

# NO SIZE FITS ALL

Empowering Engineers with Custom  
Application Security tests

Michal Kamensky  
Bounce Security

















**WHO AM I?**

**WHAT'S MY MOTIVATION?**

# MICHAL KAMENSKY



- Security researcher for Bounce Security
- Delivered talks at BSides TLV, OWASP AppSec IL and PyCon IL
- Strongly believe in combining security with the community

- Contact:



@Kam3nskyM



[michal@bouncesecurity.com](mailto:michal@bouncesecurity.com)

Volunteer/organizer for:

- Hackeriot  **HACKERIOT**
- BSides TLV 

# WHAT IS OUR MOTIVATION? BECOMING OMNISCIENT

1. having infinite awareness, understanding, and insight
2. possessed of universal or complete knowledge

<https://www.merriam-webster.com/dictionary/omniscient>



# BUT WHAT DOES THAT MEAN?



We want to enforce:

- Everything
  - Custom solutions that meet our needs
- Everywhere
  - Across our whole application
- All the time
  - Every CI/CD run

# WHAT IS A CUSTOM SOLUTION?

	<b>Generic Vulnerability</b>	<b>App-Specific Vulnerability</b>
<b>Standard Fix</b>	<b>Generic Solution</b>	<b>N/A</b>
<b>Bespoke Mitigation</b>	<b>Custom Solution</b>	<b>Custom Solution</b>



EVERYTHING

# GENERIC VULNERABILITY & STANDARD FIX = GENERIC SOLUTION

```
# Instead of this ...
cursor.execute(f"SELECT admin FROM users WHERE username = '{username}'");
# ...do this...
cursor.execute("SELECT admin FROM users WHERE username = %(username)s", {'username': username});

# From: https://snyk.io/blog/python-security-best-practices-cheat-sheet/
```

	Generic Vulnerability	App-Specific Vulnerability
Standard Fix	Generic Solution	N/A
Bespoke Mitigation	Custom Solution	Custom Solution



EVERYTHING

# SAST

\*SAST – Static Application Security Testing



EVERYTHING



# DAST

\*DAST – Dynamic Application Security Testing



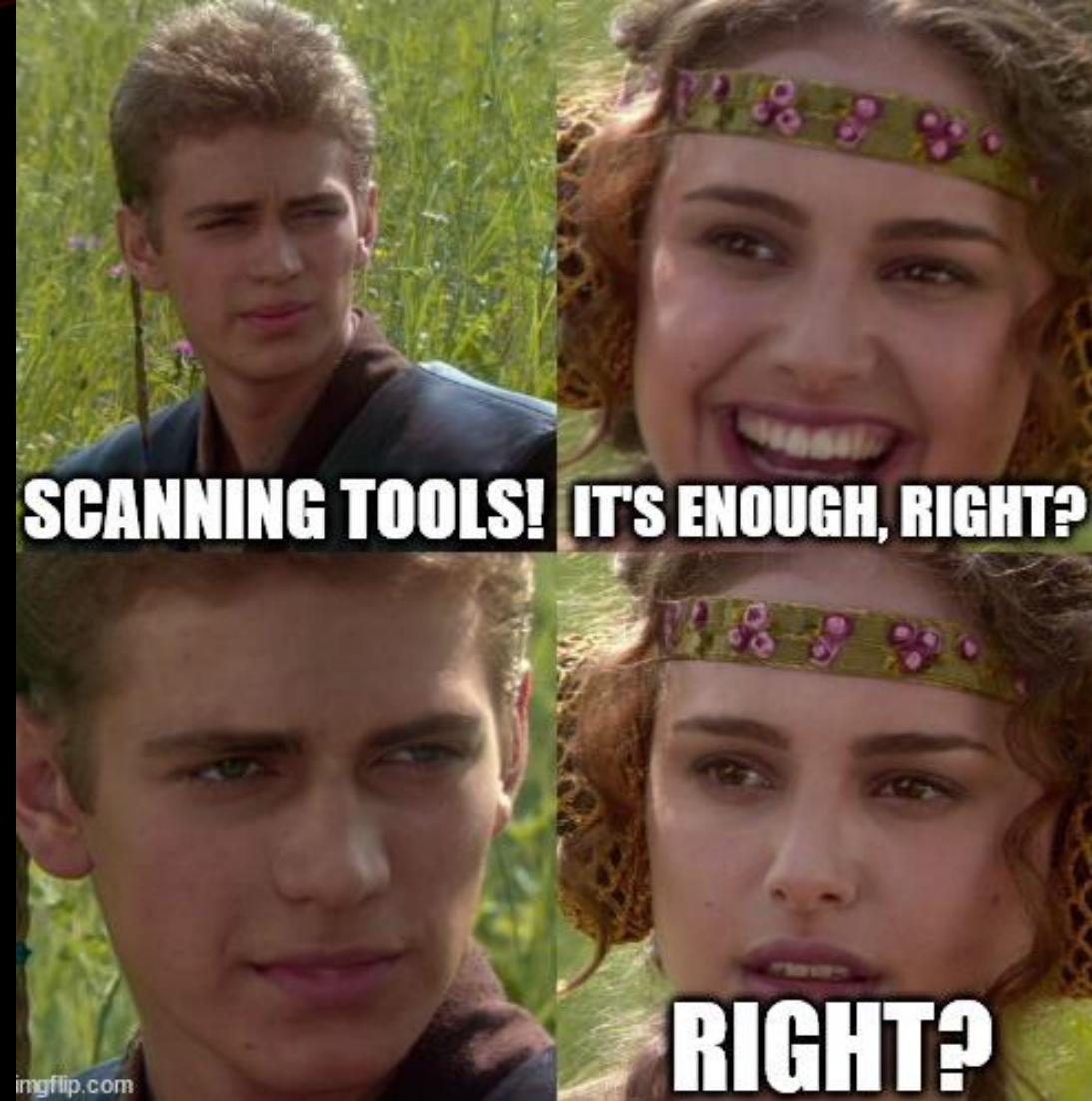
EVERYWHERE





EVERYTHING





EVERYTHING

# WHAT IS A CUSTOM SOLUTION?

	Generic Vulnerability	App-Specific Vulnerability
Standard Fix	Generic Solution	N/A
Bespoke Mitigation	Custom Solution	Custom Solution



EVERYTHING

# BUT WHAT ABOUT THIS?

	Generic Vulnerability	App-Specific Vulnerability
Standard Fix	Generic Solution	N/A
Bespoke Mitigation	Custom Solution	Custom Solution

```
1 import os
2
3 def getfile(input):
4     print("This code will be dangerous with untrusted input")
5     input = "/folder/safe/{0}".format(input)
6     print("Access a file at the following path: {0}\n".format(os.path.realpath(input)))
7     # f = open(input, "r")
```

