MUNI
CSIRT-MU

SAST Essentials

Matěj Smyčka

whoami

- Penetration tester at CSIRT Masaryk University
- I wrote thesis on SAST tools and few guides how to use them.
- I like to program things :)

Goal of this presentation

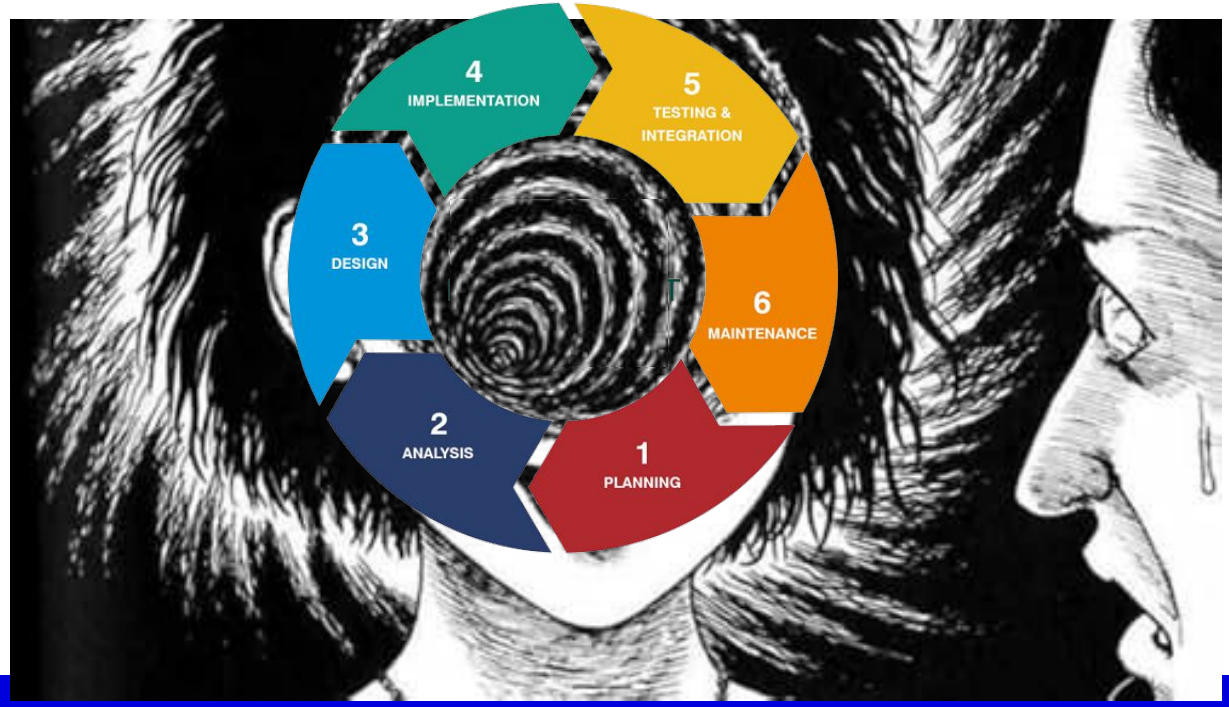
- Introduce SAST
- Show different categories of SAST tools
- Present how are we using SAST
- Share our experience

What is SAST?

- Static Application Security Testing (SAST)
- Source code approach
- Source code meaning any text
  - Config files
  - Dependency files (*requirements.txt, packages.json,* …)
  - IaC code (*Terraform, Kubernetes, Dockerfiles,* …)
  - Source code
- In theory: symbolic execution, bounded model checking, taint analysis, …
- In reality (mostly): **regex**
- **Typically for AppSec**

Our problem

1. We catch vulnerabilities too late in **SDLC**
2. “Reading” code in whitebox pentesting is repetitive
3. Need to improve AppSec of our own tooling



