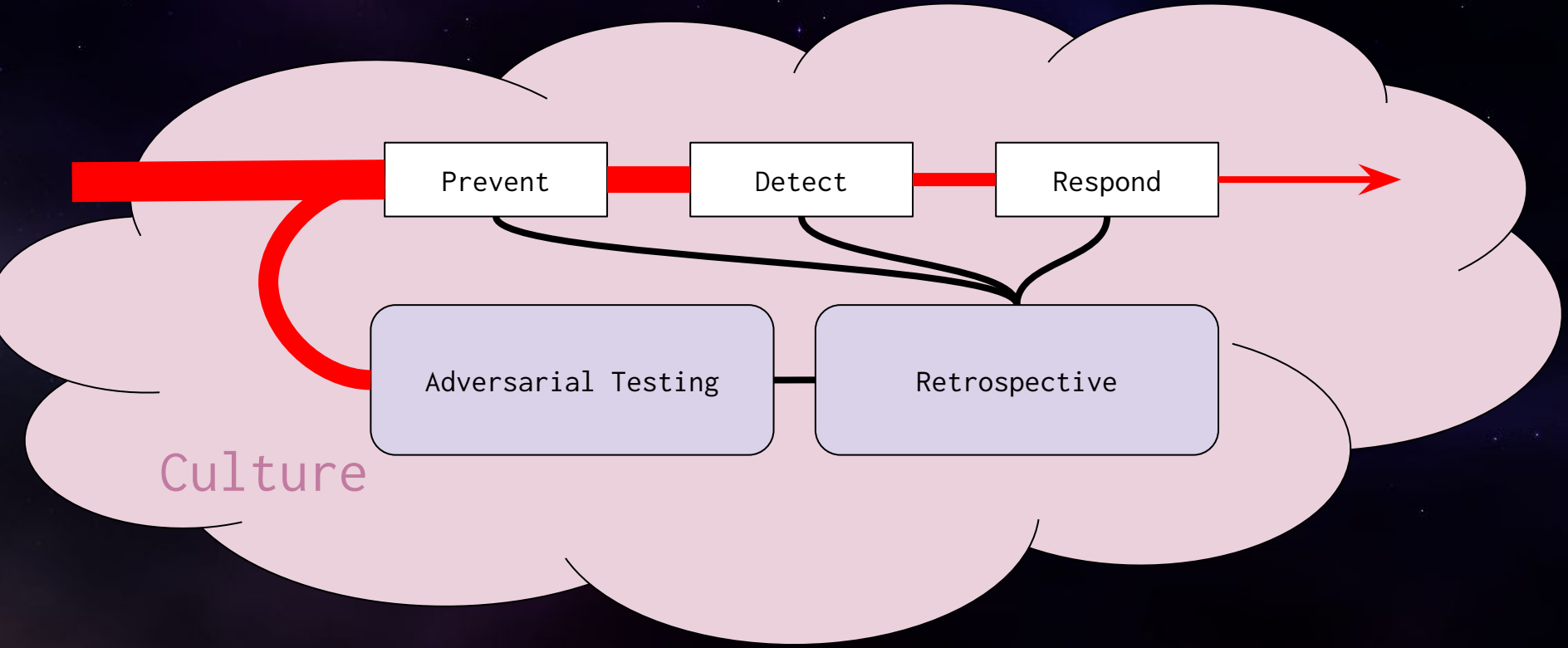
Avoid:

- Blameful environments.
- Steep or flat authority gradients.
- Get-there-itis.
- Normalization of deviance.
- Task saturation
- Alarm fatigue.
- Condition-based maintenance.