## GENERIC VULNERABILITY (PATH TRAVERSAL)



EVERYTHING

```
1 dangerous_input = "../../../../../etc/passwd"
2
3 getfile(dangerous_input)
```

This code will be dangerous with untrusted input  
Access a file at the following path: /etc/passwd

# BESPOKE MITIGATION

No  
guarantees!

```
1 def sanitize_input(input):
2     print("This function will make the untrusted input safer:")
3     input2 = input.replace("../", "")
4     print('{0}' ' has been turned into '{1}'\n'.format(input, input2))
5     return input2
```

	Generic Vulnerability	App-Specific Vulnerability
Standard Fix	Generic Solution	N/A
Bespoke Mitigation	Custom Solution	Custom Solution



EVERYTHING

# CUSTOM SOLUTION

```
1 dangerous_input = "../../../../etc/passwd"  
2  
3 getfile(dangerous_input)  
4  
5 safe_input = sanitize_input(dangerous_input)  
6  
7 getfile(safe_input)
```

	Generic Vulnerability	App-Specific Vulnerability
Standard Fix	Generic Solution	N/A
Bespoke Mitigation	Custom Solution	Custom Solution

This code will be dangerous with untrusted input  
Access a file at the following path: /etc/passwd

This function will make the untrusted input safer:  
'../../../../etc/passwd' has been turned into 'etc/passwd'

This code will be dangerous with untrusted input  
Access a file at the following path: /folder/safe/etc/passwd



EVERYTHING

# CUSTOM SOLUTION – BUSINESS LOGIC



	Generic Vulnerability	App-Specific Vulnerability
Standard Fix	Generic Solution	N/A
Bespoke Mitigation	Custom Solution	Custom Solution

coupon codes		
30204		
93052		
60232		
50801		
26578		
39917		

- There's an option to use coupons on your app
- The coupons are validated against a database
- After a coupon is used, it's deleted from the database



EVERYTHING

# BUT WHAT DOES THAT MEAN?



We want to enforce:

- Everything
  - Custom solutions that meet our needs
- Everywhere
  - Across our whole application
- All the time
  - Every CI/CD run



# Semgrep

Semantic + Grep



EVERYWHERE



# BUT WHAT DOES THAT MEAN?



We want to enforce:

- Everything
  - Custom solutions that meet our needs
- Everywhere
  - Across our whole codebase/s
- All the time
  - Every CI run



# Cyber AI Transponster

## Chess



# EXAMPLES

3 examples:

1. Security Decorators
2. Unsanitized Input
3. Vulnerable Library

# DISCLAIMERS

- Contrived Examples
- Focus on content over style
- Focus on the two key lessons:
  - Custom solutions
  - Automated, syntax aware verification

# SECURITY DECORATORS



The problem we're addressing:

- We need to verify permissions before allowing cats to make move
- Want to use a centralized function to verify this

# SECURITY DECORATORS



Proposed security solution:

- Create decorator which verifies authorization
- Apply decorators to each sensitive function

# SECURITY DECORATORS



```
def require_update_purrrmission(func):  
    @wraps(func)  
  
    def wrapper(*args, **kwargs):  
        if not Permissions.UPDATE in permissions:  
            raise ValueError("Permission denied")  
        return func(*args, **kwargs)  
    return wrapper
```

```
class SensitiveFunctions:  
  
    @require_update_purrrmission  
    def update_score_board(cat1, cat2, game_result: GameResult):  
        print("Winner is {}".format(cat1))  
        # Update score board
```

# SECURITY DECORATORS



```
def require_update_purrrmission(func):
    @wraps(func)

    def wrapper(*args, **kwargs):
        if not Permissions.UPDATE in permissions:
            raise ValueError("Permission denied")
        return func(*args, **kwargs)
    return wrapper

class SensitiveFunctions:

    @require_update_purrrmission
    def update_score_board(cat1, cat2, game_result: GameResult):
        print("Winner is {}".format(cat1))
        # Update score board
```

# SECURITY DECORATORS



Proposed security solution:

- Create decorator which verifies authorization
- Apply decorators to each sensitive function

Challenge:

- What happens if we miss a sensitive function?

# SECURITY DECORATORS - CODE EXAMPLE

```
24 class SensitiveFunctions:  
25  
26     @require_update_purrrmission  
27     def update_score_board(cat1, cat2, game_result: GameResult):  
28         print("Winner is {}".format(cat1))  
29         # Update score board  
30  
31     def next_move_cat(cat, move):  
32         print("{} plays the next move which is {}".format(cat, move))  
33         play_move(cat, move)  
34         # Update game notes  
35  
36 SensitiveFunctions.next_move_cat("tabby", "Qxh6")  
37 SensitiveFunctions.update_score_board("tabby", "simba", GameResult.CAT1_WON)
```

# SECURITY DECORATORS - CODE EXAMPLE

```
24 class SensitiveFunctions:  
25       
26         @require_update_purrrmission  
27         def update_score_board(cat1, cat2, game_result: GameResult):  
28             print("Winner is {0}".format(cat1))  
29             # Update score board  
30           
31         def next_move_cat(cat, move):  
32             print("{0} plays the next move which is {1}".format(cat, move))  
33             play_move(cat, move)  
34             # Update game notes  
35           
36     SensitiveFunctions.next_move_cat("tabby", "Qxh6")  
37     SensitiveFunctions.update_score_board("tabby", "simba", GameResult.CAT1_WON)
```

# SECURITY DECORATORS - CODE EXAMPLE

```
24 class SensitiveFunctions:  
25  
26     @require_update_purrrmission  
27     def update_score_board(cat1, cat2, game_result: GameResult):  
28         print("Winner is {}".format(cat1))  
29         # Update score board  
30  
31     def next_move_cat(cat, move):  
32         print("{} plays the next move which is {}".format(cat, move))  
33         play_move(cat, move)  
34         # Update game notes  
35  
36 SensitiveFunctions.next_move_cat("tabby", "Qxh6")  
37 SensitiveFunctions.update_score_board("tabby", "simba", GameResult.CAT1_WON)
```

# SECURITY DECORATORS - RULE



```
1 rules:
2   - id: py_ex1_auth_decorators
3     patterns:
4       - pattern: |
5         def $FUNC(...):
6           ...
7       - focus-metavariable:
8         - $FUNC
9       - pattern-not-inside: |
10         @require_update_purrrmission
11         def $FUNC(...):
12           ...
13       - pattern-inside: |
14         class SensitiveFunctions:
15           ...
16         message: Function is missing the authorization decorator
17         languages:
18           - python
19         severity: ERROR
20
```