Solution - SAST

We catch vulnerabilities late in SDLC

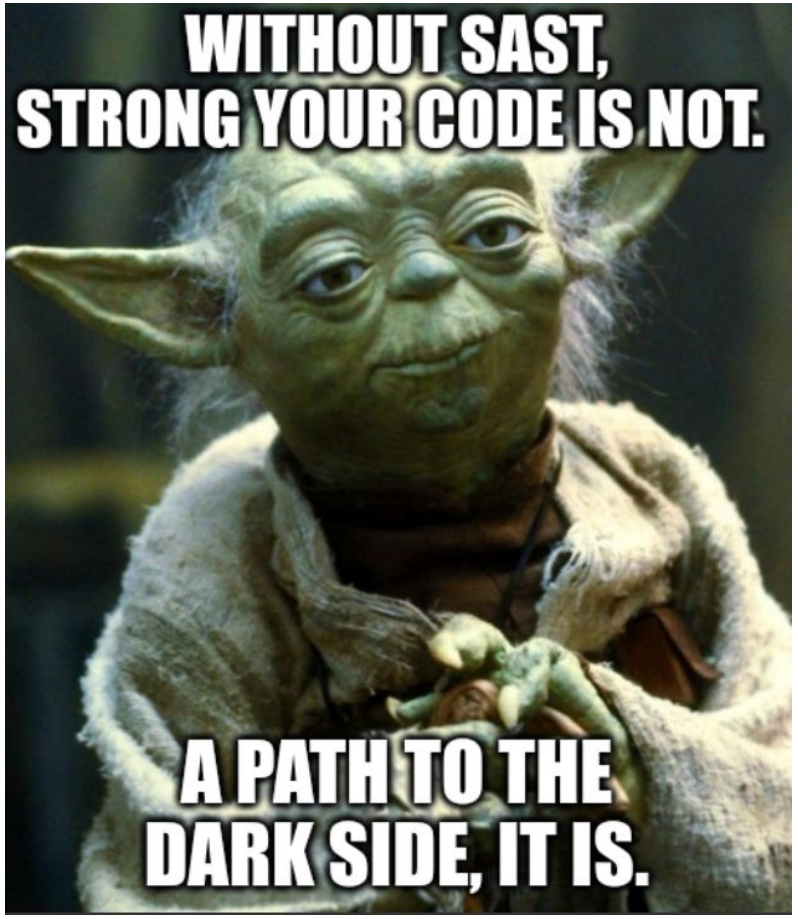
- “Shift-left” approach
- SAST integration into CI/CD pipelines
- Recommending SAST scans before penetration test

Automate “reading” code in whitebox pentesting

- Running SAST scans so we know what to look for
- Speeding up pentest

Need to improve AppSec of our own toolin

- Using SAST so we know what we are recommending
- At Least some baseline for security



SAST categories and tooling

- Vulnerability detection
- Secret detection
- Dependency scanning
- IaC SAST

Vulnerability detection

- Catching command injection, SQLi, unsecure crypto, …
- Regex + abstract syntax tree (AST)
- Tools like *Semgrep, Bandit, SonarQube* …

```
from bandit.core import utils

SIMPLE_SQL_RE = re.compile(
    r"(select\s.*from\s|"
    r"delete\s+from\s|"
    r"insert\s+into\s.*values\s|"
    r"update\s.*set\s)",
    re.IGNORECASE | re.DOTALL,
)
```

```
pattern-either:
- pattern: |
    $X = flask.request.args.get(...)
    ...
    flask.make_response("...".format($X))
- pattern: |
    $X = flask.request.args.get(...)
    ...
    flask.make_response(f"...{$X}...")
- pattern: |
    $X = flask.request.args.get(...)
    ...
```

