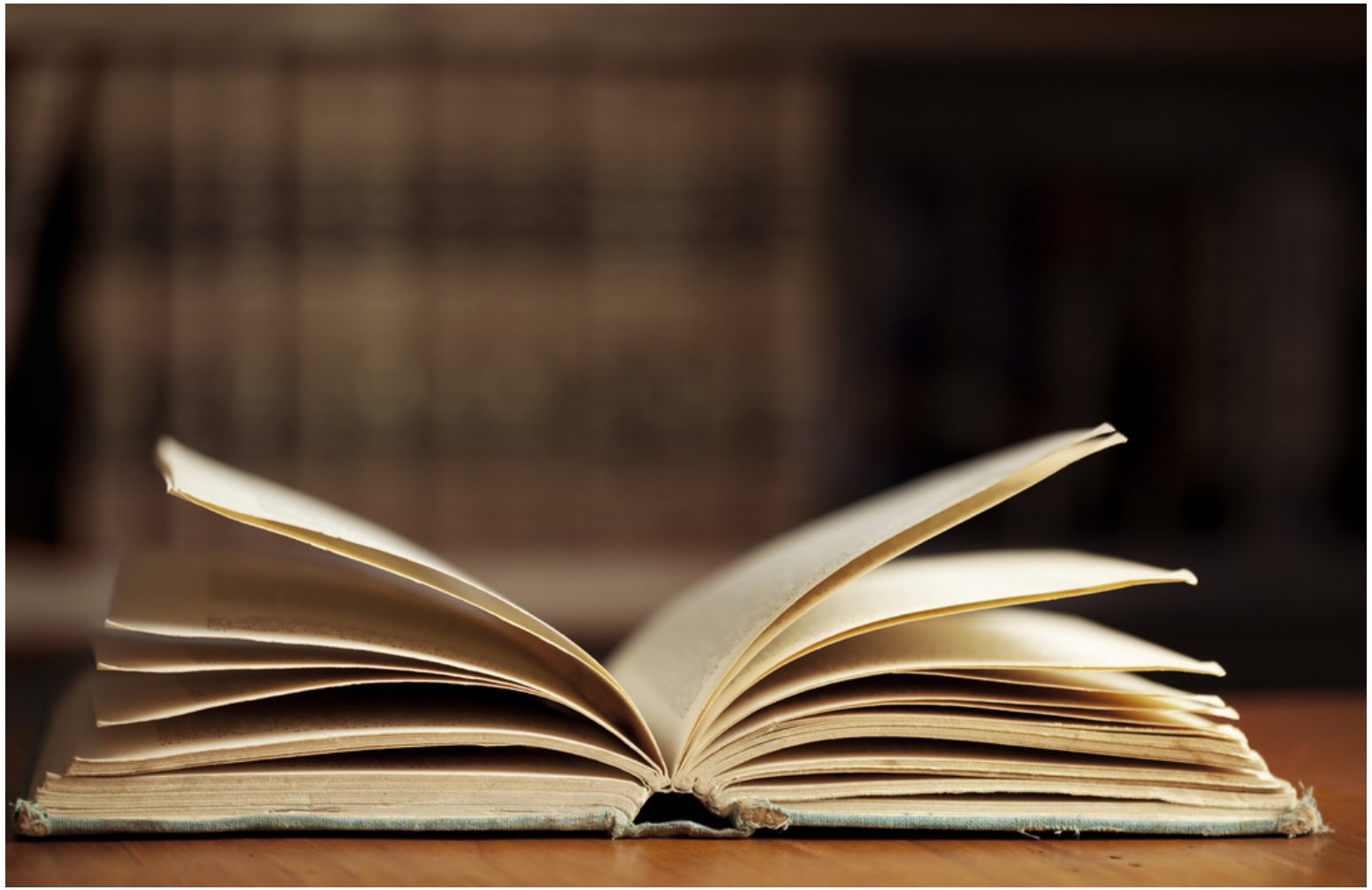
# The Open Source Fortress

# @iosifache

- Previous lives
  - 1.5 years in the Romanian Army
  - Tech lead in a cybersec startup
- Now software security engineer in the Ubuntu Security Team
- Bucharest-based
- Powered by Americanos
- Long-distance running as a hobby



## Roundcube Webmail

- Open source, browser-based IMAP client
- Hosted on GitHub
- With 5.2k stars (as per November 13, 2023)
- Written in XHTML, CSS, JavaScript (with jQuery), and PHP

## Q: What are we missing here?

1. `/installer/index.php` route stores the user-controlled configuration in `rcube->config`.
2. When an email with a non-standard format is received, `rcube::exec` executes the output of `getCommand`.

```php
private static function getCommand($opt_name)
{
    static $error = [];

    $cmd = rcube::get_instance()->config->get($opt_name);

    if (empty($cmd)) {
        return false;
    }

    if (preg_match('/^(convert|identify)(\.exe)?$/i', $cmd)) {
        return $cmd;
    }

    // Executable must exist, also disallow network shares on Windows
    if ($cmd[0] != "\\" && file_exists($cmd)) {
        return $cmd;
    }

    if (empty($error[$opt_name])) {
        rcube::raise_error("Invalid $opt_name: $cmd", true, false);
        $error[$opt_name] = true;
    }

    return false;
}
```