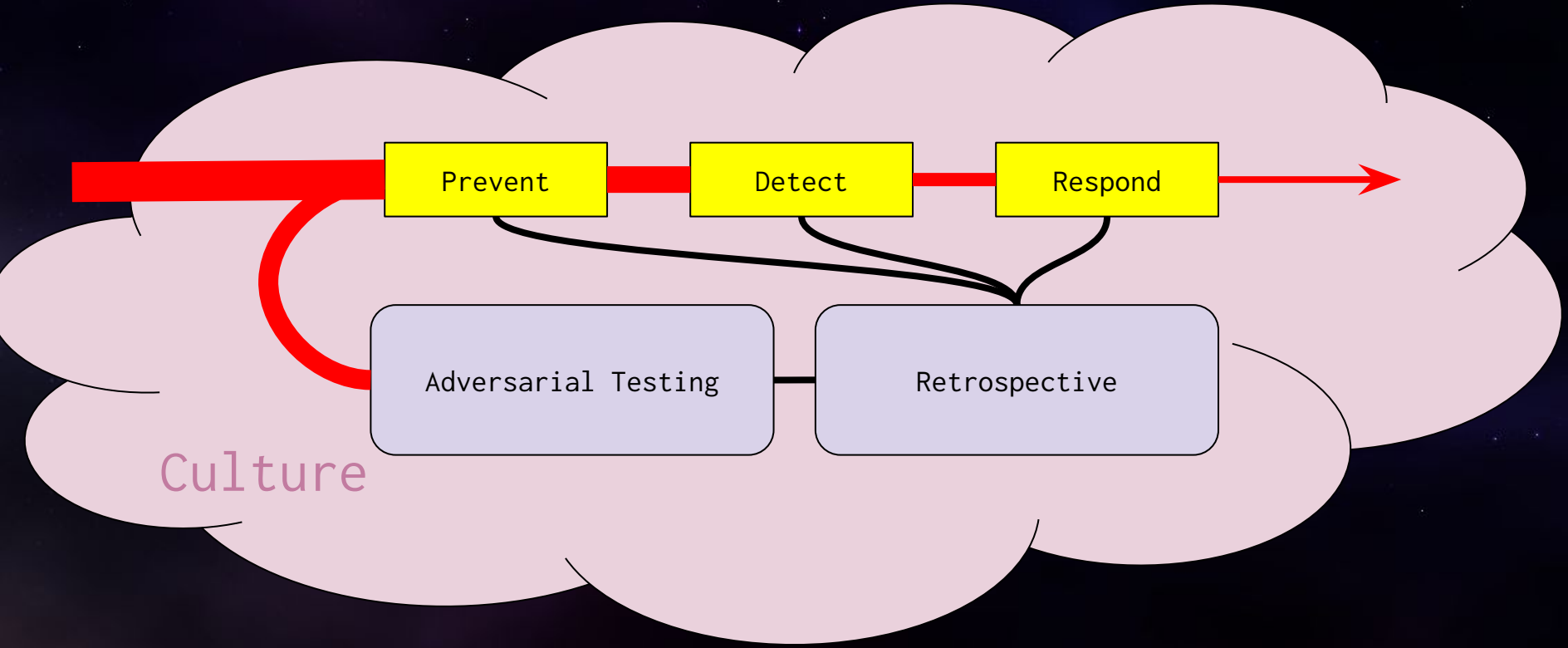
Implement:

- Blameless investigation.
- Healthy authority gradient.
- Open discussion of fallibility.
- “Hey That’s Weird”
- Time Outs
- Positive Transfer of Control
- Checklists
- Pointing and Calling

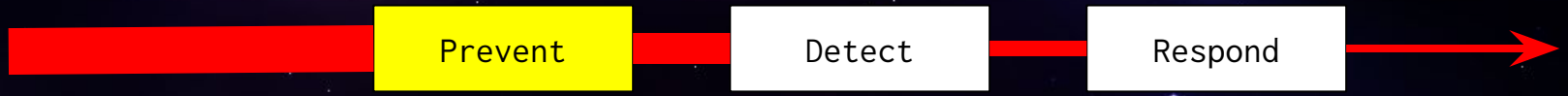
Security Engineering



Security Engineering



Prevent: Threat Model



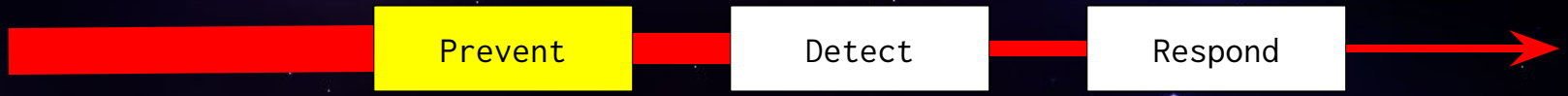
Enumerate all possible kinds of attacks against your system
(complicated, technical)

and/or

Describe *expected security properties* in language users understand
(simpler, considers user experience, harder to do)

“I expect that nobody except me and my friend can read our encrypted messages unless someone breaks into one of our phones.”

Prevent: Full Stack Security



Organization

Password Manager

2FA

Accessible Security Support

Org-Wide Security Culture

Code

Memory-safe Languages

Parameterized Queries

Unit Tests

Code Review Policies

Audits

DevOps

Gitops

Automatic Updates

Firewalls

Avoid Storing Data

Detect

No matter how good your prevention is...

Machines get malware

Networks get breached

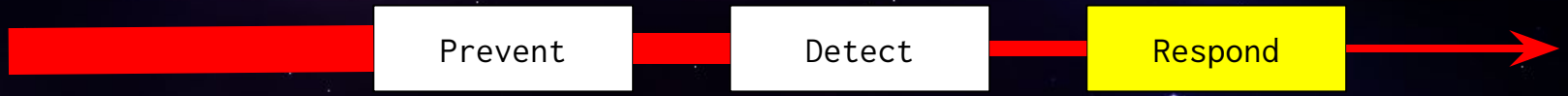
Bugs get exploited

Accounts get hacked

Data gets leaked

You need systems in place to detect intrusions.