Secret detection

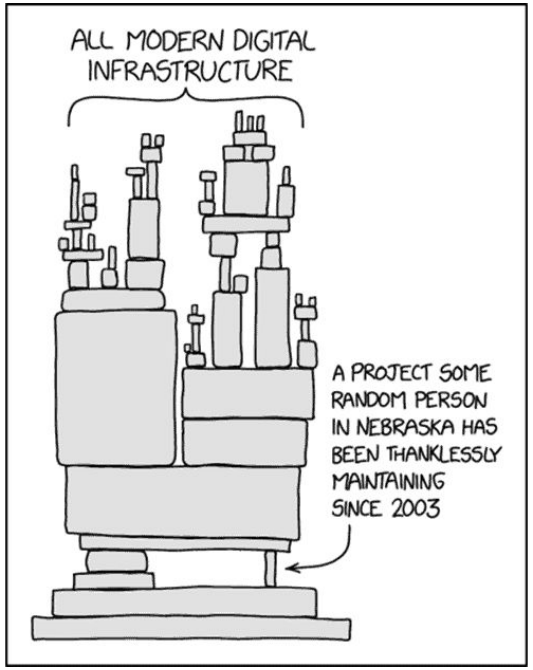
- Looking for secrets (API keys, tokens, ssh keys) in text
- Tools like *Gitleaks, Trufflehog* or just *grep*.
- FPs vs FNs
- *Git* commit history
- Logs

```
^.*secret\s*(\=|\:)+.*$
^.*password\s*(\=|\:)+.*$
^.*token\s*(\=|\:)+.*$
^.*heslo\s*(\=|\:)+.*$
^.*login\s*(\=|\:)+.*$
^.*dbname\s*(\=|\:)+.*$
^.*pass\s*(\=|\:)+.*$
https?:\/\/.*:.*@gitlab.ics.muni.cz
```

```
matej@debian ~/D/t/d/secrets (master) [1]> gitleaks git -v

    ○
    │╲
    │ ○
    ○ ▯
      ▯    gitleaks

Finding:     ...l_project_id="123", gl_token="xD2j_Am_CozdGUysp_DA"
Secret:      xD2j_Am_CozdGUysp_DA
RuleID:      generic-api-key
Entropy:     3.884184
File:        create_issue.py
Line:        3
Commit:      37f6dc9aa468ebd5da0928e0ad7888f25b0a7e9a
Author:      Matej
Email:       smycka@ics.muni.cz
Date:        2024-11-11T10:29:30Z
Fingerprint: 37f6dc9aa468ebd5da0928e0ad7888f25b0a7e9a:create_issue.py:generic-api-key:3

11:29AM INF 1 commits scanned.
11:29AM INF scan completed in 10ms
11:29AM WRN leaks found: 1
```

Dependency scanning

- Dependency files (*requirements.txt, packages.json, Dockerfile …*)
- Parse dependency files -> Compare versions to vulnerability DB -> yield result
- Tools like *Grype*, *Renovate*, *Dependabot, nmp audit, DependencyCheck …*
- Autofix features
- *SBOM*



body-parser	1.18.2	1.20.3	npm	GHSA-qwcr-r2fm-qrc7	High
cookie	0.3.1	0.7.0	npm	GHSA-pxg6-pf52-xh8x	Low
dicer	0.2.5		npm	GHSA-wm7h-9275-46v2	High
ejs	3.1.9	3.1.10	npm	GHSA-ghr5-ch3p-vcr6	Medium
express	4.16.0	4.19.2	npm	GHSA-rv95-896h-c2vc	Medium
express	4.16.0	4.20.0	npm	GHSA-qw6h-vgh9-j6wx	Medium
jinja2	3.1.2	3.1.4	python	GHSA-h75v-3vvj-5mfj	Medium
jinja2	3.1.2	3.1.3	python	GHSA-h5c8-rqwp-cp95	Medium
jpeg-js	0.2.0	0.4.4	npm	GHSA-xvf7-4v9q-58w6	High
jpeg-js	0.2.0	0.4.0	npm	GHSA-w7q9-p3jq-fmhm	Medium
minimist	0.0.8	0.2.4	npm	GHSA-xvch-5gv4-984h	Critical
minimist	0.0.8	0.2.1	npm	GHSA-vh95-rmgr-6w4m	Medium
path-to-regexp	0.1.7	0.1.10	npm	GHSA-9wv6-86v2-598j	High
phin	2.9.3	3.7.1	npm	GHSA-x565-32qp-m3vf	Medium
pillow	3.1.0	6.2.2	python	GHSA-vcqg-3p29-xw73	Critical

Infrastructure as a Code (IaC) SAST

- Looking for misconfigurations in:
  - *Terraform (Terrascan, Checkov)*
  - *Dockerfile (Hadolint, Grype)*
  - *Kubernetes/Helm Charts (Terrascan, Checkov)*
- Not as useful for pentests
- Good for checking best practises