# SECURITY DECORATORS



```
24 class SensitiveFunctions:  
25  
26     @require_update_purrrmission  
27     def update_score_board(cat1, cat2, game_result: GameResult):  
28         print("Winner is {}".format(cat1))  
29         # Update score board  
30  
31     def next_move_cat(cat, move):  
32         print("{} plays the next move which is {}".format(cat, move))  
33         play_move(cat, move)  
34         # Update game notes  
35  
36 SensitiveFunctions.next_move_cat("tabby", "Qxh6")  
37 SensitiveFunctions.update_score_board("tabby", "simba", GameResult.CAT1_WON)
```

# DANGEROUS FUNCTION



The problem we're addressing:

- We need to execute OS commands including user input
- We need to prevent malicious input getting to the OS
- Command
- We cannot currently centralize the code or modify the run\_ping implementation

# DANGEROUS FUNCTION



```
def get_input():
    #some actions before getting an input
    return input("please enter address")
```

```
def run_ping(address):
    cmd = "ping -c 1 {0}".format(address)
    subprocess.Popen(cmd, shell=True)
```

```
new_address = get_input()
```

```
#ruleid: unsanitized_input
run_ping(new_address)
```

# DANGEROUS FUNCTION



```
def get_input():
    #some actions before getting an input
    return input("please enter address")

def run_ping(address):
    cmd = "ping -c 1 {}".format(address)
    subprocess.Popen(cmd, shell=True)

new_address = get_input()

#ruleid: unsanitized_input
run_ping(new_address)
```

# DANGEROUS FUNCTION



Proposed security solution:

- Create a sanitization function
- Use function on input before we call run\_ping

# DANGEROUS FUNCTION



```
def sanitize_it(user_input):  
    #Sanitize the input to avoid code execution  
    return user_input
```

```
def get_input():  
    #some actions before getting an input  
    return input("please enter address")
```

```
def run_ping(address):  
    cmd = "ping -c 1 {}".format(address)  
    subprocess.Popen(cmd, shell=True)
```

```
new_address = get_input()  
run_ping(new_address)
```

```
new_address = sanitize_it(new_address)  
run_ping(new_address)
```

# DANGEROUS FUNCTION



```
def sanitize_it(user_input):  
    #Sanitize the input to avoid code execution  
    return user_input
```

```
def get_input():  
    #some actions before getting an input  
    return input("please enter address")
```

```
def run_ping(address):  
    cmd = "ping -c 1 {}".format(address)  
    subprocess.Popen(cmd, shell=True)
```

```
new_address = get_input()  
run_ping(new_address)
```

```
new_address = sanitize_it(new_address)  
run_ping(new_address)
```

# DANGEROUS FUNCTION



```
def sanitize_it(user_input):
    #Sanitize the input to avoid code execution
    return user_input

def get_input():
    #some actions before getting an input
    return input("please enter address")

def run_ping(address):
    cmd = "ping -c 1 {}".format(address)
    subprocess.Popen(cmd, shell=True)

new_address = get_input()
run_ping(new_address)

new_address = sanitize_it(new_address)
run_ping(new_address)
```

# DANGEROUS FUNCTION



Proposed security solution:

- Create a sanitization function
- Use function on input before we call run\_ping

Challenge:

- How do we ensure we don't forget to do this somewhere?

# DANGEROUS FUNCTION - CODE EXAMPLE

```
15 new_address = get_input()  
16  
17 #ruleid: py_ex2_unsanitized_input  
18 run_ping(new_address)  
19  
20 new_address = sanitize_it(new_address)  
21  
22 #ok: py_ex2_unsanitized_input  
23 run_ping(new_address)  
24
```

# DANGEROUS FUNCTION - CODE EXAMPLE

```
15 new_address = get_input()  
16  
17 #ruleid: py_ex2_unsanitized_input  
18 run_ping(new_address)  
19  
20 new_address = sanitize_it(new_address)  
21  
22 #ok: py_ex2_unsanitized_input  
23 run_ping(new_address)  
24
```

# DANGEROUS FUNCTION - CODE EXAMPLE

```
15 new_address = get_input()  
16  
17 #ruleid: py_ex2_unsanitized_input  
18 run_ping(new_address)  
19  
20 new_address = sanitize_it(new_address)  
21  
22 #ok: py_ex2_unsanitized_input  
23 run_ping(new_address)  
24
```

# DANGEROUS FUNCTION - CODE EXAMPLE

```
15 new_address = get_input()  
16  
17 #ruleid: py_ex2_unsanitized_input  
18 run_ping(new_address)  
19  
20 new_address = sanitize_it(new_address)  
21  
22 #ok: py_ex2_unsanitized_input  
23 run_ping(new_address)  
24
```

# DANGEROUS FUNCTION - CODE EXAMPLE

```
15 new_address = get_input()  
16  
17 #ruleid: py_ex2_unsanitized_input  
18 run_ping(new_address)  
19  
20 new_address = sanitize_it(new_address) # Line 20 is highlighted with a red box  
21  
22 #ok: py_ex2_unsanitized_input  
23 run_ping(new_address)  
24
```

# DANGEROUS FUNCTION



```
1 import subprocess
2
3 def sanitize_it(user_input):
4     #Sanitize the input to avoid code execution
5     return user_input
6
7 def get_input():
8     #some actions before getting an input
9     return input("please enter address")
10
11 def run_ping(address):
12     cmd = "ping -c 1 {}".format(address)
13     subprocess.Popen(cmd, shell=True)
14
```

# DANGEROUS FUNCTION - RULE



```
1 rules:
2   - id: py_ex2_unsanitized_input
3     mode: taint
4     pattern-sources:
5       - pattern: input(...)
6     pattern-sinks:
7       - pattern: run_ping(...)
8     pattern-sanitizers:
9       - pattern: sanitize_it(...)
10    message: run_ping is being used without first sanitizing
11    languages:
12      - python
13    severity: ERROR
14
```

# DANGEROUS FUNCTION



```
16
17 #ruleid: py_ex2_unsanitized_input
18 run_ping(new_address)
19
20 new_address = sanitize_it(new_address)
21
22 #ok: py_ex2_unsanitized_input
23 run_ping(new_address)
24
```

# VULNERABLE LIBRARY

A large red number 3 is centered within a white circle. The circle has a thick black outline and a slight shadow, giving it a three-dimensional appearance.

The problem we're addressing:

- We're using a vulnerable library (py7zr)
- We don't want to upgrade (yet?)
- We believe that the vulnerable function is not in use

# VULNERABLE LIBRARY

3

## CVE-2022-44900 Detail

### Description

A directory traversal vulnerability in the `SevenZipFile.extractall()` function of the python library `py7zr` v0.20.0 and earlier allows attackers to write arbitrary files via extracting a crafted 7z file.