> From `program/lib/Roundcube/rcube_image.php`

## A: Input sanitisation

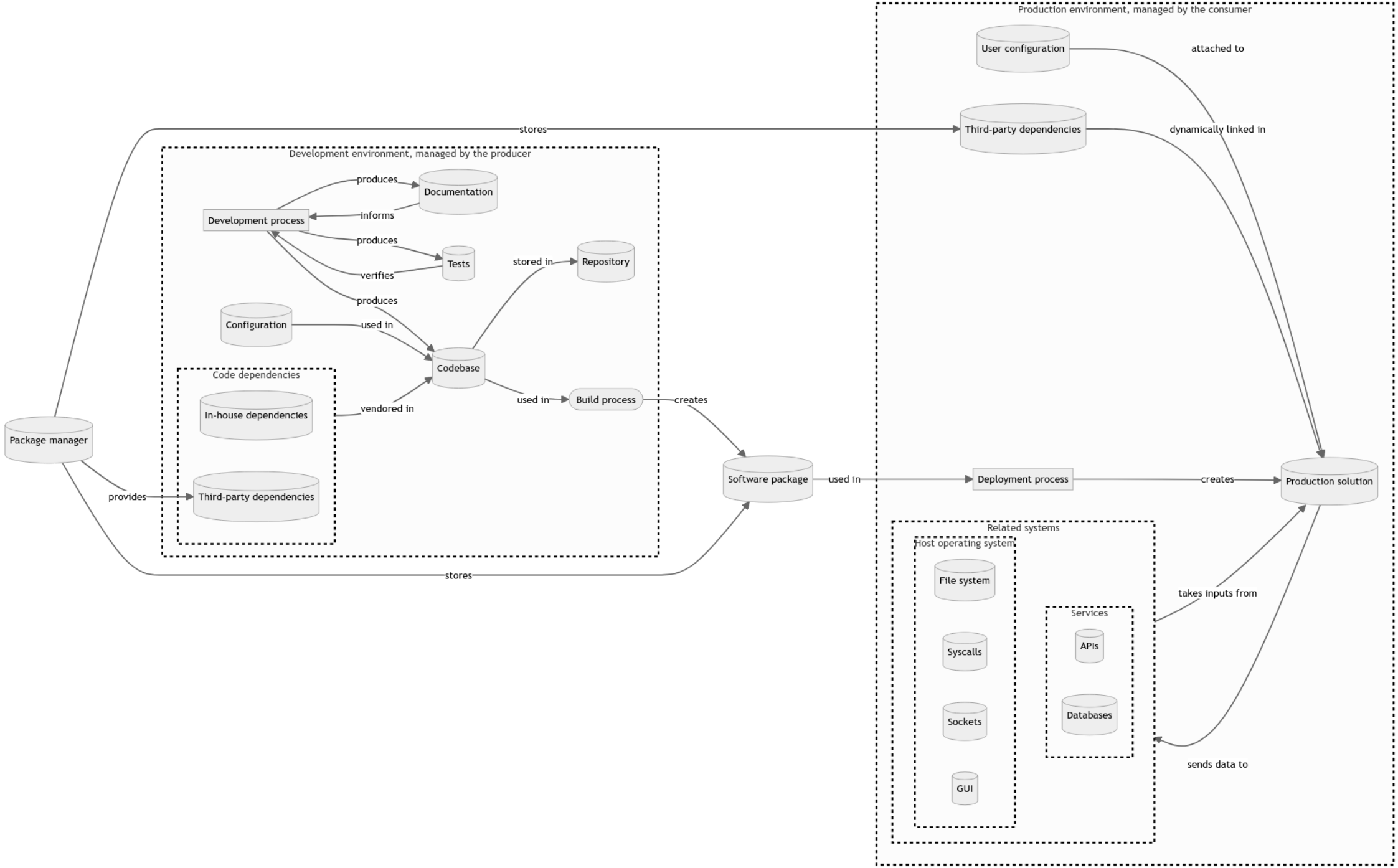
- CVE-2020-12641
- Many vulnerable configuration items, leading to arbitrary code execution
- 7.66% EPSS and 9.8 CVSS
- Used by APT28 to compromise Ukrainian organisations' servers
- Added by CISA in the Known Exploited Vulnerabilities Catalogue

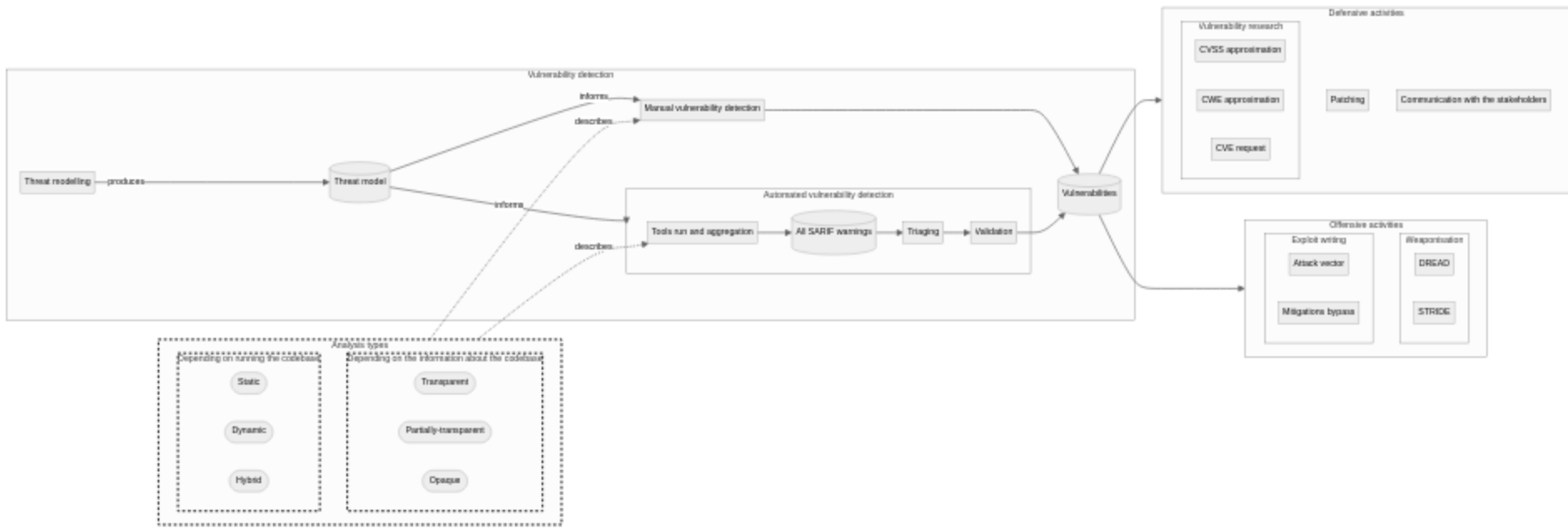
## But … Was it preventable?

- Yes, but not with standard linters or scanners
- Taint analysis as a possible solution
  - `rcube->config` as a tainted data source
  - `rcube::exec` as a sensitive sink

