Continuous Delivery **Vs.** **Security Assurance**

By : Abdessamad TEMMAR

About me

- Abdessamad TEMMAR
- Application Security Engineer (Siris Advisory)
- Ex – full time Pentester
- OWASP Contributor

What this talk is all about ?

DevOps Teams

Appsec team

Automated attacks

Appsec Toolbelt

- SCA : Dependencies Analysis
- DAST (Dynamic) : Scans interface of running application
- SAST (Static) : Scans source code / binaries
- IAST (Interactive) : Detection at runtime

What is static analysis ?

Source/Byte Code

Analysis

Findings

Scanning rules (SQLi, XSS, etc)

Why static analysis ?

- Easy to automate/integarte
- Run anytime
- Fast
- Points directly to suspect code

Which SAST tool ?

Hunting Bugs To Extinction with Static Analysis

The problem :

- An unsecure usage of an external API : download_file

```
download_file(file_id = id, search_type = "file_id")
```

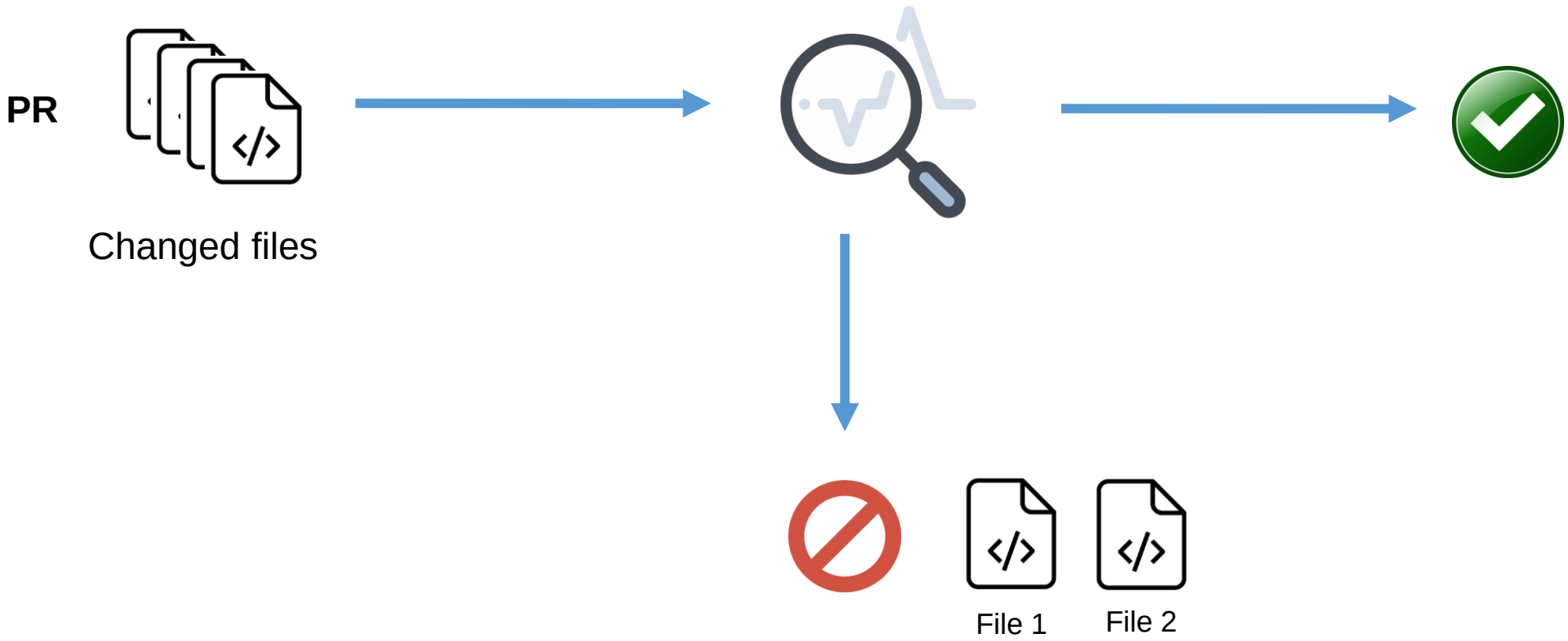
```
download_file(file_id = id, search_type = "path")
```

- We want to identify the same issue for the hole application, and avoid it for future changes.