Respond



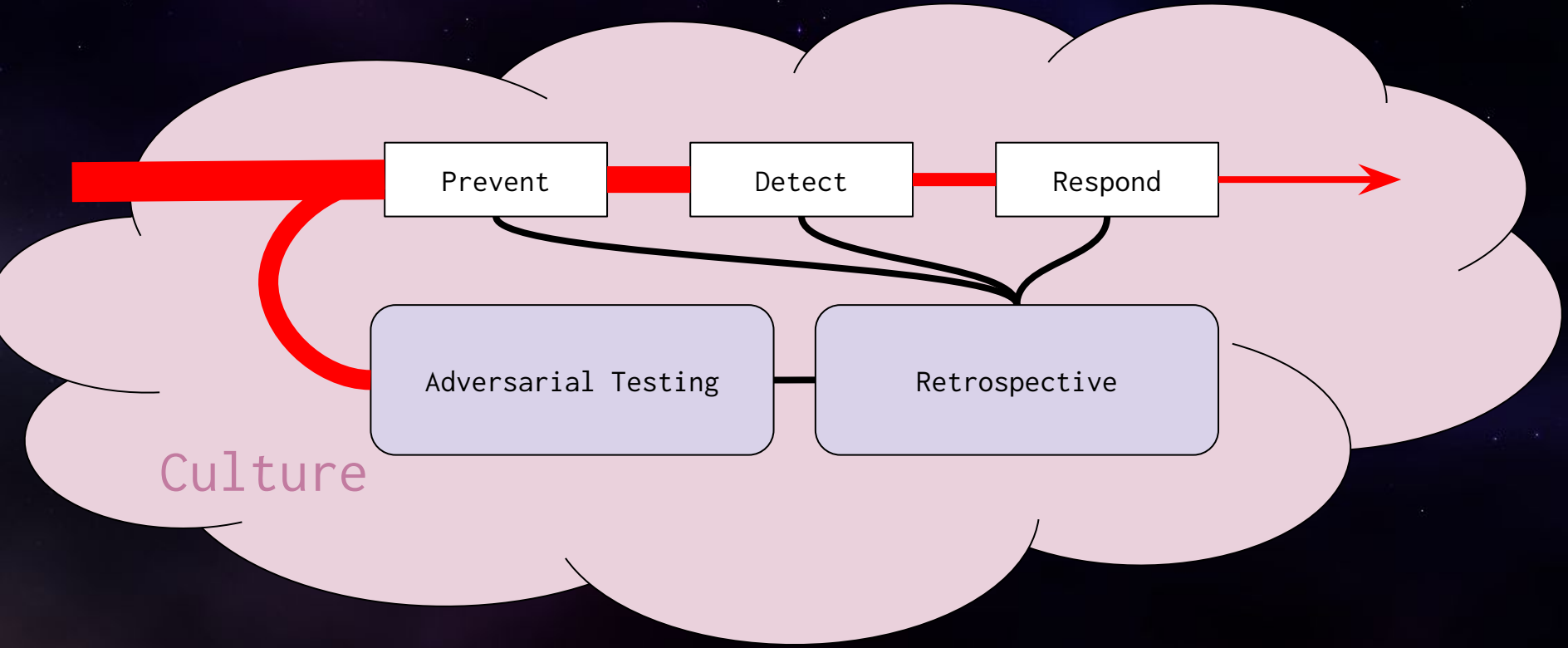
Create *and Practice* an Incident Response Procedure

- Clear severity rankings (drop-everything vs. for-next-sprint)
- Roles (investigators, decision-makers)
- How to communicate internally
- Public comms / PR