# VULNERABLE LIBRARY

3

Proposed security solution:

- Delay upgrade
- Search across codebase to ensure we are not using vulnerable function

Challenge:

- What if we start using this vulnerable function in the future

# VULNERABLE LIBRARY – CODE EXAMPLE

```
7 def extract_game(game_name):
8     with py7zr.SevenZipFile(game_name, 'r') as archive:
9         archive.extractall()
10
11 def extract_game2(game_name):
12     py7 = py7zr.SevenZipFile()
13     py7.extractall(path)
14
15 def compress_game(game_name):
16     with py7zr.SevenZipFile(game_name, 'w') as archive:
17         archive.writeall('/cats/Simba/games', 'games')
18
19 def parse_data(json):
20     data_in = json.loads(json)
```

# VULNERABLE LIBRARY – CODE EXAMPLE

```
7 def extract_game(game_name):
8     with py7zr.SevenZipFile(game_name, 'r') as archive:
9         archive.extractall()
10
11 def extract_game2(game_name):
12     py7 = py7zr.SevenZipFile()
13     py7.extractall(path)
14
15 def compress_game(game_name):
16     with py7zr.SevenZipFile(game_name, 'w') as archive:
17         archive.writeall('/cats/Simba/games', 'games')
18
19 def parse_data(json):
20     data_in = json_li.extractall(json)
```

# VULNERABLE LIBRARY – CODE EXAMPLE

```
7 def extract_game(game_name):  
8     with py7zr.SevenZipFile(game_name, 'r') as archive:  
9         archive.extractall()  
10  
11 def extract_game2(game_name):  
12     py7 = py7zr.SevenZipFile()  
13     py7.extractall(path)  
14  
15 def compress_game(game_name):  
16     with py7zr.SevenZipFile(game_name, 'w') as archive:  
17         archive.writeall('/cats/Simba/games', 'games')  
18  
19 def parse_data(json):  
20     data_in = json.loads(json)
```

# VULNERABLE LIBRARY

Why do we need  
custom verification:

- SAST usually  
doesn't look for  
CVEs in known  
libraries
- SCA usually just  
looks for the library,  
not the function

```
7 def extract_game(game_name):  
8     with py7zr.SevenZipFile(game_name, 'r') as archive:  
9         archive.extractall()  
10  
11 def extract_game2(game_name):  
12     py7 = py7zr.SevenZipFile()  
13     py7.extractall(path)  
14  
15 def compress_game(game_name):  
16     with py7zr.SevenZipFile(game_name, 'w') as archive:  
17         archive.writeall('/cats/Simba/games', 'games')  
18  
19 def parse_data(json):  
20     data_in = json_li.extractall(json)
```

\*SAST – Static Application Security Testing

\*SCA – Software Composition Analysis

# VULNERABLE LIBRARY - RULE

3

```
1 rules:
2   - id: py_ex3_vulnerable_library
3     mode: taint
4     pattern-sources:
5       - pattern: py7zr.$FUNC(...)
6     pattern-sinks:
7       - pattern: $0.extractall(...)
8     message: Semgrep found a match
9     languages:
10    - python
11    severity: ERROR
12
```

# VULNERABLE LIBRARY

3

```
7 def extract_game(game_name):
8     with py7zr.SevenZipFile(game_name, 'r') as archive:
9         archive.extractall()
10
11 def extract_game2(game_name):
12     py7 = py7zr.SevenZipFile()
13     py7.extractall(path)
14
15 def compress_game(game_name):
16     with py7zr.SevenZipFile(game_name, 'w') as archive:
17         archive.writeall('/cats/Simba/games', 'games')
18
19 def parse_data(json):
20     data_in = json.loads(json)
21
```

# DAST

\*DAST – Dynamic Application Security Testing



EVERYWHERE

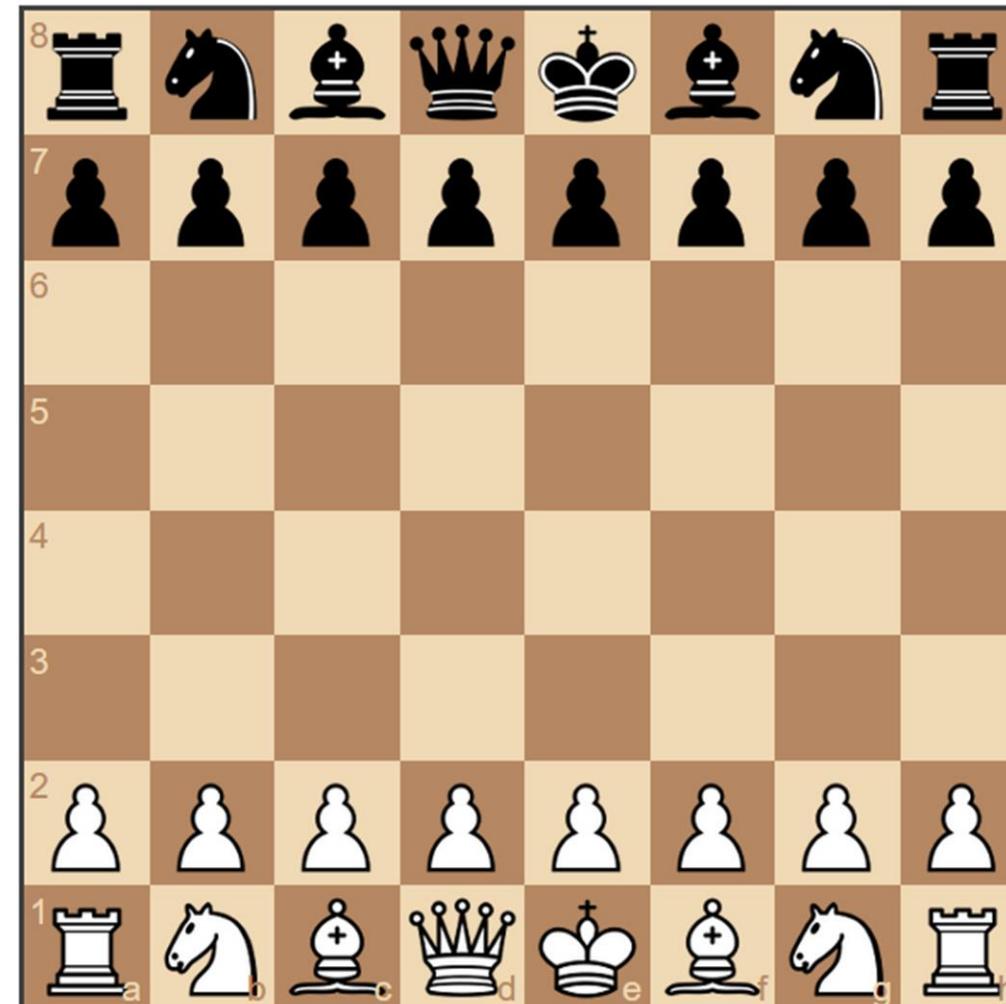




EVERYWHERE

 **Bounce**  
SECURITY

# C.A.T Chess



White to move

# EXAMPLES

4 examples:

1. “Bad Word” Checker
2. Race Condition
3. SSRF

# “BAD WORD” CHECKER



The problem we’re addressing:

- We need to make sure cats aren’t registering with offensive names, such as:
  - Cucumber
  - Dog
  - Roomba
- Want to make sure those words are not part of a username



# “BAD WORD” CHECKER



Proposed security solution:

- Check username input using a complicated regex

# “BAD WORD” CHECKER



Proposed security solution:

- Check username input using a complicated regex

Challenge:

- What happens if we missed some combination?
- What if the regex is not applied to every username input?