# The Open Source Fortress

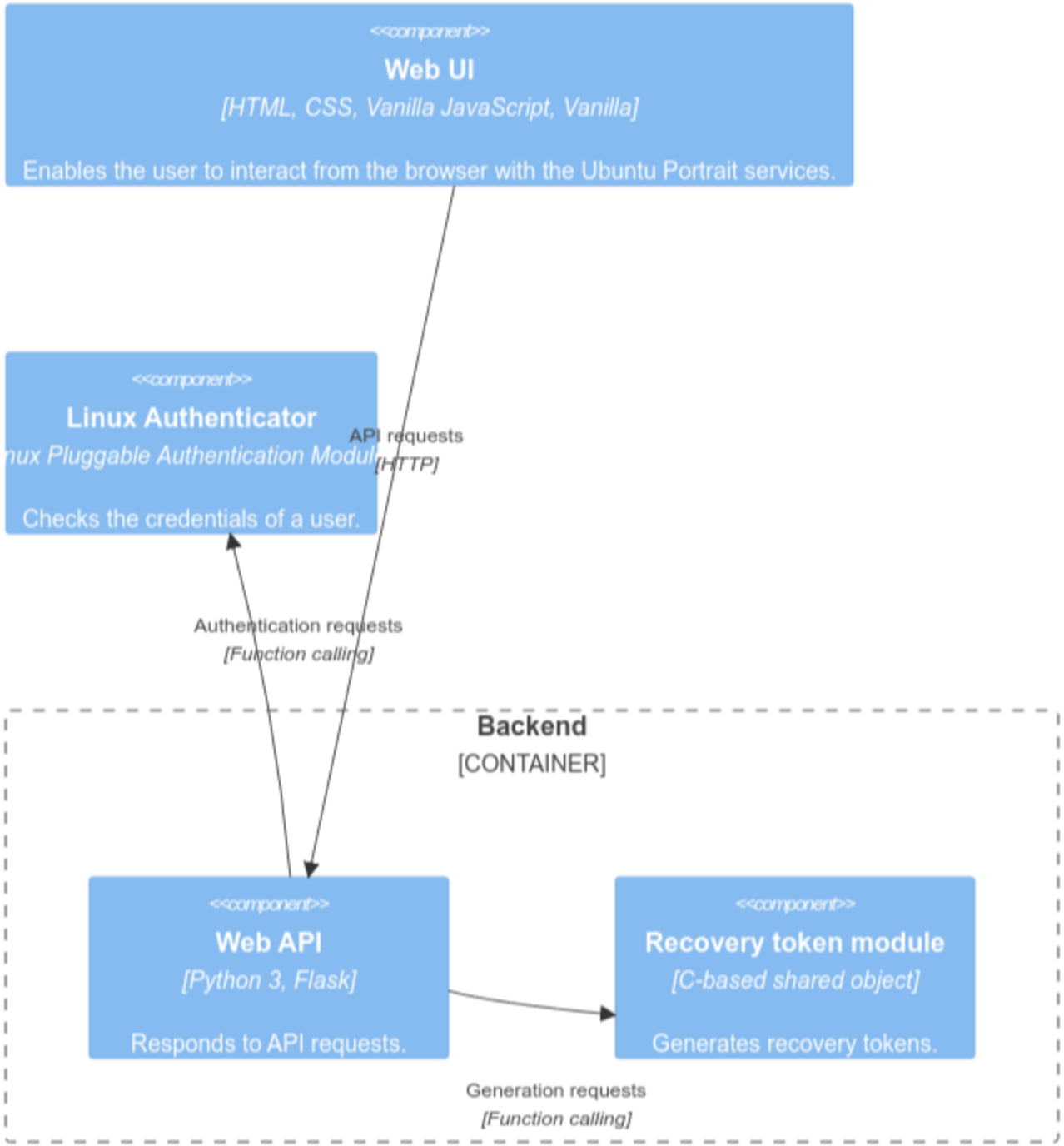
- Lots of OSS tools that can be used to proactively detect vulnerabilities
- Structure
  - Factual information
    - **General software and software security topics**
    - **Brief presentation of each analysis technique**
  - **Practical examples for analysing a vulnerable codebase**
    - Infrastructure and access
    - Documentations
    - Proposed solutions

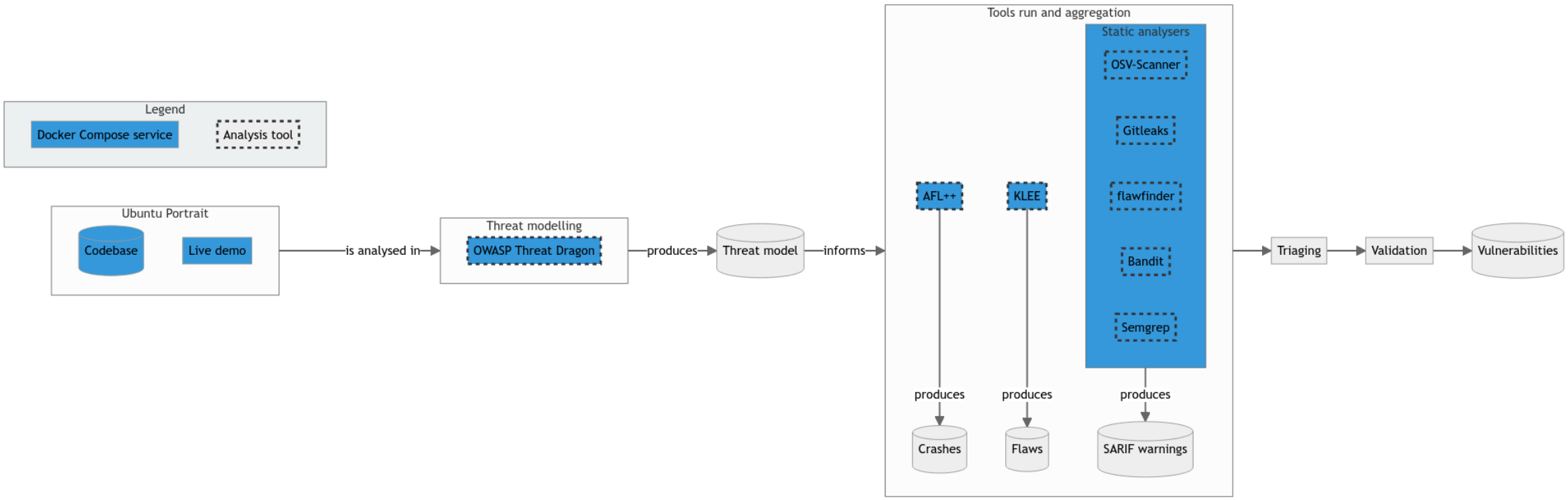

Production environment, managed by the consumer
User configuration
attached to
Third-party dependencies
dynamically linked in
stores
Development environment, managed by the producer
produces
Documentation
Development process
informs
produces
Tests
stored in
Repository
verifies
produces
Configuration
used in
Codebase
Code dependencies
In-house dependencies
vendored in
used in
Build process
creates
Package manager
provides
Third-party dependencies
Software package
used in
Deployment process
creates
Production solution
stores
Related systems
Host operating system
File system
Syscalls
Sockets
GUI
Services
APIs
Databases
takes inputs from
sends data to


Vulnerability detection
Threat modelling
produces
Threat model
informs
describes
Manual vulnerability detection
informs
describes
Automated vulnerability detection
Tools run and aggregation
All SARIF warnings
Triaging
Validation
Vulnerabilities
Analysis types
Depending on running the codebase
Static
Dynamic
Hybrid
Depending on the information about the codebase
Transparent
Partially-transparent
Opaque
Defensive activities
Vulnerability research
CVSS approximation
CWE approximation
CVE request
Patching
Communication with the stakeholders
Offensive activities
Exploit writing
Attack vector
Mitigations bypass
Weaponisation
DREAD
STRIDE