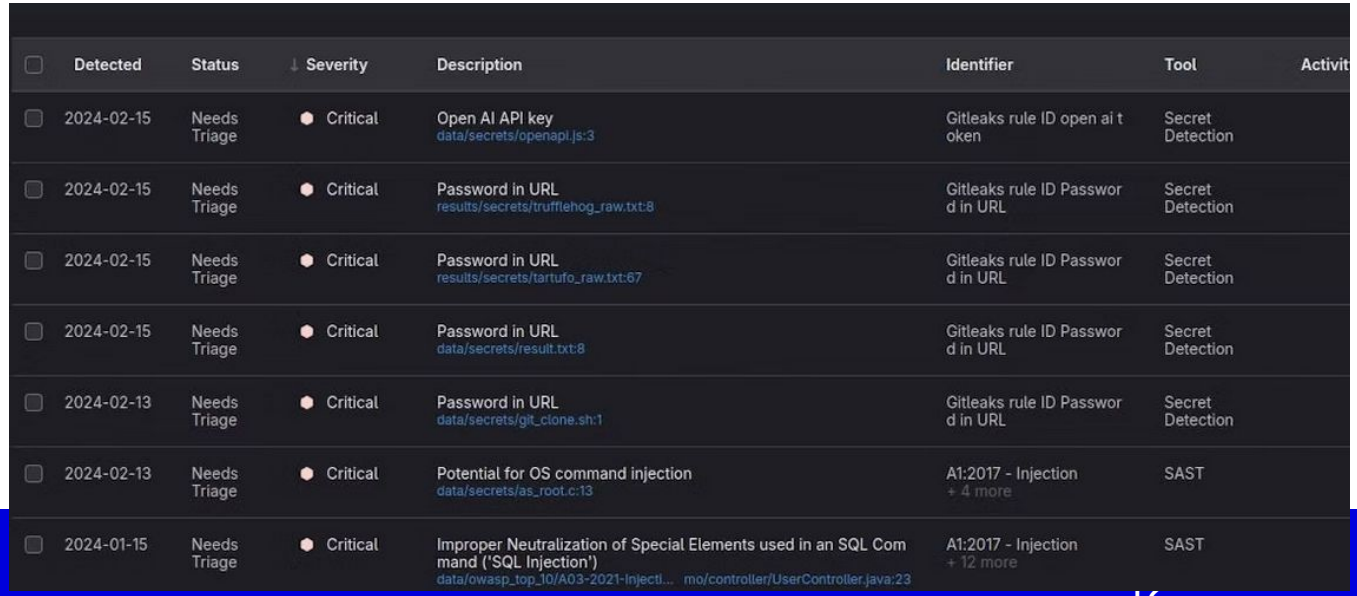
```
Description     :    Memory Limits Not Set in config file.
File            :    charts/app-deps/ingress-controller/dns/dns-debug.yaml
Line            :    1
Severity        :    MEDIUM
```

```
Status: Downloaded newer image for hadolint/hadolint:latest
-:12 DL3002 warning: Last USER should not be root
-:15 DL3027 warning: Do not use apt as it is meant to be a end-user tool, use apt-get or apt-cache instead
-:22 DL3047 info: Avoid use of wget without progress bar. Use `wget --progress=dot:giga <url>`. Or consider
-:23 DL3059 info: Multiple consecutive `RUN` instructions. Consider consolidation.
-:24 DL3003 warning: Use WORKDIR to switch to a directory
-:31 DL3059 info: Multiple consecutive `RUN` instructions. Consider consolidation.
-:32 DL3059 info: Multiple consecutive `RUN` instructions. Consider consolidation.
-:33 DL3059 info: Multiple consecutive `RUN` instructions. Consider consolidation.
```

SAST for development

- **Deployment:**
  - Integration with pipelines
  - pre-commit hooks
  - Manual checks
- **Useful tips:**
  - Select tools based on where you host your code (*Github, Gitlab, Bitbucket* ..)
  - Setup auto-fix on dependency scanning (*Renovate*, *Dependabot*)
  - Do not use *GitLab* built-in SAST only
  - Use Security dashboard on *Gitlab*.

```
semgrep:
  image: semgrep/semgrep
  variables:
    SEMGREP_GITLAB_JSON: "1"
  script:
    - semgrep scan . --config="r/all" --metrics="off" --error --gitlab-sast -o gl-sast-report.json
  artifacts:
    reports:
      sast: gl-sast-report.json
```