# “BAD WORD” CHECKER - TEMPLATE

```
8   http:
9     - method: GET
10    path:
11      - "{{BaseUrl}}"
12
13   payloads:
14     bad_words:
15       - "!"
16       - "@"
17       - "$"
18       - "`"
19
20   fuzzing:
21     - part: query
22       mode: single
23       type: infix
24       keys:
25         - username
26       fuzz:
27         - "{{bad_words}}"
28
29   matchers:
30     - type: word
31       part: body
32       words:
33         - "Username is invalid as it contains restricted words"
34       negative: true
```

# RACE CONDITION



The problem we're addressing:

- Making sure a discount coupon can't be used multiple times

# RACE CONDITION



Proposed security solution:

- Check the validity of the coupon before applying it, and making sure to invalidate it afterwards

# RACE CONDITION



Proposed security solution:

- Check the validity of the coupon before applying it, and making sure to invalidate it afterwards

Challenge:

- Possible race condition between applying the coupon and invalidating it

# RACE CONDITION - TEMPLATE



```
8   http:
9     - raw:
10      - |
11        POST /validateCoupon HTTP/1.1
12        Host: {{Hostname}}
13        Content-Type: application/json
14        Connection: keep-alive
15        credentials: 'include'
16        Cookie: connect.sid=s%3AqFpkbCmHn7hpvqKgCm
17
18        {"coupon":"123"}
19
20        race: true
21        race_count: 3
22
23        matchers:
24          - type: status
25            status:
26              - 200
```

# SSRF



The problem we're addressing:

Avoiding SSRF attacks on our endpoints.

Test case - the profile picture loading mechanism

# SSRF

3

Proposed security solution:

- Whitelisting domains

# SSRF

3

Proposed security solution:

- Whitelisting domains

Challenge:

- Checking for SSRF across all the relevant endpoints

# SSRF

3

Interactsh-url:

- Creates a random subdomain at .oast
- Checks whether your domain interacted with the given subdomain

# SSRF- TEMPLATE

3

```
1 id: blind-ssrf-detection
2
3 info:
4   name: Blind SSRF Detection
5   author: bounce
6   severity: high
7
8 http:
9   - method: GET
10  path:
11    - "{{BaseUrl}}/getCatImage?gender=https://{{interactsh-url}}&enable=true"
12
13
14 matchers:
15   - type: word
16     part: interactsh_protocol
17     words:
18       - "http"
```

# BUT WHAT DOES THAT MEAN?



We want to enforce:

- Everything
  - Custom solutions that meet our needs
- Everywhere
  - Across our whole codebase/s
- All the time
  - Every CI run

# SO, HOW DO WE DO THIS ALL THE TIME?

- Let's see a demo!



ALL THE TIME



omniscientexamples/.github/wor x +



github.com/BounceSecurity/omniscientexamples/blo...



43



← Files

main



## omniscientexamples / .github / workflows / semgrep\_py\_plain.yml



joshbouncesecurity Split workflows ✘

19 minutes ago



44 lines (40 loc) • 1.66 KB

Code

Blame

Raw



1 # Name of this GitHub Actions workflow.

2 name: Semgrep Python Plain

3

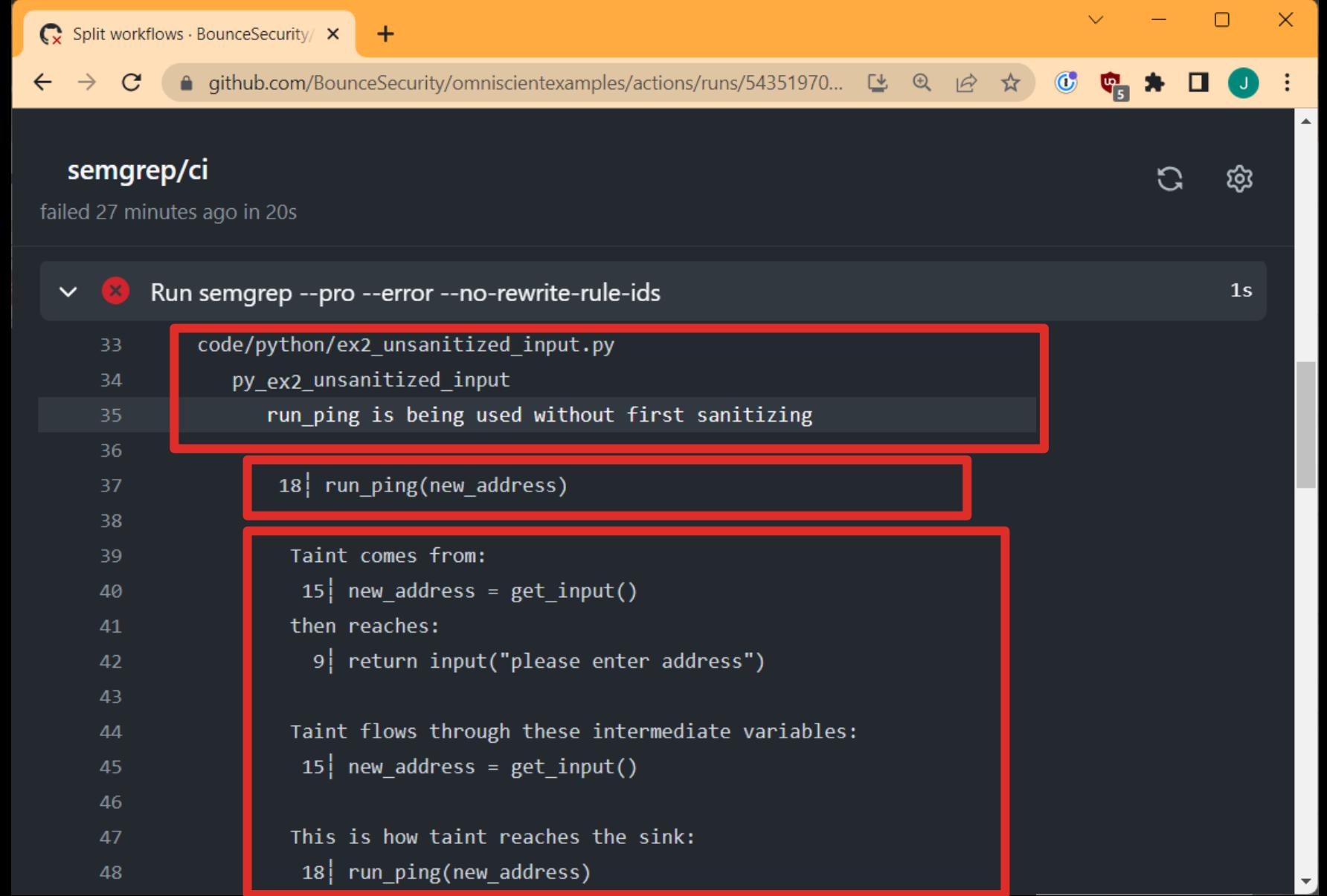
4 on:

```
4   on:
5     # Scan changed files in PRs (diff-aware scanning):
6     #pull_request: {}
7     # Scan on-demand through GitHub Actions interface:
8     workflow_dispatch: {}
9     # Scan mainline branches and report all findings:
10    push:
11      branches: ["master", "main"]
12    # Schedule the CI job (this method uses cron syntax):
13    #schedule:
14    #  - cron: '20 13 * * *' # Sets Semgrep to scan every day at 17:20 UTC.
15    # # It is recommended to change the schedule to a random time.
16
17  jobs:
18    semgrep:
19      # User-definable name of this GitHub Actions job:
20      name: semgrep/ci
21      # If you are self-hosting, change the following `runs-on` value:
22      runs-on: ubuntu-latest
```

```
26 # A Docker image with Semgrep installed. Do not change this.
27 image: returntocorp/semgrep
28
29 # Skip any PR created by dependabot to avoid permission issues:
30 if: (github.actor != 'dependabot[bot]')
31
32 steps:
33   # Fetch project source with GitHub Actions Checkout.
34   - uses: actions/checkout@v3
35   - run: semgrep install-semgrep-pro
36   # Run semgrep on the command line of the docker image.
37   - run: semgrep --pro --error --no-rewrite-rule-ids
38
39   env:
40     # Add the rules that Semgrep uses by setting the SEMGREP_RULES environment variable.
41     SEMGREP_RULES: >
42       semgrep_rules/py_ex1_auth_decorators.yml
43       semgrep_rules/py_ex2_unsanitized_input.yml
44       semgrep_rules/py_ex3_vulnerable_library.yml
```