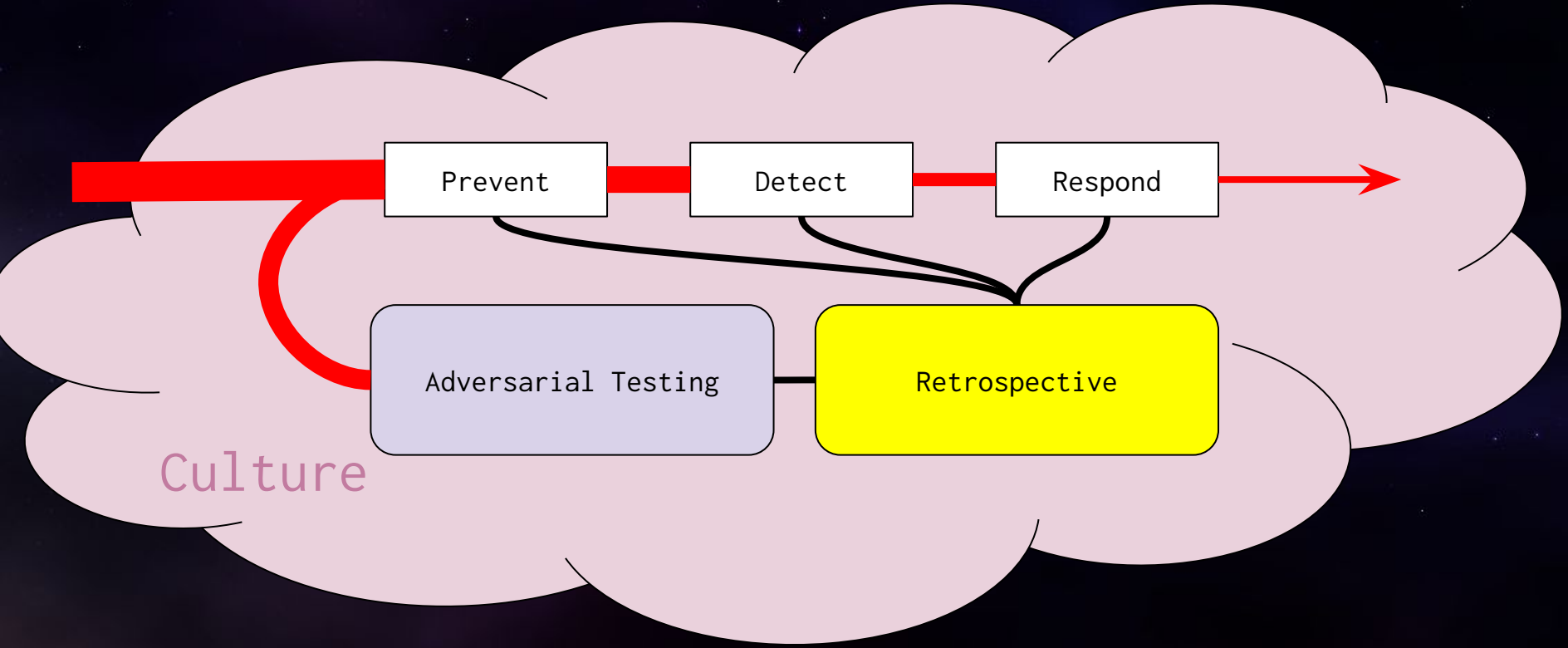
Prepare for likely disasters

- Make it easy to restore servers to known-good state
- Know which accounts' passwords you'll need to reset
- Have engineers on-call to make bug fixes

Security Engineering



Security Engineering



The Retrospective

Before meeting:

1. Collect and share technical information

At meeting:

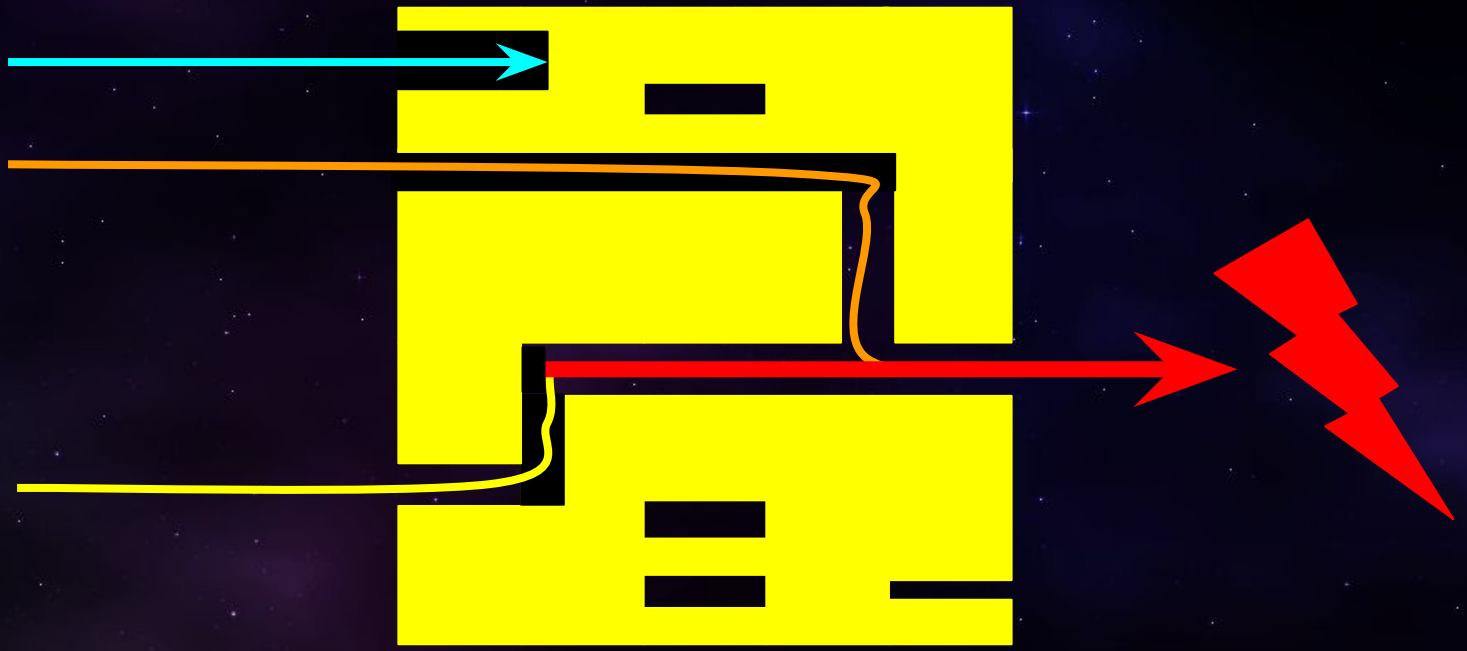
2. What went well
3. What didn’t go well
4. Go around the room & record all ideas (free-form)
5. Action items