CATCH
ALL THE BUGS

Hunting Bugs To Extinction with Static Analysis

The solution :



Success criteria

Accuracy

Speed

Easy

Hunting Bugs To Extinction with Static Analysis

Option 1 : grep for the rescue

```
$ more scan_changed_files.sh
git checkout $@ --quiet
files=$(git diff --name-status master HEAD | grep -E "^(A|M)" | cut -f 2)
grep "download_file(.*, search_type = "path")" $files
```

```
$ git diff --name-status
A README.md
M lib/blah.py
D test/new_stuff.yml
```

Hunting Bugs To Extinction with Static Analysis

Option 1 : grep for the rescue

- False Positives

```
# Remember not to use download_file(file_id, search_type = "path") !
```

```
...
def download_file(file_id, search_type = "path"):
...
```

Hunting Bugs To Extinction with Static Analysis

Option 1 : grep for the rescue

- Pro :
  - Easy to use
  - Easy to understand
  - Intercative
- Cons :
  - Line-oriented
  - Mismatch with program structure (trees, ASTs)

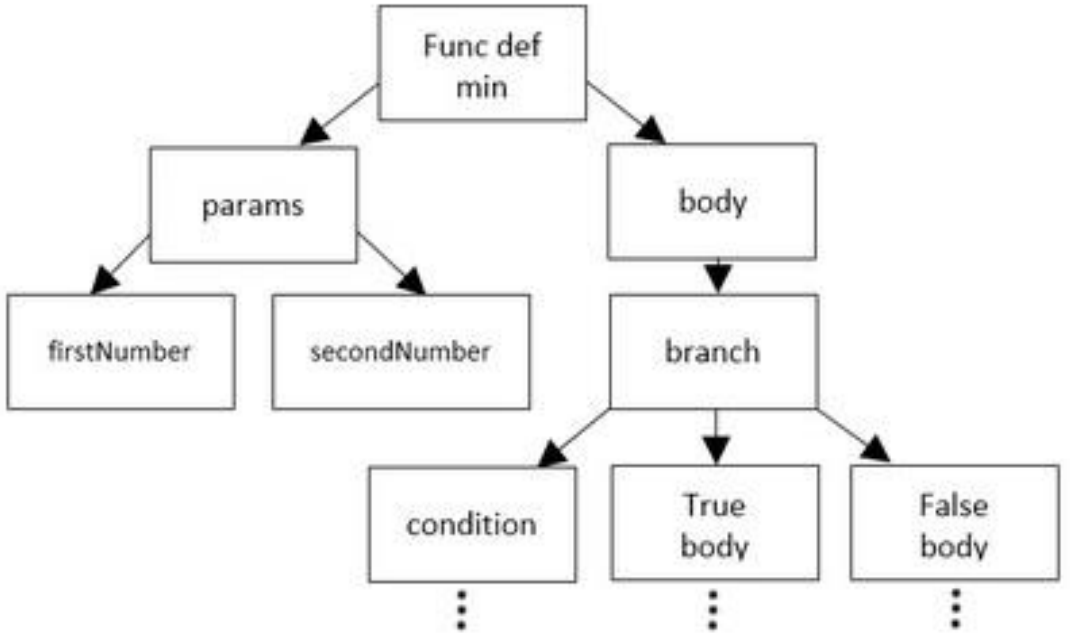
Hunting Bugs To Extinction with Static Analysis

String

```
int min(int firstNumber, int secondNumber)
{
    if (firstNumber > secondNumber) {
        return secondNumber;
    }
    else {
        return firstNumber;
    }
}
```