## Ubuntu Portrait

- WebGoat-like codebase
- "*lightweight piece of software that runs on an Ubuntu server and allows users to control it through their browsers*"
- On-premise deployment
- Written in Python and C
- 12+ embedded vulnerabilities

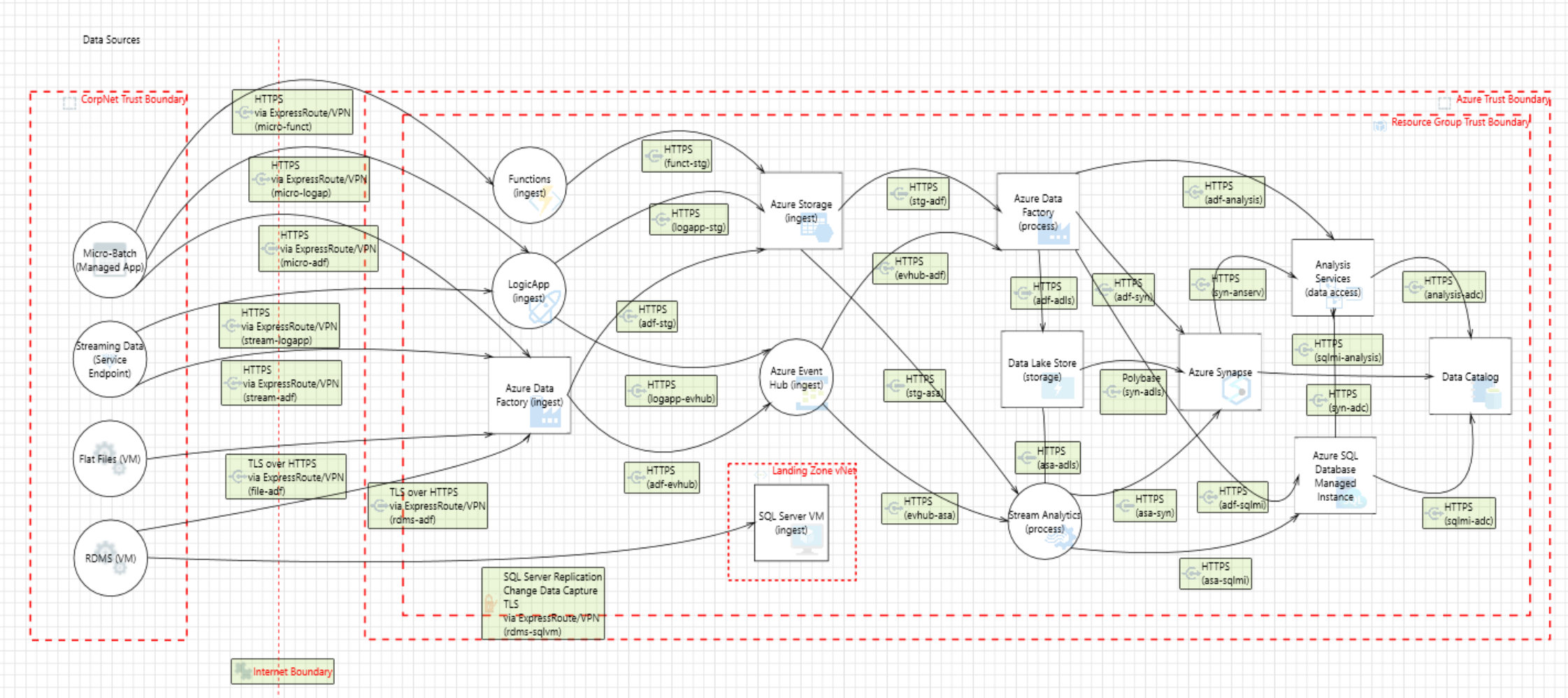

<<component>>
Web UI
[HTML, CSS, Vanilla JavaScript, Vanilla]
Enables the user to interact from the browser with the Ubuntu Portrait services.
<<component>>
Linux Authenticator
nux Pluggable Authentication Modul
Checks the credentials of a user.
API requests
[HTTP]
Authentication requests
[Function calling]
Backend
[CONTAINER]
<<component>>
Web API
[Python 3, Flask]
Responds to API requests.
<<component>>
Recovery token module
[C-based shared object]
Generates recovery tokens.
Generation requests
[Function calling]


Legend
Docker Compose service
Analysis tool
Ubuntu Portrait
Codebase
Live demo
is analysed in
Threat modelling
OWASP Threat Dragon
produces
Threat model
informs
Tools run and aggregation
Static analysers
OSV-Scanner
Gitleaks
flawfinder
Bandit
Semgrep
AFL++
KLEE
produces
produces
produces
Crashes
Flaws
SARIF warnings
Triaging
Validation
Vulnerabilities


Soymilk
ORIGINAL
7g SOY POWERED PROTEIN
CHECK FOR
CAT ARM B4
CLOSING

# Threat modelling

- Identifying asset and threats
  - What we need to defend?
  - What can go wrong?
- Advantages
  - Secure by design
  - Prioritisation
  - Stakeholder confidence booster
  - Legal requirement (e.g., USA and Singapore)




> From `AzureArchitecture/threat-model-templates`

# OWASP Threat Dragon

- Threat modelling tool backed by OWASP
- Usual process
    i. Threat model creation
    ii. Diagram creation: STRIDE, CIA
    iii. Asset representation: stores, process, actor, data flow, trust boundaries
    iv. Manual threat identification, with type, status, score, priority, description, and mitigation

# Demo


"You do realize the key is under the mat."