After:

Document, Brief, Review

Retro: Ideation

Retro: Root Cause Seduction



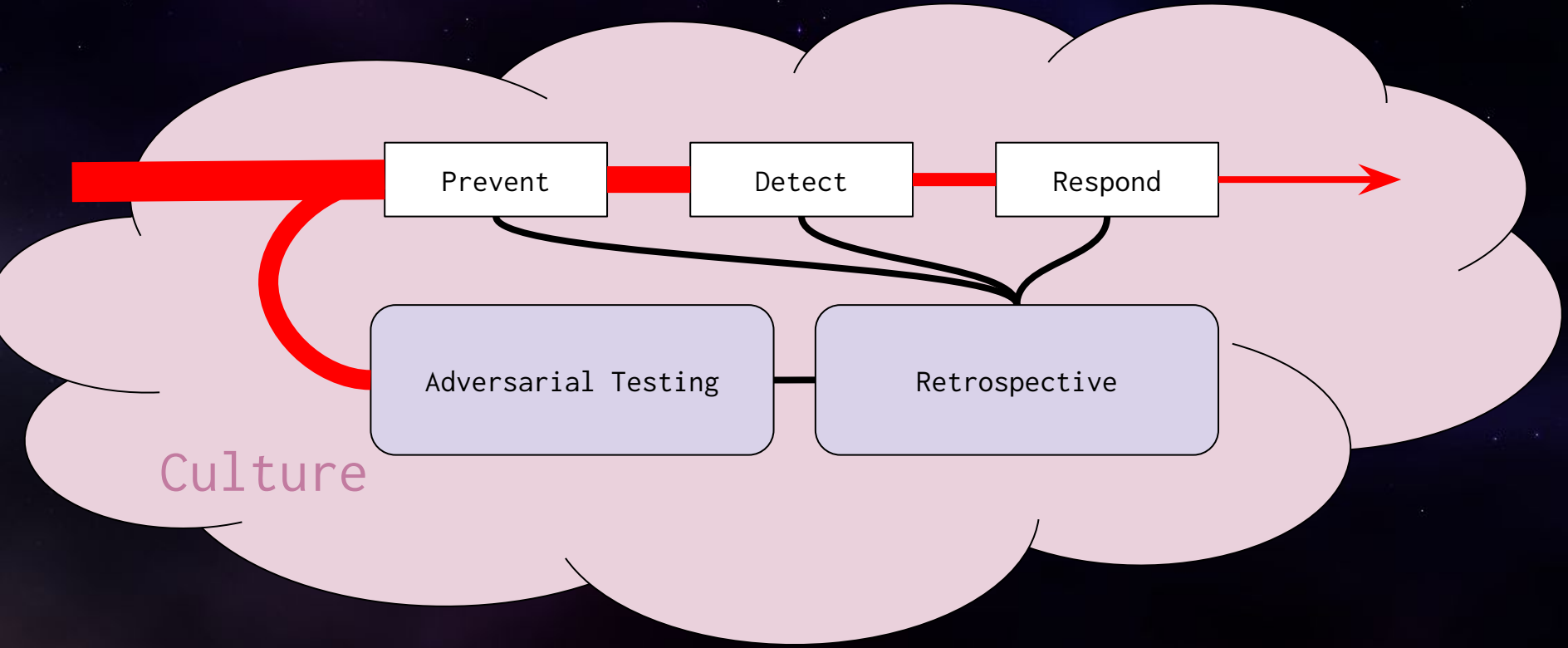
Retro: The Final Report

Document (1-2 pages)