Detected	Status	Severity	Description	Identifier	Tool	Activit
2024-02-15	Needs Triage	Critical	Open AI API key data/secrets/openapi.js:3	Gitleaks rule ID open ai t oken	Secret Detection	
2024-02-15	Needs Triage	Critical	Password in URL results/secrets/trufflehog_raw.txt:8	Gitleaks rule ID Passwor d in URL	Secret Detection	
2024-02-15	Needs Triage	Critical	Password in URL results/secrets/tartufo_raw.txt:67	Gitleaks rule ID Passwor d in URL	Secret Detection	
2024-02-15	Needs Triage	Critical	Password in URL data/secrets/result.txt:8	Gitleaks rule ID Passwor d in URL	Secret Detection	
2024-02-13	Needs Triage	Critical	Password in URL data/secrets/git_clone.sh:1	Gitleaks rule ID Passwor d in URL	Secret Detection	
2024-02-13	Needs Triage	Critical	Potential for OS command injection data/secrets/as_root.c:13	A1:2017 - Injection + 4 more	SAST	
2024-01-15	Needs Triage	Critical	Improper Neutralization of Special Elements used in an SQL Com mand ('SQL Injection') data/owasp_top_10/A03-2021-injecti... mo/controller/UserController.java:23	A1:2017 - Injection + 12 more	SAST	

SAST for penetration testers

- Catch low hanging fruits
- Create your own rules
- High FPs, Lower FNs than in development
- Scan manually before each pentest
- Automate repetitive tasks
  - Like *Ansible* to get *APT* packages and their *CVE*s

```
patterns:
  - pattern-either:
      - patterns:
          - pattern: |
              ...
              $QUERY = "..." . $INPUT . "...";
              ...
          - pattern: $SQLFUNC(..., $QUERY, ...);
      - pattern: |
          $SQLFUNC(...,"..." . $INPUT . "...",...);

  - metavariable-pattern:
      metavariable: $SQLFUNC
      patterns:
        - pattern-either:
            - pattern: query
            - pattern: mysql_query
            - pattern: pg_query
            - pattern: pg_send_query
```

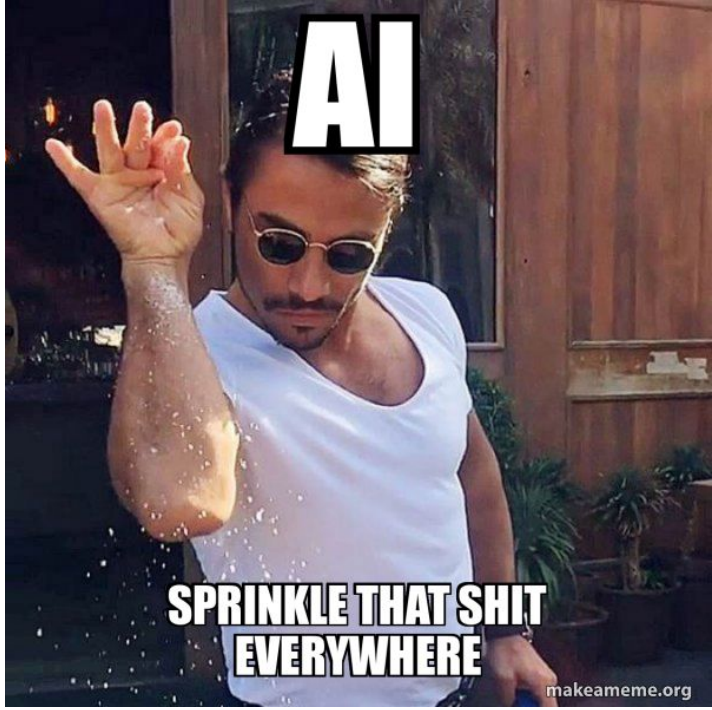
```
$pass = $_POST['password'];

$sql = "SELECT * FROM users WHERE username = '" . $user . "' AND password = '" . $pass . "'";

$result = $conn->query($sql);
```

SAST limitations

- SAST won’t solve everything, no silver bullet :(
- Typically can’t see access control vulns like IDOR.
- Higher overhead (Dealing with False Positives).

SAST future

- *LLMs*
  - Detecting vulnerabilities
  - Writing templates based on our input
- Each *git* hosting service is doing “their” own solution
  - *GitHub* Advanced Security (GHAS)
  - *Gitlab* SAST



Thank you for the attention

Any questions?

Contact me on LinkedIn - Matěj Smyčka

We also do Hacker MEETUPs

Contact: tomci@ics.muni.cz