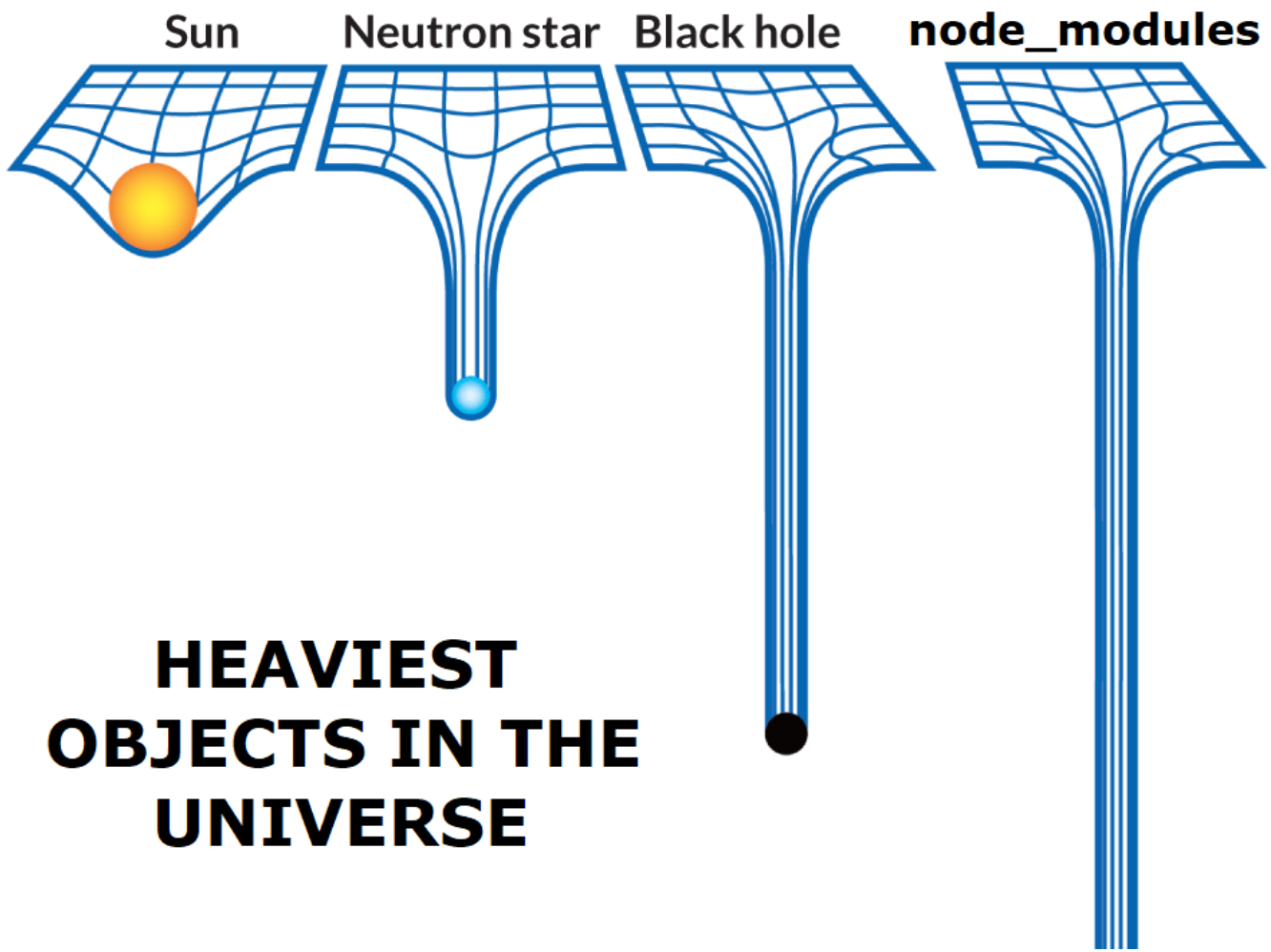
# Secret scanning

- Searching for specific patterns or entropy for a secret
- Secrets
  - API keys
  - Credentials
  - Tokens
- Community (generic) rules

## Gitleaks

- Detector for hardcoded secrets
- Analysis of the entire Git history
- Support for baselines and custom formats of secrets

# Demo


Sun
Neutron star
Black hole
node_modules
HEAVIEST OBJECTS IN THE UNIVERSE

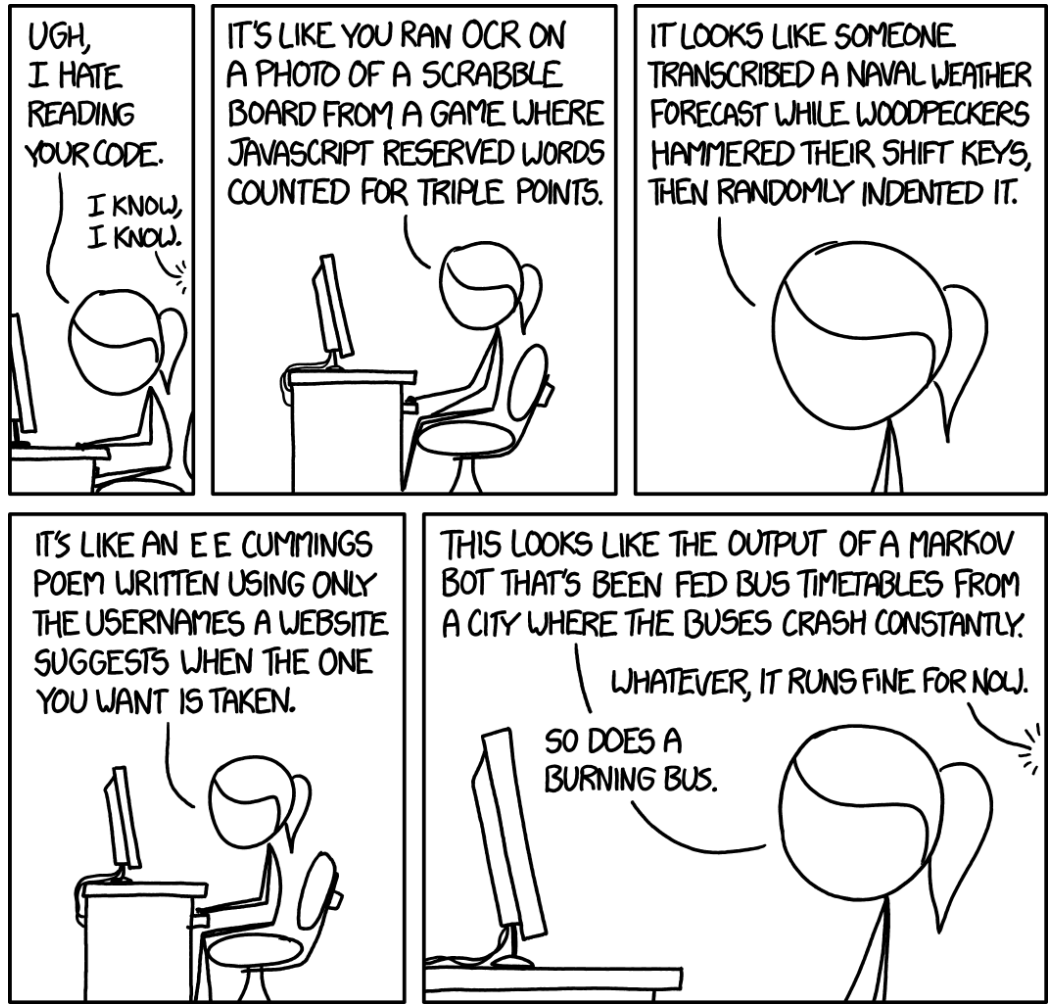
## Dependency scanning

- Iterating through all dependencies for finding their vulnerabilities
- Usage of the dependencies declaration list