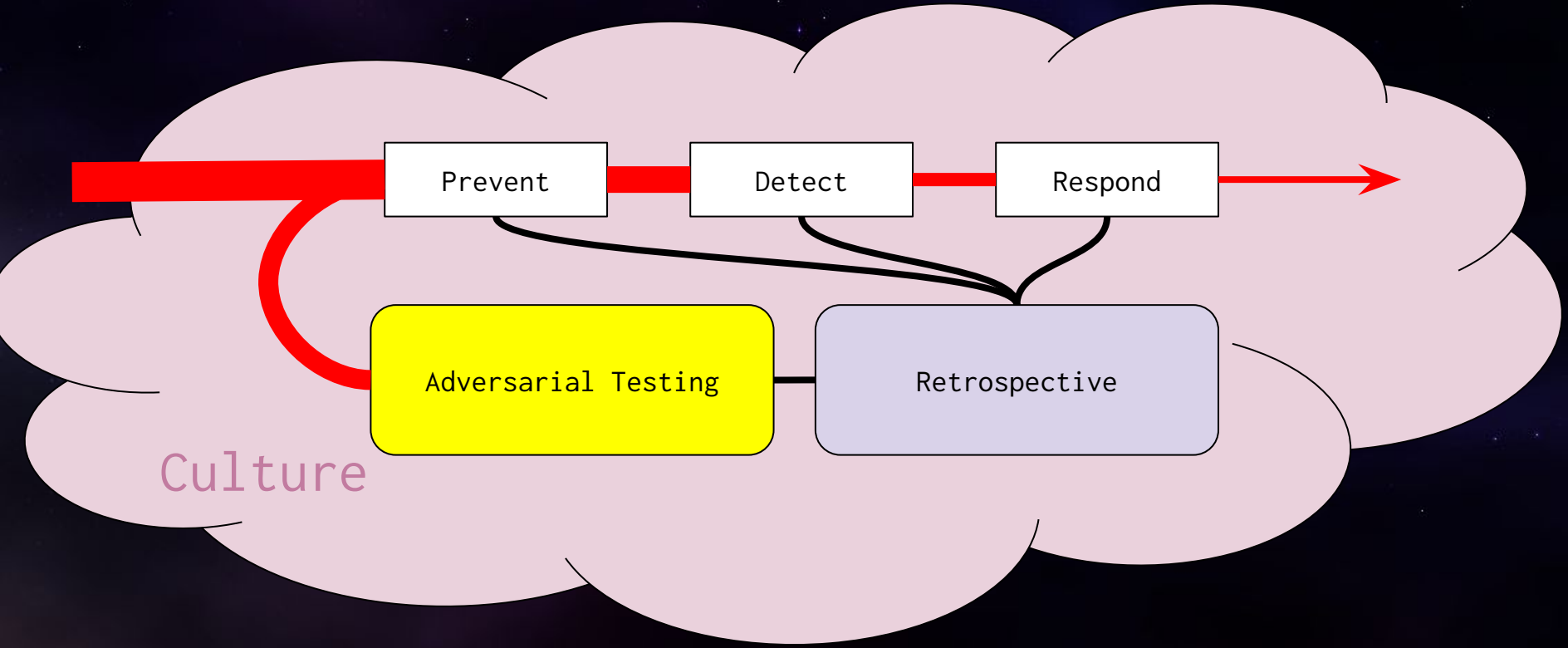
Brief (+ 2 weeks)

Review (+ 3-6 months)

Security Engineering



Security Engineering



You Have Bugs

Security bugs exist, you just haven’t found them.

You need to find bugs to understand how your processes are failing.

Invest in bug discovery to fix current bugs *and prevents future ones*.

Adversarial Mindset

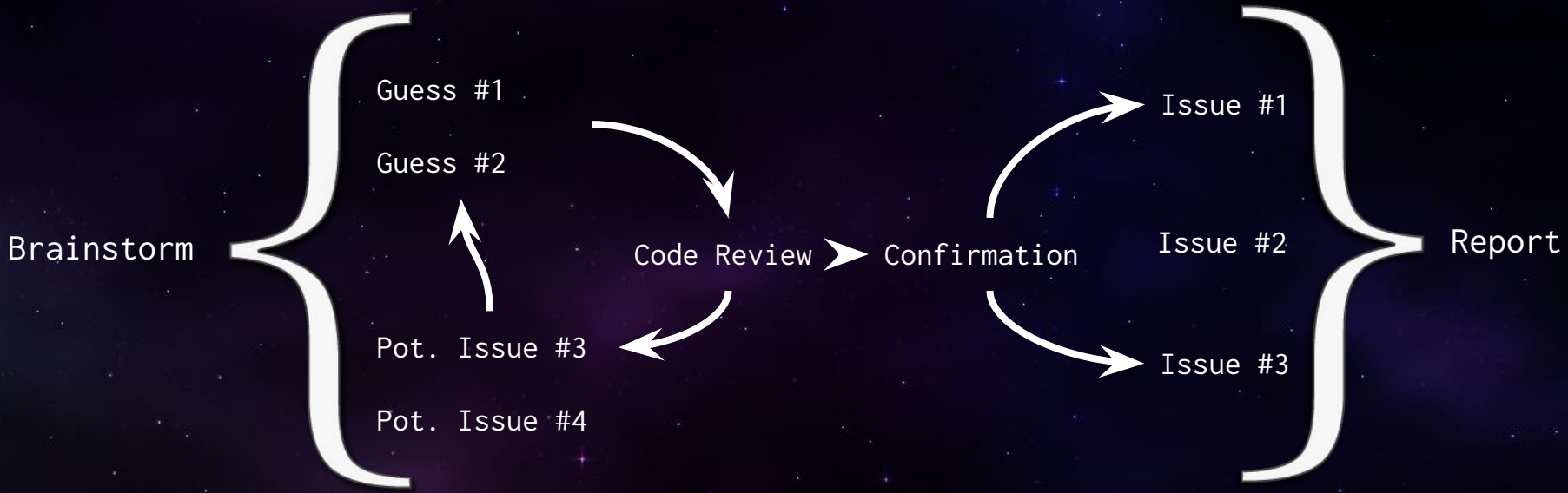
```rust
struct Account {
    name: String,
    balance: i32,
}

fn transfer(from: &mut Account, to: &mut Account, amount: i32) -> Result<(), String> {
    if from.balance < amount {
        return Err("Not enough funds.".to_string());
    }

    to.balance += amount;
    from.balance -= amount;

    Ok(())
}
```