# DEMO

- Results in Workflow log



The screenshot shows a browser window displaying a GitHub Actions workflow log. The title bar says "Split workflows · BounceSecurity/" and the URL is "github.com/BounceSecurity/omniscientexamples/actions/runs/54351970...". The workflow name is "semgrep/ci" and it failed 27 minutes ago in 20s. A red box highlights the following log output:

```
Run semgrep --pro --error --no-rewrite-rule-ids
  code/python/ex2_unsanitized_input.py
    py_ex2_unsanitized_input
      run_ping is being used without first sanitizing
  18| run_ping(new_address)
  Taint comes from:
    15| new_address = get_input()
  then reaches:
    9| return input("please enter address")
  Taint flows through these intermediate variables:
    15| new_address = get_input()
  This is how taint reaches the sink:
    18| run_ping(new_address)
```



ALL THE TIME

# DEMO

- Results uploaded to GitHub

SPLIT WORKFLOWS · BOUNCESECURITY/...

semgrep/ci

failed 33 minutes ago in 29s

Run semgrep --pro --error --no-rewrite-rule-ids --sarif --output=semgrep.sarif 1s

Scan was limited to files tracked by git.

Ran 4 rules on 4 files: 7 findings.

Error: Process completed with exit code 1.

Upload SARIF file for GitHub Advanced Security Dashboard 6s

Run github/codeql-action/upload-sarif@v2

/usr/bin/docker exec f06586f5148f3056d9257266a10da5eb214be8c1bb1745b6f372f958cf791a56 sh -c "cat /etc/\*release | grep ^ID"

Uploading results

Waiting for processing to finish

Post Run actions/checkout@v3 0s

Stop containers 0s

The screenshot shows a GitHub Actions workflow named 'semgrep/ci' that failed 33 minutes ago. The first step, 'Run semgrep', failed with an exit code of 1 due to finding 7 security issues. The second step, 'Upload SARIF file for GitHub Advanced Security Dashboard', succeeded after 6 seconds. A red box highlights the error message from the semgrep step and the entire Sarif upload step. A small icon of a character with a speech bubble is in the bottom left corner, and the text 'ALL THE TIME' is written next to it.

# DEMO

- Results uploaded to GitHub



ALL THE TIME

The screenshot shows a GitHub repository page for 'BounceSecurity / omniscientexamples'. The top navigation bar includes links for Code, Pull requests, Actions, Security (which has 7 notifications), Insights, and Settings. The main content area is titled 'Security overview'.

**Reporting**

- Policy**
- Advisories**

**Vulnerability alerts**

- Dependabot**
- Code scanning** (has 7 notifications)
- Secret scanning**

**Security policy • Disabled**  
Define how users should report security vulnerabilities for this repository [Set up a security policy](#)

**Security advisories • Enabled**  
View or disclose security advisories for this repository [View security advisories](#)

**Private vulnerability reporting • Disabled**  
Allow users to privately report potential security issues [Enable vulnerability reporting](#)

# DEMO

- Results uploaded to GitHub



**ALL THE TIME**

Code scanning alerts · BounceSec x +

github.com/BounceSecurity/omniscientexamples/security/code-scanning

Code scanning

Semgrep is reporting errors

Tool status 1 + Add tool

is:open branch:main

7 Open ✓ 44 Closed

Language Tool Branch Rule Severity Sort

**Semgrep Finding: py\_ex3\_vulnerable\_library** ✖ Error main  
#50 opened 7 months ago • Detected by Semgrep in code/python/ex3\_vulnerable\_library.p...:13

**Semgrep Finding: py\_ex3\_vulnerable\_library** ✖ Error main  
#49 opened 7 months ago • Detected by Semgrep in code/python/ex3\_vulnerable\_library.p...:9

**Semgrep Finding: py\_ex2\_unsanitized\_input** ✖ Error main  
#48 opened 7 months ago • Detected by Semgrep in code/python/ex2\_unsanitized\_input.py:18

**Semgrep Finding: py\_ex1\_auth\_decorators** ✖ Error main

# DEMO

- Results uploaded to GitHub



ALL THE TIME

The screenshot shows a GitHub browser interface with a yellow header bar. The title bar says "Semgrep Finding: py\_ex1\_wrong\_". The address bar shows the URL "github.com/BounceSecurity/omniscientexamples/security/code-scanning/51". The main content is titled "Code scanning alerts / #51" and "Semgrep Finding: py\_ex1\_auth\_decorators". A green button labeled "Open" is visible. The alert details a file "code/python/ex1\_auth\_decorators.py" at line 31, which contains the code "def next\_move\_cat(cat, move):". A red box highlights this line. To the right, a "Severity" section shows an "Error" status with a red shield icon. Below it, "Affected branches" show the "main" branch with 2 issues. The alert message states: "Function is missing the authorization decorator" and "Semgrep". The full code snippet shown is:

```
28     print("Winner is {0}".format(cat1))
29     # Update score board
30
31 def next_move_cat(cat, move):
Function is missing the authorization decorator
Semgrep
32     print("{0} plays the next move which is {1}".format(cat, move))
33     play_move(cat, move)
34     # Update game notes
```