## OSV-Scanner

- Client for Google's OSV database, which embeds:
  - GitHub Security Advisories
  - PyPA
  - RustSec
  - Global Security Database
- Support for ignored vulnerabilities

# Demo


UGH, I HATE READING YOUR CODE.
I KNOW, I KNOW.
IT'S LIKE YOU RAN OCR ON A PHOTO OF A SCRABBLE BOARD FROM A GAME WHERE JAVASCRIPT RESERVED WORDS COUNTED FOR TRIPLE POINTS.
IT LOOKS LIKE SOMEONE TRANSCRIBED A NAVAL WEATHER FORECAST WHILE WOODPECKERS HAMMERED THEIR SHIFT KEYS, THEN RANDOMLY INDENTED IT.
IT'S LIKE AN E E CUMMINGS POEM WRITTEN USING ONLY THE USERNAMES A WEBSITE SUGGESTS WHEN THE ONE YOU WANT IS TAKEN.
THIS LOOKS LIKE THE OUTPUT OF A MARKOV BOT THAT'S BEEN FED BUS TIMETABLES FROM A CITY WHERE THE BUSES CRASH CONSTANTLY.
WHATEVER, IT RUNS FINE FOR NOW.
SO DOES A BURNING BUS.

# Linting

- Static analysis tool for finding issues before compiling/running the code
- Issues
  - Formatting
  - Grammar (for example, non-inclusive expressions)
  - Security

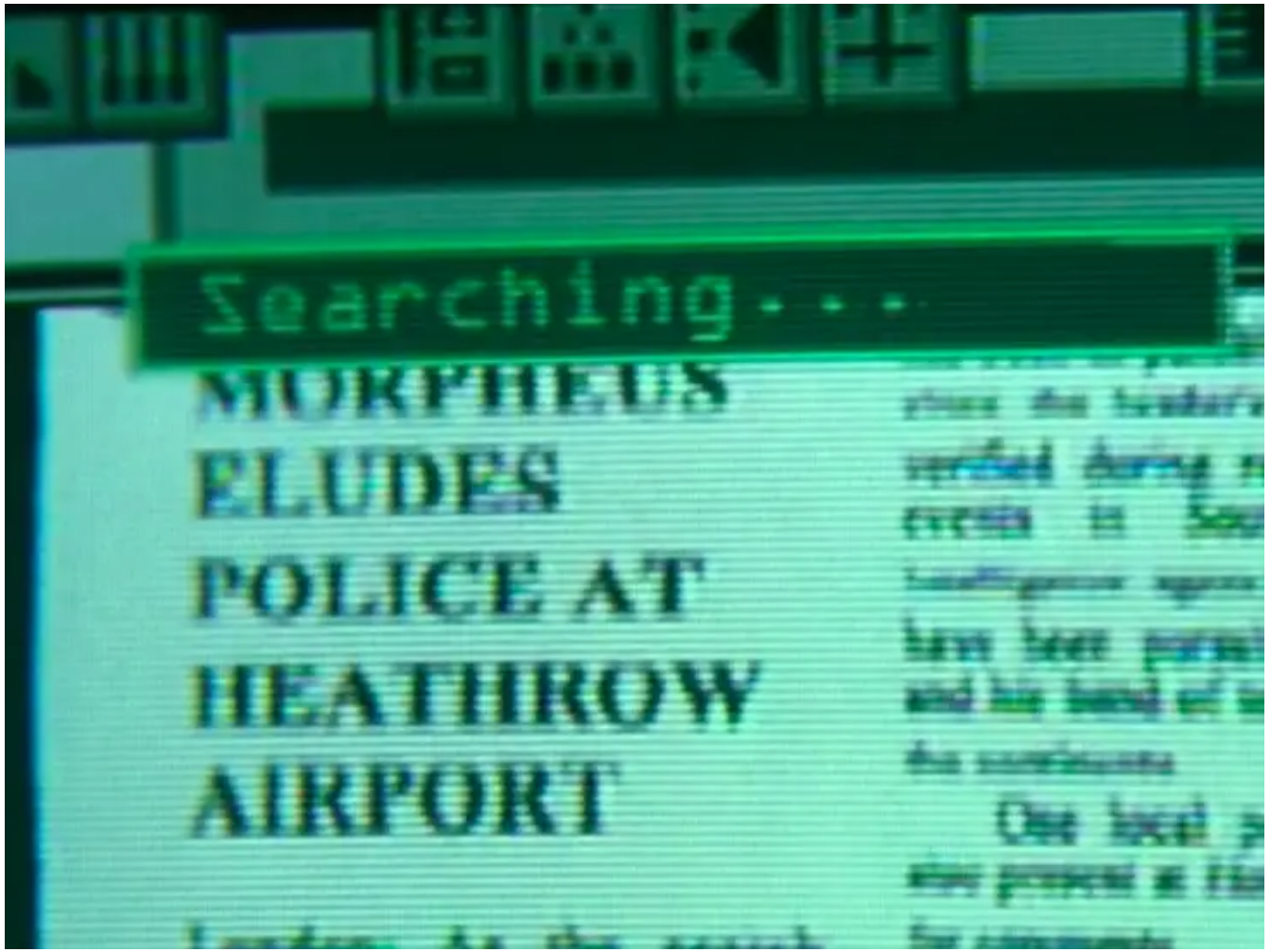
# Bandit

- Linter for Python
- Abstract syntax tree representation of the code
- Custom modules for:
  - Patterns of suspicious code
  - Deny lists of imports and function calls
  - Report generation
- Support for baselines