Security Audit: Process



Internal vs. External Audits

Internal:

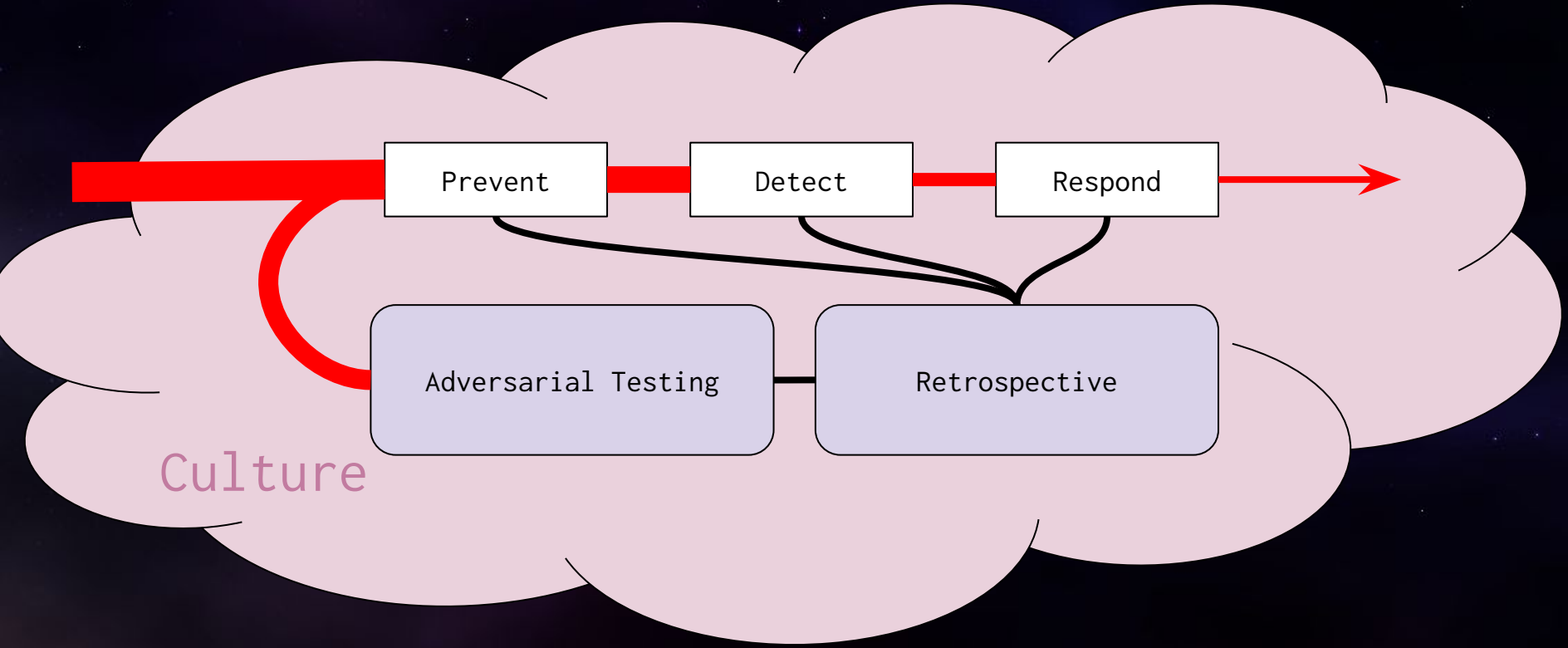
- You’re the most familiar with your code.
- Develops in-house adversarial mindset.

External:

- More likely to question entrenched assumptions.
- Fresh eyes can see bugs that are “habituated” in developers’ minds.

Do both!