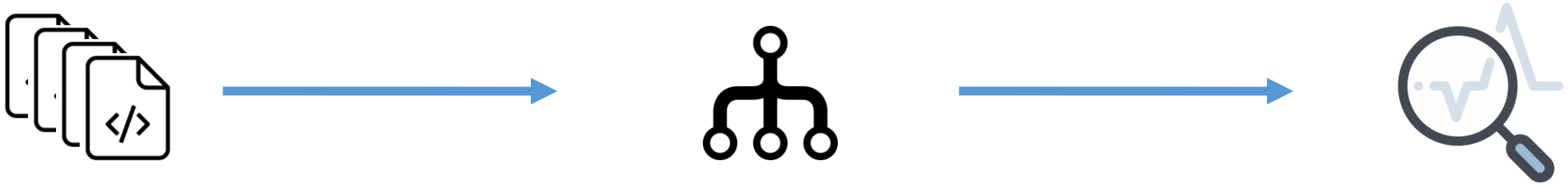
≠

Tree



Hunting Bugs To Extinction with Static Analysis

Option 2 : AST



Source/Byte Code

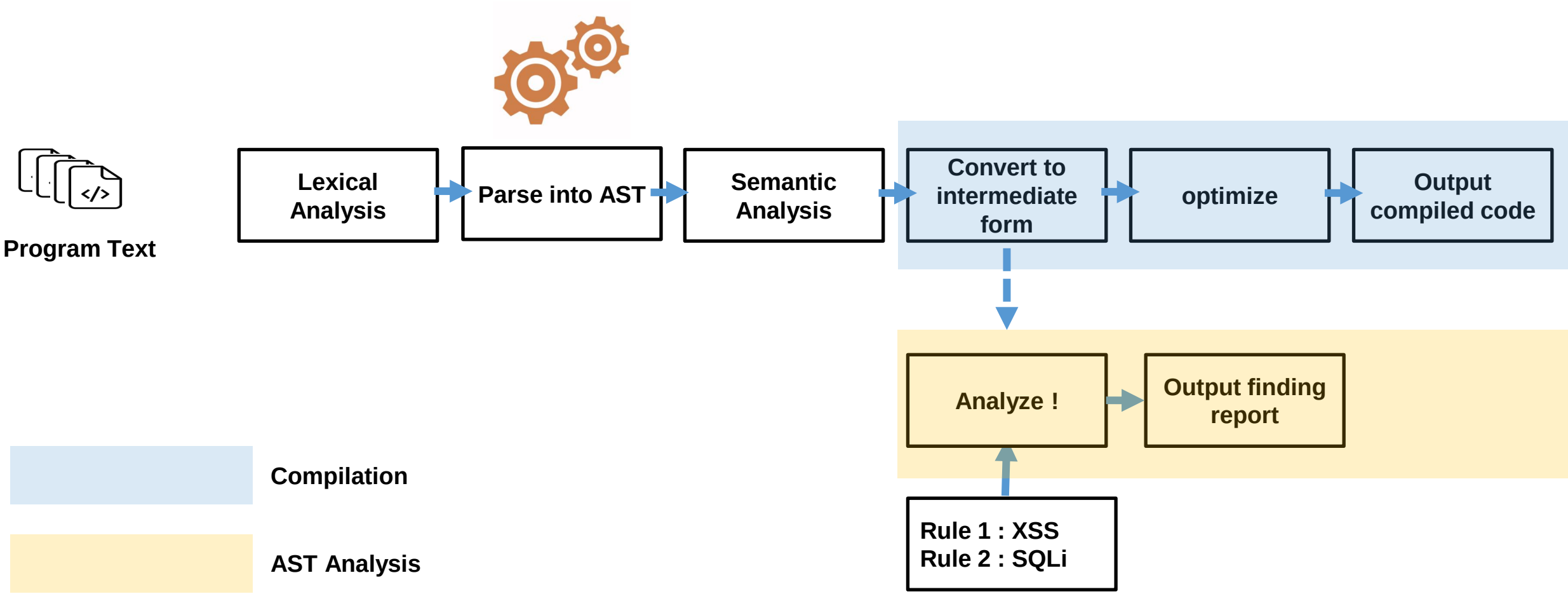
Intermediate reprensentations

Abstract Syntax Tree

Analysis

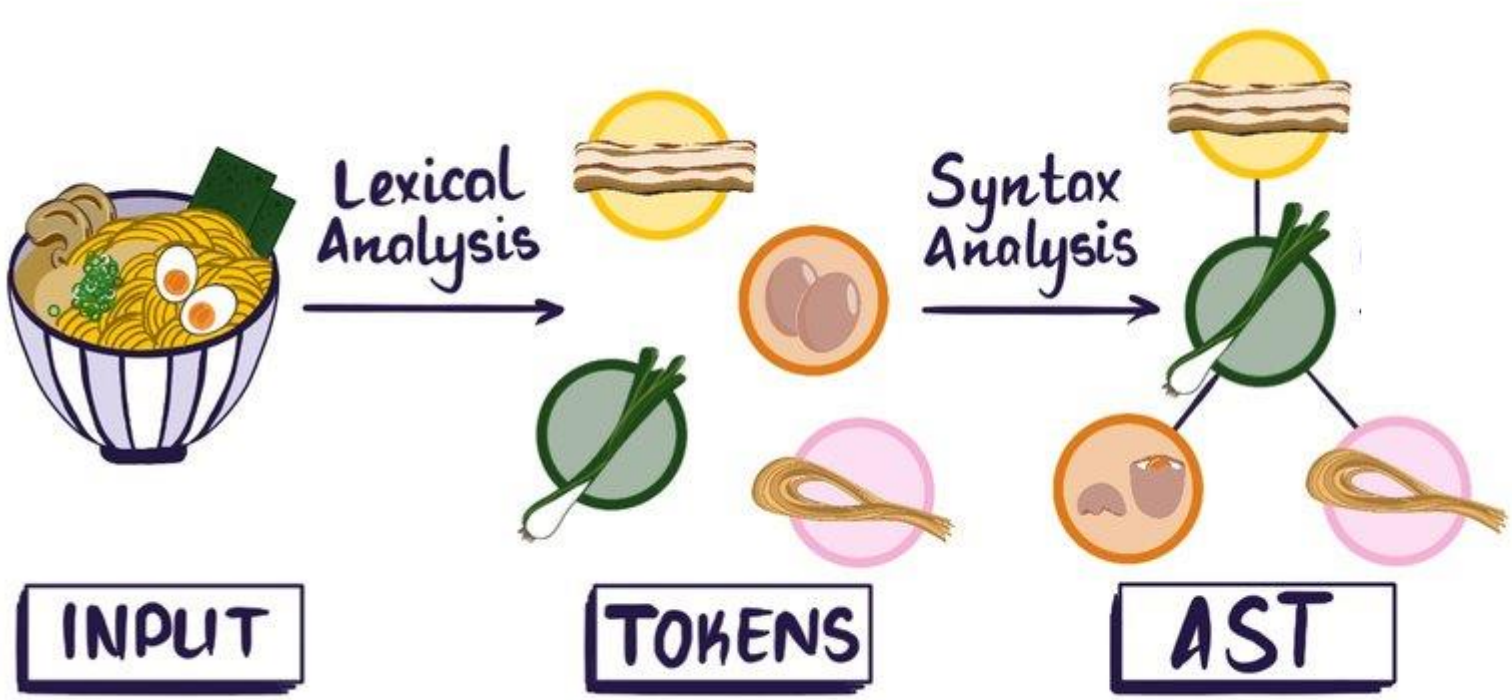
Hunting Bugs To Extinction with Static Analysis

Option 2 : AST based



Hunting Bugs To Extinction with Static Analysis

Option 2 : AST



Hunting Bugs To Extinction with Static Analysis

Option 2 : bandit module

```
import bandit
from bandit.core import test_properties as test

@test.checks('Call')
@test.test_id('B001')

def unsafe_get_userinfo(context):

    if (context.call_function_name_qual == 'download_file' and context.call_args[1] == 'path'):

        return bandit.Issue(
            severity=bandit.HIGH,
            confidence=bandit.HIGH,
            text="Unsafe usage of download_file."
            )
```