## flawfinder

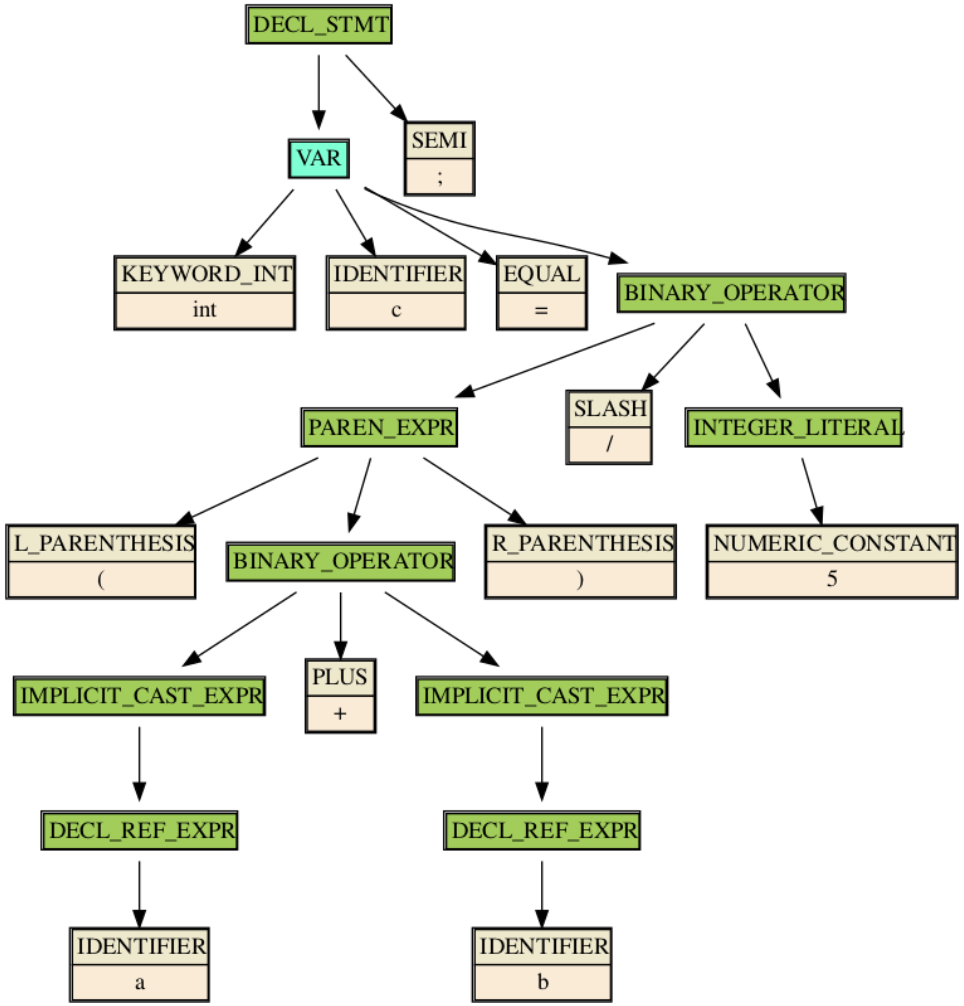
- Linter for C
- Lexical scanning with detection of sensitive tokens

# Demo


Searching...
MORPHEUS
ELUDES
POLICE AT
HEATHROW
AIRPORT

# Code querying

- Searchin a specific pattern in the codebase
- Optional abstract representation of the codebase
  - Abstract syntax trees
  - Control flow graphs
- Query types
  - Lexical
  - Regex
  - Data structures specific to the abstract representation
- Community queries (but generic)




> From Trail of Bit's "Fast and accurate syntax searching for C and C++"

## Semgrep

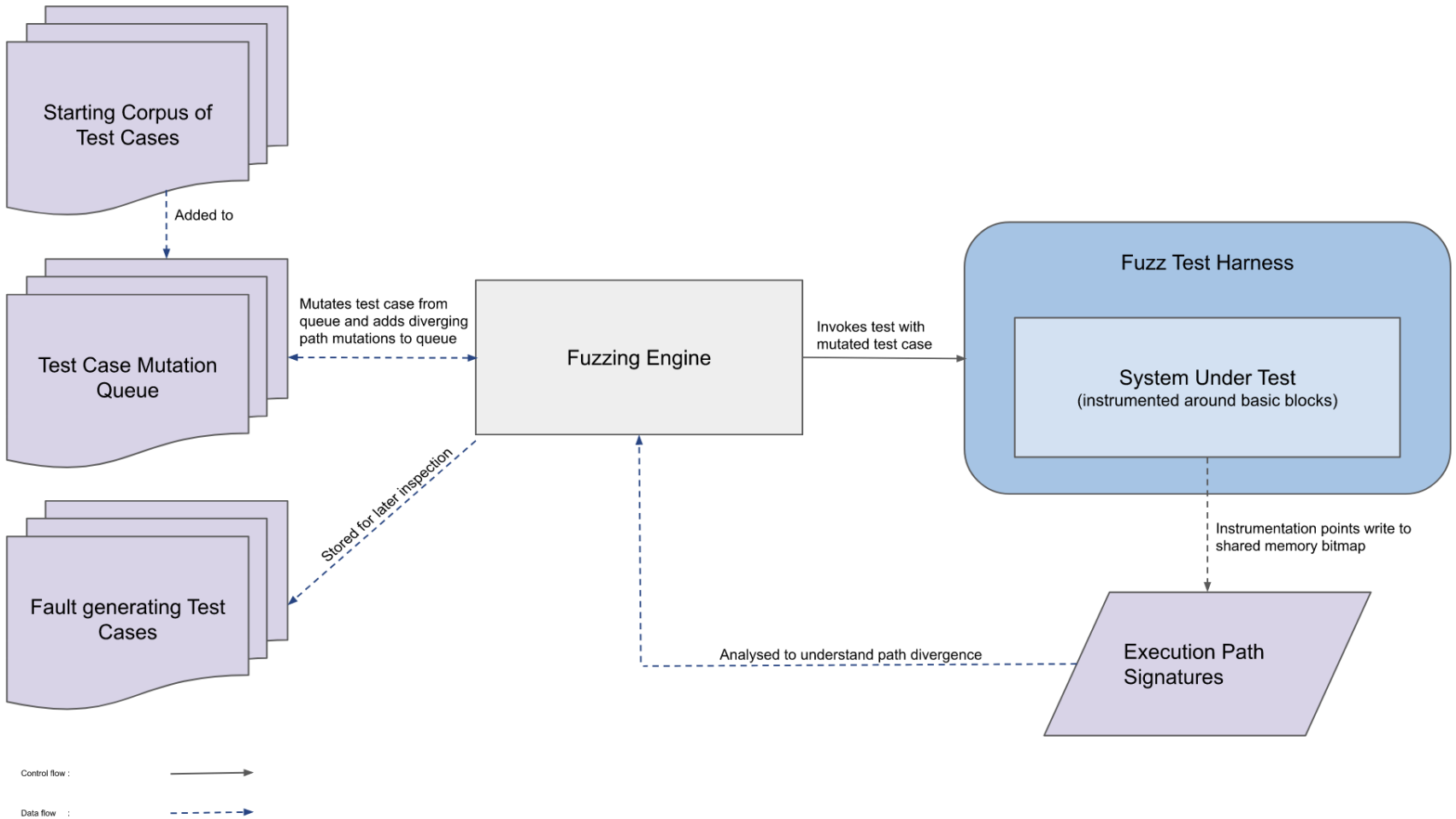
- (Partially) open-source code scanner
- Support for 30+ programming languages
- No prior build requirements
- No DSL for rules
- Default or third-party rules

# Demo


18 ONLY
FREE

# Fuzzing

- Running a program and offering random, unexpected inputs
- A crash = a security issue
- BFS traversal of the CFG
- Optimisation
  - Instrumenting the source code
  - Knowing the input format
  - Defining the states
  - Testing all input streams