# SO, HOW DO I DO THIS ALL THE TIME?

- Runs on every CI process in a simple way
- No special code processing/preparation!!!
- Alerts on discrepancies
- You can either:
  - Block CI pipeline
  - Setup out of band alerting



ALL THE TIME

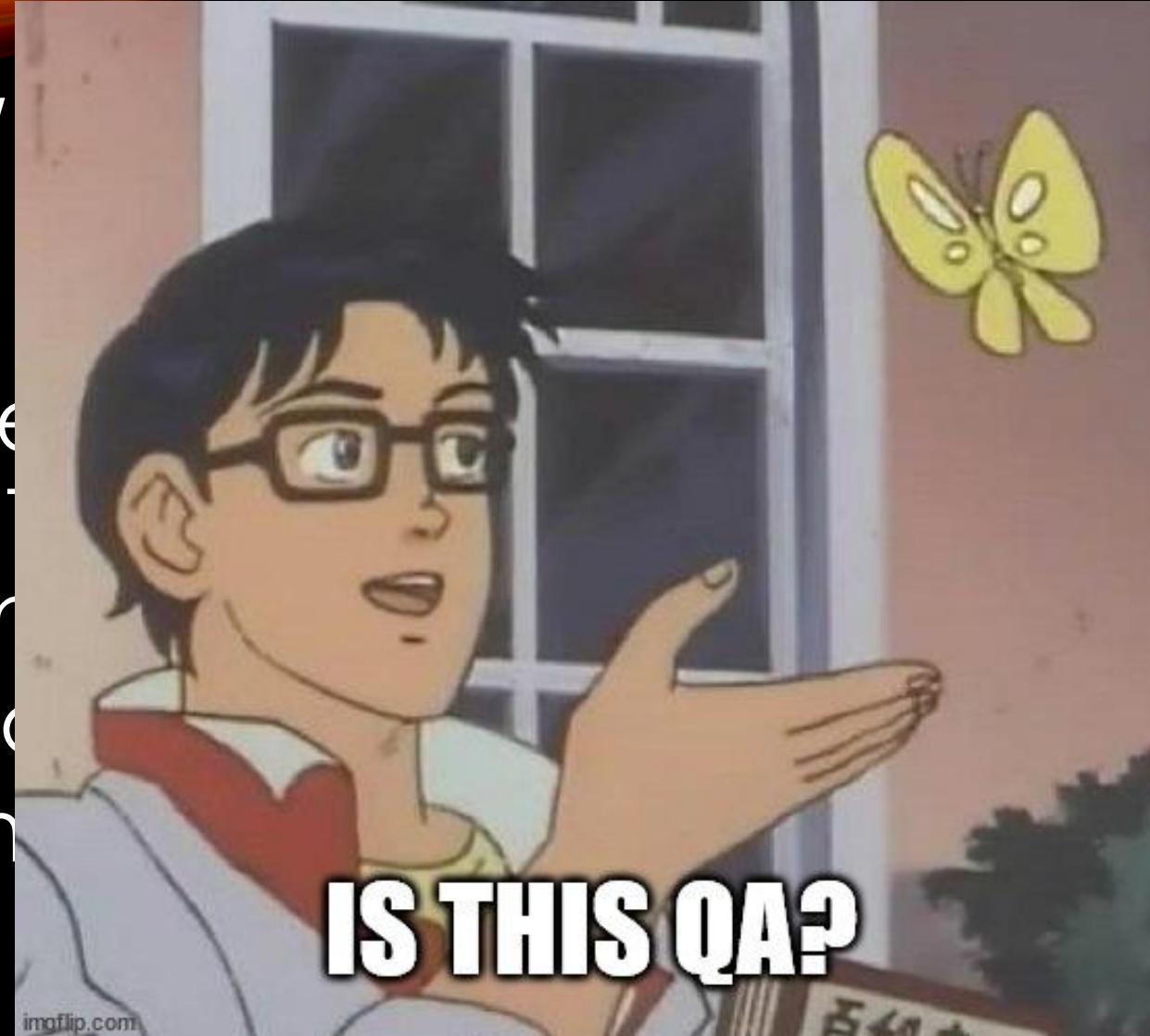
# SO HOW DO YOU START?



- Test case analyses – what can be automated?
- Focus on recurring issues
- Regression testing
- CI/CD Integration

SO HOW

- Test case automation
- Focus on
- Regression
- CI/CD Int



**IS THIS QA?**

imaflick.com

# SEMGREP LIMITATIONS

- Cross file scanning
- Not all languages
- Limited number of team members for free