Security Engineering



Tenets of ~~Safety~~ Engineering
Security

Learn from failure

Design for human error

Create a blameless environment

Prevent, Detect, Respond

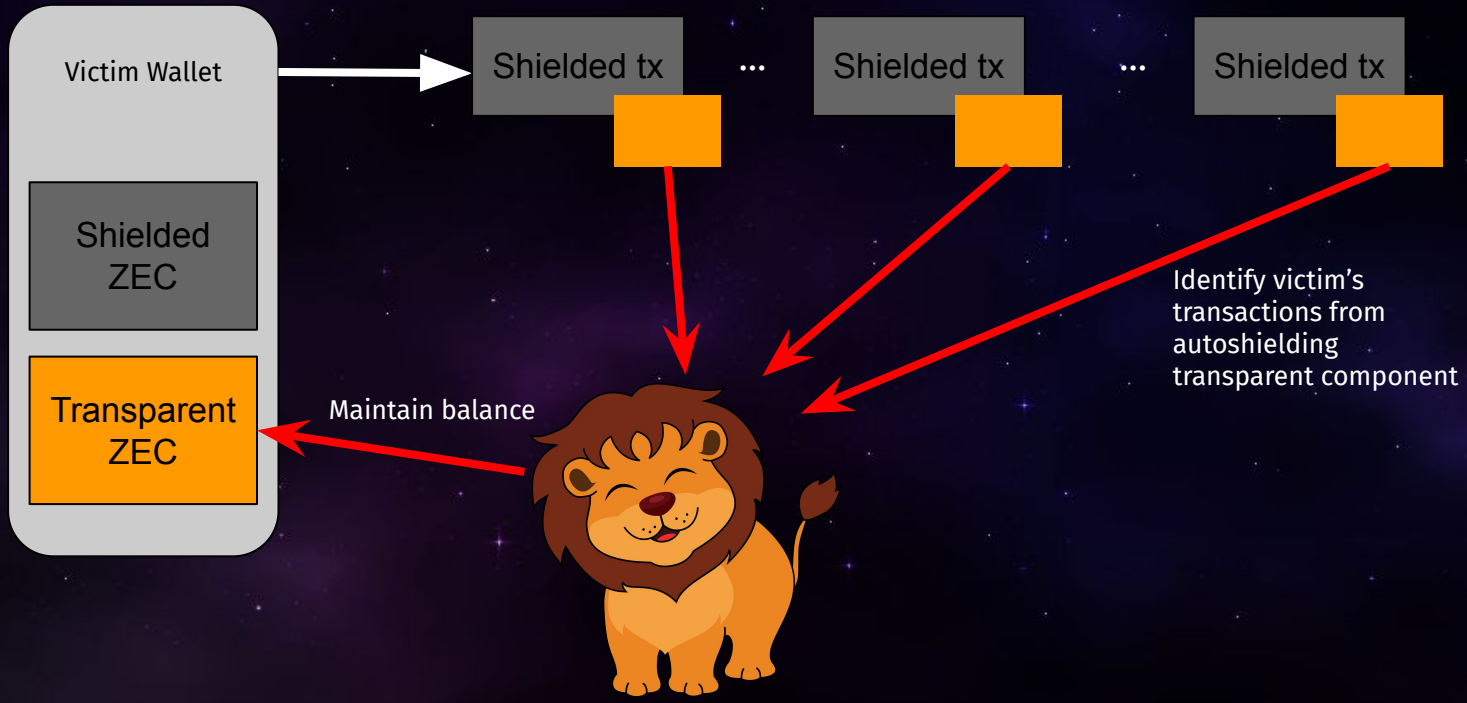
Attack your own systems

Bonus Part:

Bugs!

MITM thru memo-based contact list

Autoshielding Deanonymization



Paying for orders by paying yourself

```
for all customer addresses z:
    for all memos m sent to z:
        if m == “ORDERPAYMENT:<id>” and id is a valid order ID
            and payment is the correct amount:
                Mark the order as paid.
```

The attack:

1. Sign up as a customer of the payment processor yourself.
2. Get the order ID for the order you want to pay.
3. Send the payment to *your own address*.
4. The processing server will mark the order as paid.

Hardware wallet state machine bug

```
typedef enum {
    IDLE,
    T_IN,
    T_OUT,
    S_OUT,
    O_ACTION,
    FEE,
    SIGN,
} signing_stage_t;
```

```
1  int change_stage(uint8_t new_stage) {
2      if (new_stage != G_context.signing_ctx.stage + 1) {
3          reset_app();
4          return io_send_sw(SW_BAD_STATE);
5      }
6
7      cx_hash_t *ph = (cx_hash_t *)&G_context.hasher;
8
9      switch (new_stage) {
10         case T_OUT:
11         // ...
12         // ... cases that handle stages S_OUT, O_ACTION, and FEE
13         // originally there was no default case
14     }
15     G_context.signing_ctx.stage = new_stage;
16
17     return io_send_sw(SW_OK);
18 }
```

Tenets of ~~Safety~~ Engineering
Security

Learn from failure

Design for human error

Create a blameless environment

Prevent, Detect, Respond

Attack your own systems

https://zecsec.com/posts/security-engineering/

</presentation>