> From AdaCore's "Finding Vulnerabilities using Advanced Fuzz testing and AFLplusplus v3.0"

# AFL++

- An American Fuzzy Lop (AFL) fork
- Additional features compared to AFL
  - QEMU emulation
  - Persistent mode
  - Optimisations
- Embedded in Google's OSS-Fuzz

# Demo

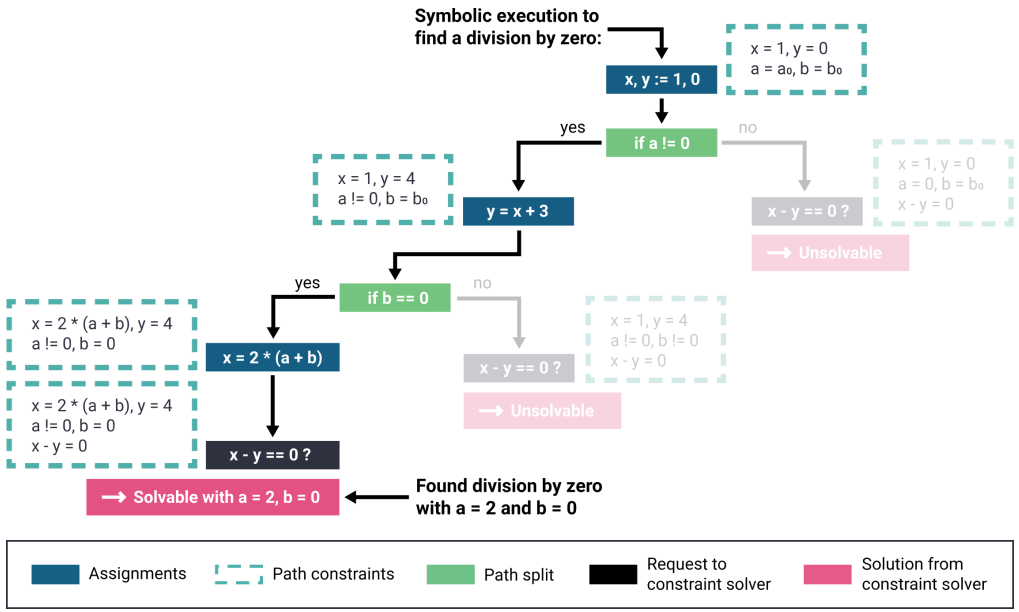
## Symbolic execution

- Investigating all CFG paths by replacing the concrete values with symbolic ones
- Components
  - Sources
  - Sinks
  - Patterns
- Path explosion problem

```
int f(int a, int b){
    int x = 1, y = 0;

    if (a != 0) {
        y = x + 3;
        if b == 0 {
            x = 2 * (a + b);
        }
    }

    return (a + b) / (x - y);
}
```




> From symflower's "What is symbolic execution for software programs"

# KLEE

- Generic symbolic execution with security use cases
- Built on LLVM

# Demo

## Other techniques

- Stress/load testing
  - JMeter for many protocols and services
  - `k6` for Kubernetes
- Web dynamic analysis
  - OWASP's Zed Proxy Attack


Programmer
Task that takes
5 minutes
Can it be automated?

## Security tooling automation

- SARIF Multitool for performing operations with SARIF files (merging, paging, querying, supressing, etc.)
- Make and Poe the Poet for running tasks
- IDE workflows (e.g., VSCode tasks) for running the tooling while coding
- `pre-commit` for managing Git pre-commit hooks
- `act` or GitLab Runner for running CI/CD workflows locally
- GitHub Actions or GitLab pipelines for running CI/CD workflows


CHECK AFTER ENGINE START
1 Oil pressure CHECKED 1 pf
2 RPM 740 +/- 30 CHECKED 2 pf
3 Warm up time START 3 pf
Warm up:
Idle.....2 minutes
50% load.....until Oil >50° and Coolant > 60°C
4 Fuel pumps (2) check OFF 4 pf
5 Fuel selectors (2) X-FEED 5 pf
6 Pitot heat ON, annunciation + Amps checked 6 pf
7 Pitot heat OFF 7 pf
8 PFD/MFD NORMAL MODE 8 pf
9 Avionics master ON 9 pf
FMS SETUP
Initialize profile (AUX 4, MAP)
Flight plan
Radios (COM, NAV, ADF, DME, CDI, BRG ½)
Performance (speed bugs; Flight ID if applicable)
10 FMS setup COMPLETED 10 pf
AUTOPILOT TEST
DISCONN press, check electric trim not working
AP ON, check annunciations and FD
DISCONN press, check AP off
GA button press, check FD commands climb
11 Autopilot test CHECKED, ON as required 11 pf
12 Flood light COMPLETED 12 pf
13 lights ON as 13 pf
ectors (2)
s (2)
y horizon CODE / MODE CHEC
onder RELEAS
brake
Diamond Aircraft Flight Training Division
Does not replace
COOLANT TEMP
FUEL TEMP
FUEL QTY GAL
ENGINE
MAP
EHLE
EHLEB
FLEVO
RANGE
PAN
FMS
RH MAIN BUS
AVIONICS BUS
FUEL PUMPS LH ENGINE
FUEL PUMPS RH ENGINE
FUEL PUMP A
FUEL PUMP B
max. usable fuel
main tank:
2 x 25 US gal
auxiliary tank:
2 x 13 US gal
rudder trim
AP DISC
PTT
PARKING BRAKE
CABIN

# Security checklist I: Proactive vulnerability discovery

☑️ Create a threat model.

☑️ Choose a suite of security tools to scan your codebase.

☑️ Automate the suite of security tools in local/development environments and CI/CD pipelines, with quality gates.

☑️ Request the integration of your project with OSS-Fuzz.

🔁 Periodically check for vulnerabilities in your dependencies.

🔁 Constantly validate the warnings from your security tooling.

🔁 Keep the threat model updated.

> One-time activities are marked with ☑️, and the recurrent ones with 🔁.

# Security checklist II: Secure users

- ☑️ Design your software to be secure by default.
- ☑️ Have security recommendations for users.
- ☑️ Create SBOMs.

> One-time activities are marked with ☑️, and the recurrent ones with 🔁.

# Security checklist III: Established security reporting process

☑️ Have a standardised, documented process for responding to vulnerabilities.
☑️ Create a security policy with preferred way to contact and report format.
☑️ Find backup security responders.
🔁 Be transparent and verbose with the reported vulnerabilities: mention patching commits, attach security tags to issues, and request CVE IDs.

> One-time activities are marked with ☑️, and the recurrent ones with 🔁.


That's all Folks!