# ALTERNATIVES?



EVERYWHERE

# SAST

\*SAST – Static Application Security Testing



EVERYWHERE



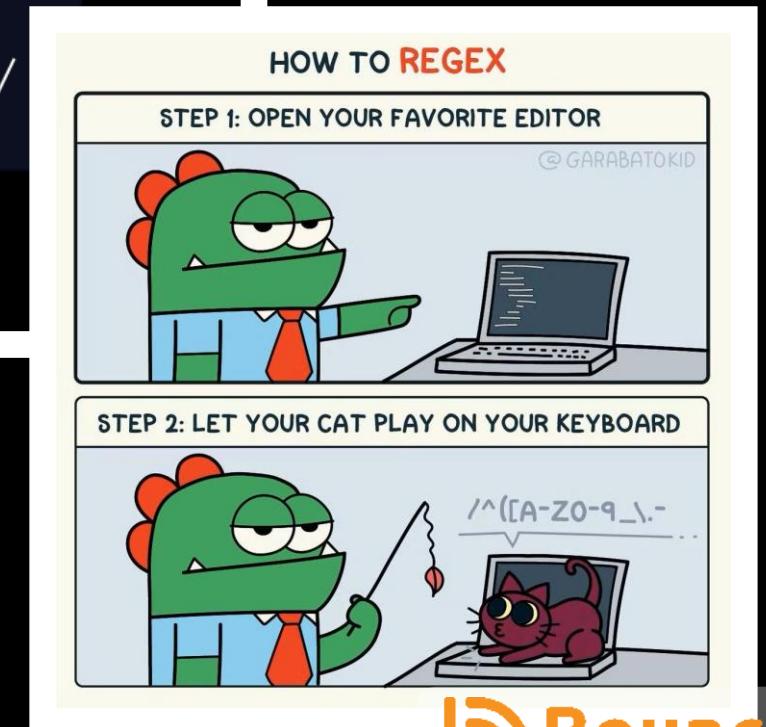
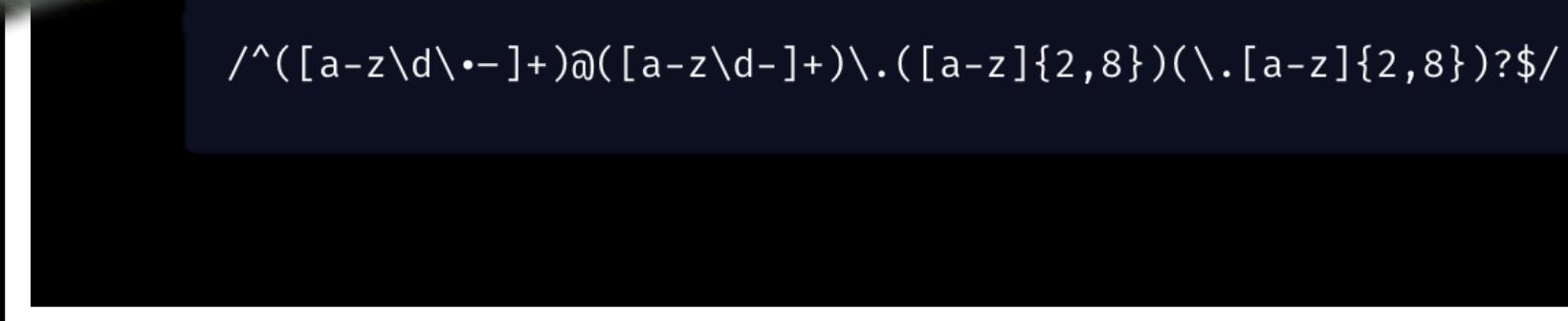
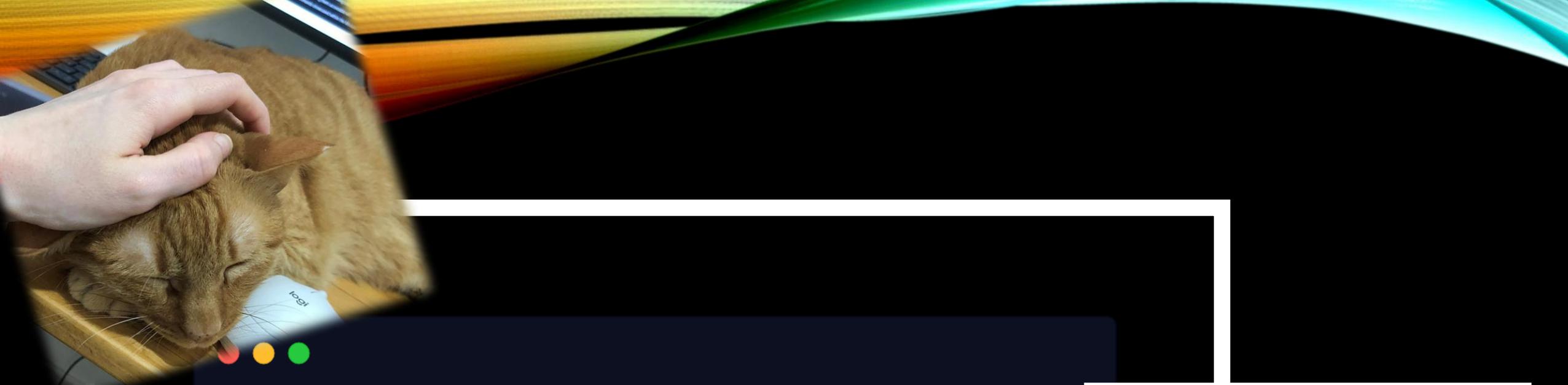


```
1 ► Run github/codeql-action/analyze@v1
26 /opt/hostedtoolcache/CodeQL/0.0.0-20220214/x64/codeql/codeql version --format=terse
27 2.8.1
28 ► Extracting javascript
293 ► Finalizing javascript
297 ► Running queries for javascript
789 ► Interpreting results for javascript
1050 Analysis produced the following diagnostic data:
1051
1052 | Diagnostic | Summary |
1053 +-----+-----+
1054 | Extraction errors | 0 results |
1055 | Successfully extracted files | 20 results |
1056 Analysis produced the following metric data:
1057
1058 | Metric | Value |
1059 +-----+-----+
1060 | Total lines of JavaScript and TypeScript code in the database | 687 |
1061 | Total lines of user written JavaScript and TypeScript code in the database | 687 |
1062
1063 /opt/hostedtoolcache/CodeQL/0.0.0-20220214/x64/codeql/codeql database print-baseline
```



EVERYWHERE





EVERYWHERE

# PROXIES

PortSwigger/  
**bambdas**

Bambdas collection for Burp Suite Professional and Community.

6

Contributors

1

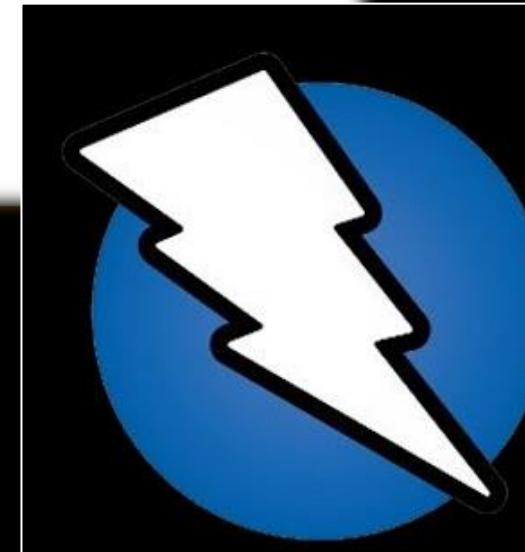
Issue

11

Stars

3

Forks



Zed Attack Proxy  
Setup



EVERYWHERE

# KEY TAKEAWAYS

- Everything
  - Not every vulnerability has a generic solution
  - Some vulnerabilities are custom to our situation
- Everywhere
  - Simple but syntax and flow aware scanning
  - Scan the whole codebase/app or just relevant files/API endpoints
- All the time
  - Simple one-liner to run the scan
  - Easy to integrate and alert from regular CI/CD





Credit: Emette Cohen Douglan

# QUESTIONS?

- Everything
  - Not every vulnerability has a generic solution
  - Some vulnerabilities are custom to our situation
- Everywhere
  - Simple but syntax and flow aware scanning
  - Scan the whole codebase/app or just relevant files/API endpoints
- All the time
  - Simple one-liner to run the scan
  - Easy to integrate and alert from regular CI/CD



Michal Kamensky

✉ [michal@bouncesecurity.com](mailto:michal@bouncesecurity.com)

🐦 @Kam3nskyM