Hunting Bugs To Extinction with Static Analysis

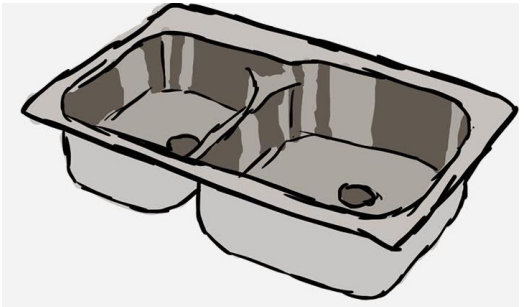
Option 2 : AST

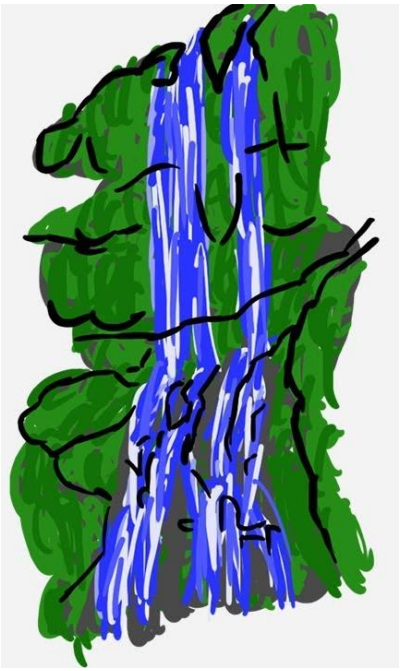
- Pros :
  - Robust analysis
- Cons :
  - Learning curve
  - Where data originated from ?

Hunting Bugs To Extinction with Static Analysis

Option 3 : Dataflow

- Collect run-time (dynamic) information about data in software while it is in a static state
- Sources :
  - Origin of data
  - Arbitrary inputs
- Sinks :
  - Dangerous functions
  - Exploitable targets





Hunting Bugs To Extinction with Static Analysis

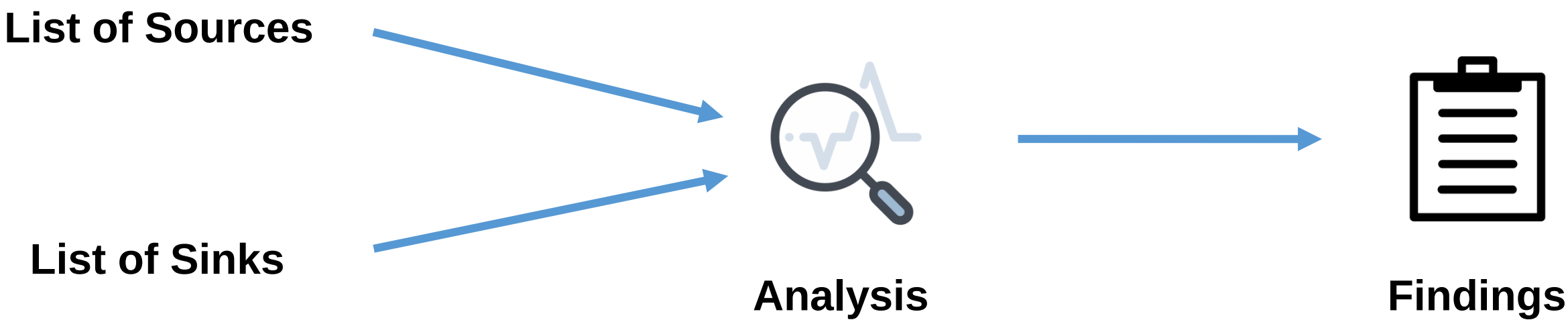
Option 3 : Source & Sink example

```
import os


def nslookup(request):
    domain = request.GET['domain']
    os.system("nslookup " + domain)
```

Hunting Bugs To Extinction with Static Analysis

Option 3 : Dataflow



Hunting Bugs To Extinction with Static Analysis

Option 3 : Dataflow

```
List inputs = Find_Interactive_Inputs();

List download_file_calls = All.Find_By_Name("download_file");
download_file_calls = download_file_calls.FindByParameterValue(1,"path",BinaryOperator.Equal);

List absolutePathTraversalSanitizers = Find_Absolute_PathTraversal_Sanitizers();

result = inputs.InfluencingOnAndNotSanitized(download_file_calls,
absolutePathTraversalSanitizers);
```

Hunting Bugs To Extinction with Static Analysis

Option 3 : Dataflow

- Pro :
  - Unified syntaxe to write scanning rules for different languages
  - Tracking data flow for deep analysis
- Cons :
  - Learning curve : very hard to tune
  - Distance between concerete code and correponding representation
  - Very slow for large code base
  - For projects > 2 Gb : Incremental scan = Normal scan
  - Libraries/frameworks = unknown sources


WHAT IF I TOLD YOU
THAT THERE IS ALWAYS ANOTHER WAY
made on imgur

Hunting Bugs To Extinction with Static Analysis

Option 3 : syntactical-(and semantic)-grep

- Free tool for writing lightweight checks with code patterns to find bugs using a familiar syntax.

```
$ semgrep -lang python -e 'subprocess.open(...)' /path/to/my/project
```

- An in-between solution
- Good support : Go · Java · JavaScript · JSON · Python · Ruby
- Beta : TypeScript · JSX · TSX

Hunting Bugs To Extinction with Static Analysis

Option 3 : syntactical-(and semantic)-grep

```
$ more rules.yaml
rules:
  - id: unsafe-usage-download_file
    pattern: download_file($X, search_type = "path")
    message: Unsafe usage of download_file method
    languages: [python]
    severity: WARNING
```

```
$ docker run --rm -v $(pwd):/home/repo returntocorp/semgrep –f rules.yaml file.py
```

Hunting Bugs To Extinction with Static Analysis

Technique 3 : syntactical-(and semantic)-grep

- Pros :
  - Fast
  - Easy to learn
- Cons :
  - We can’t track data over multiple files

To sum up …

	Lightweight / Basic Check	AST/CFG/… Complexe Check
Custom security policies	+++	++
Ease of use	+++	+
Speed Analysis	+++	+
Code coverage	++	+++

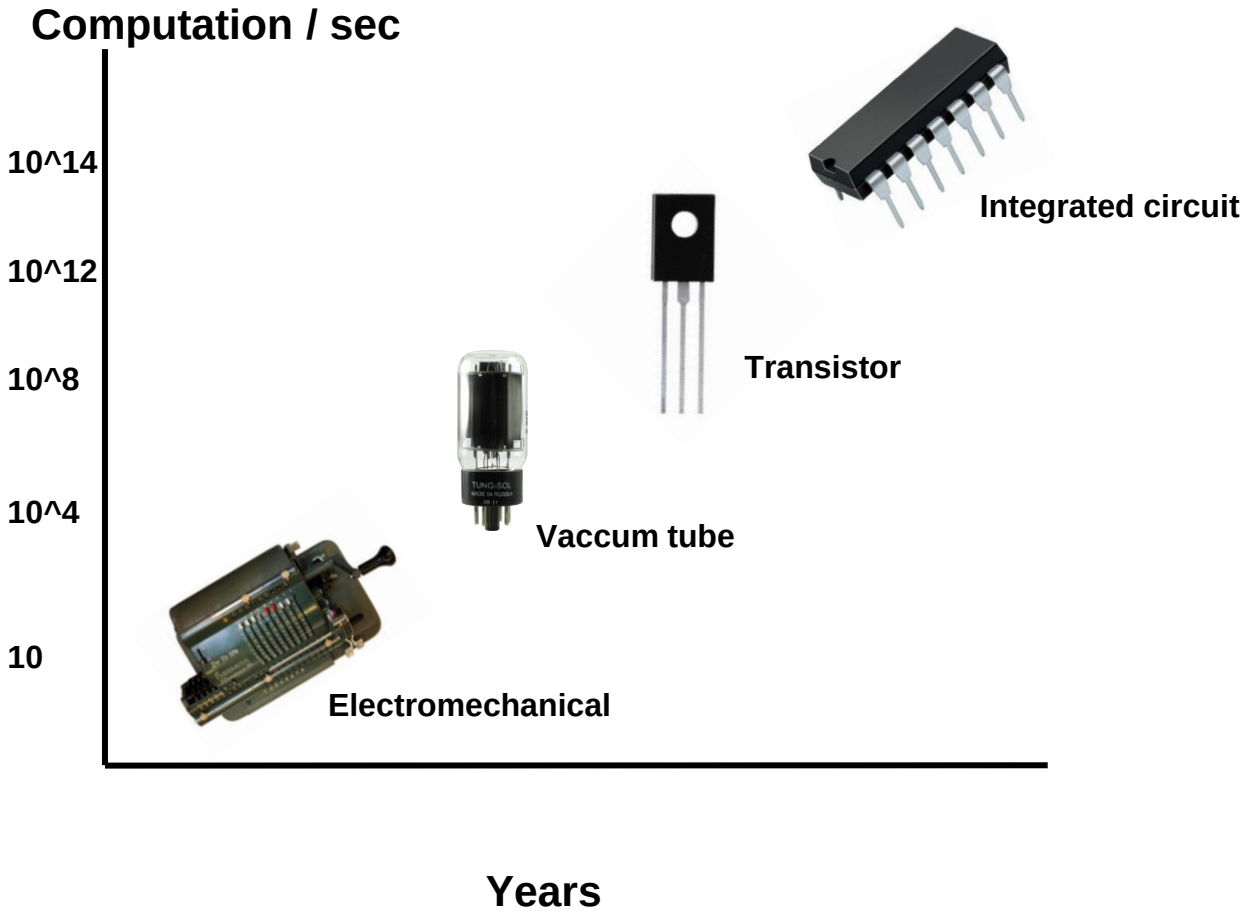
It's not all about detecting issues …

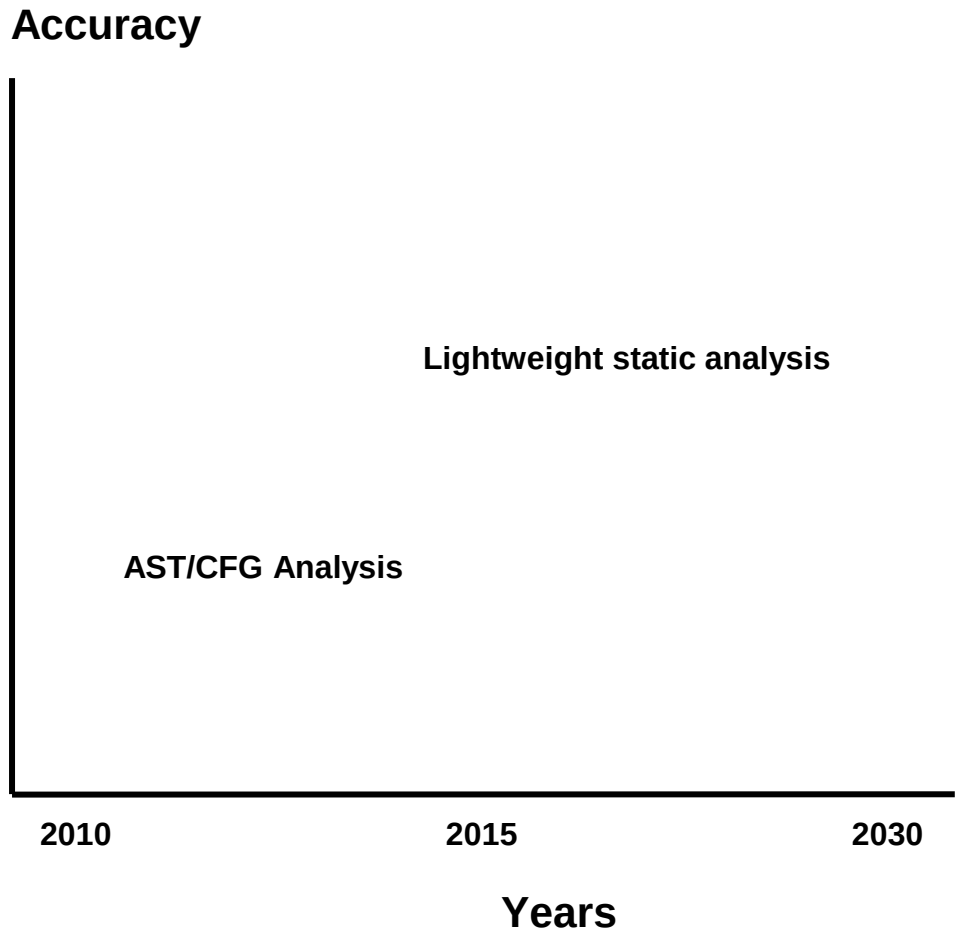
- Remember : the goal is to write secure apps …
- "Tools can't find every bug (FNs) … trying to will yield way too many FPs" - @clintgibler
- The raise of secure by default framework !
- Angular : `grep dangerouslySetInnerHTML`

It’s not all about detecting issues …

	Lightweight / Basic Check	AST/CFG/… Complexe Check	Lightweight / Basic Check + Secure by default
Custom security policies	**+++**	**++**	**+++**
Ease of use	**+++**	**+**	**+++**
Speed Analysis	**+++**	**+**	**+++**
Code coverage	**+**	**+++**	**++**

What does the future hold for us ?





sec4dev

THANKS!

Any questions?

You can find me at :

TWITTER : @abdel_tmr

LINKEDIN : /in/abdessamad-temmar/