OMNISCIENT APPSEC

Custom, Continuous Security Verification

Michal Kamensky & Josh Grossman
Bounce Security

Bounce SECURITY

Bounce
SECURITY

Bounce SECURITY

Bounce SECURITY

Bounce SECURITY

Bounce SECURITY

Bounce SECURITY

Bounce SECURITY

WHO ARE WE?

WHAT IS OUR MOTIVATION?

Bounce SECURITY

WHO ARE WE?

- Michal Kamensky
- Josh Grossman

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 codebase/s
- All the time
  - Every CI run

Bounce SECURITY

WHAT IS A CUSTOM SOLUTION?

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

EVERYTHING

Bounce SECURITY

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

Bounce SECURITY

Pylint

Star your Python code!

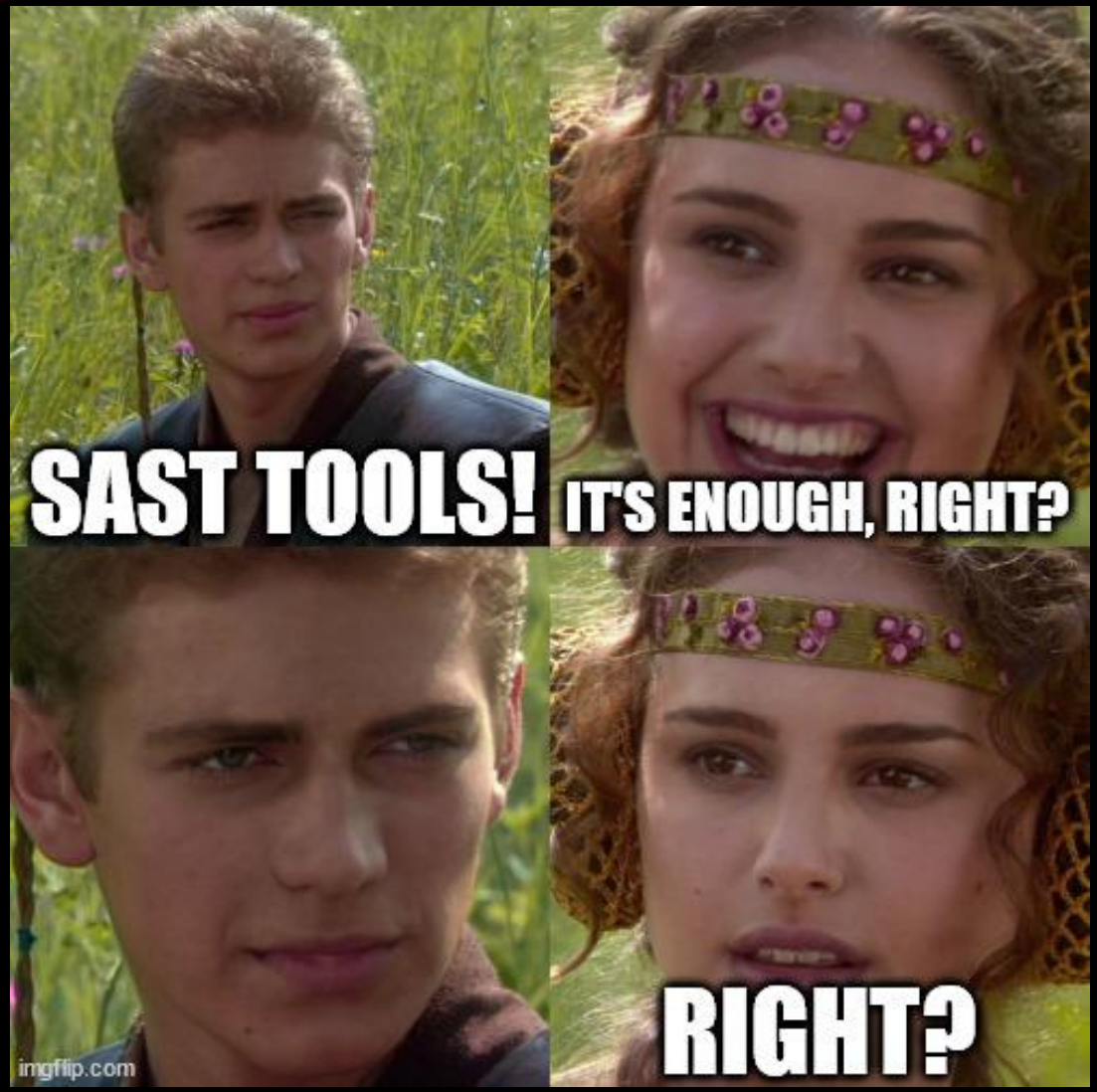
SAST

Bandit

*SAST – Static Application Security Testing

EVERYTHING

Bounce SECURITY



EVERYTHING

*SAST – Static Application Security Testing

Bounce SECURITY

WHAT IS A CUSTOM SOLUTION?

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

EVERYTHING

Bounce SECURITY

BUT WHAT ABOUT THIS?

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

```python
import os

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

GENERIC VULNERABILITY (PATH TRAVERSAL)

EVERYTHING

```python
dangerous_input = "../../../../etc/passwd"

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

Bounce SECURITY

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

EVERYTHING

```
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
```

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

EVERYTHING

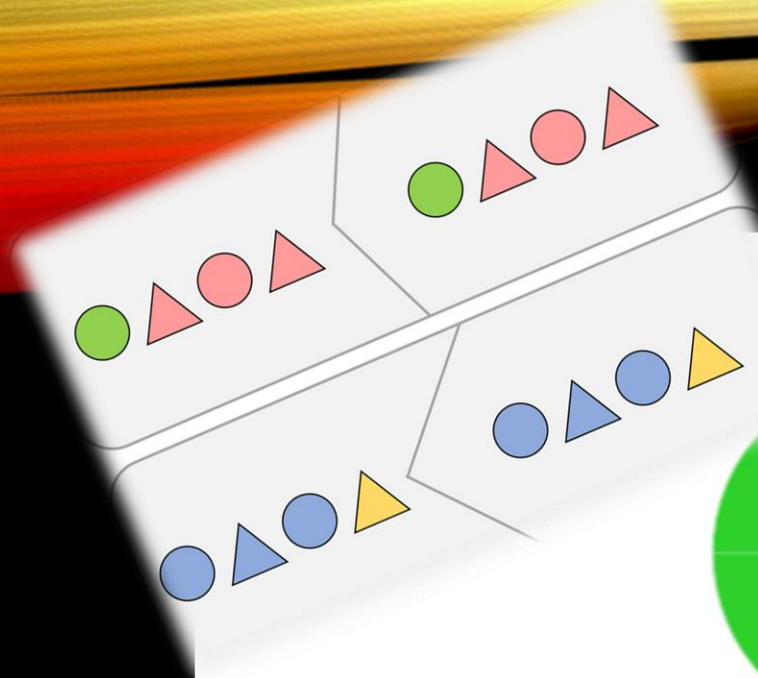
```
This code will be dangerous with untrusted input
Access a file at the following path: /etc/passwd

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
```

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

Bounce SECURITY



Semgrep

Semantic + Grep

EVERYWHERE

Bounce SECURITY

ALTERNATIVES?

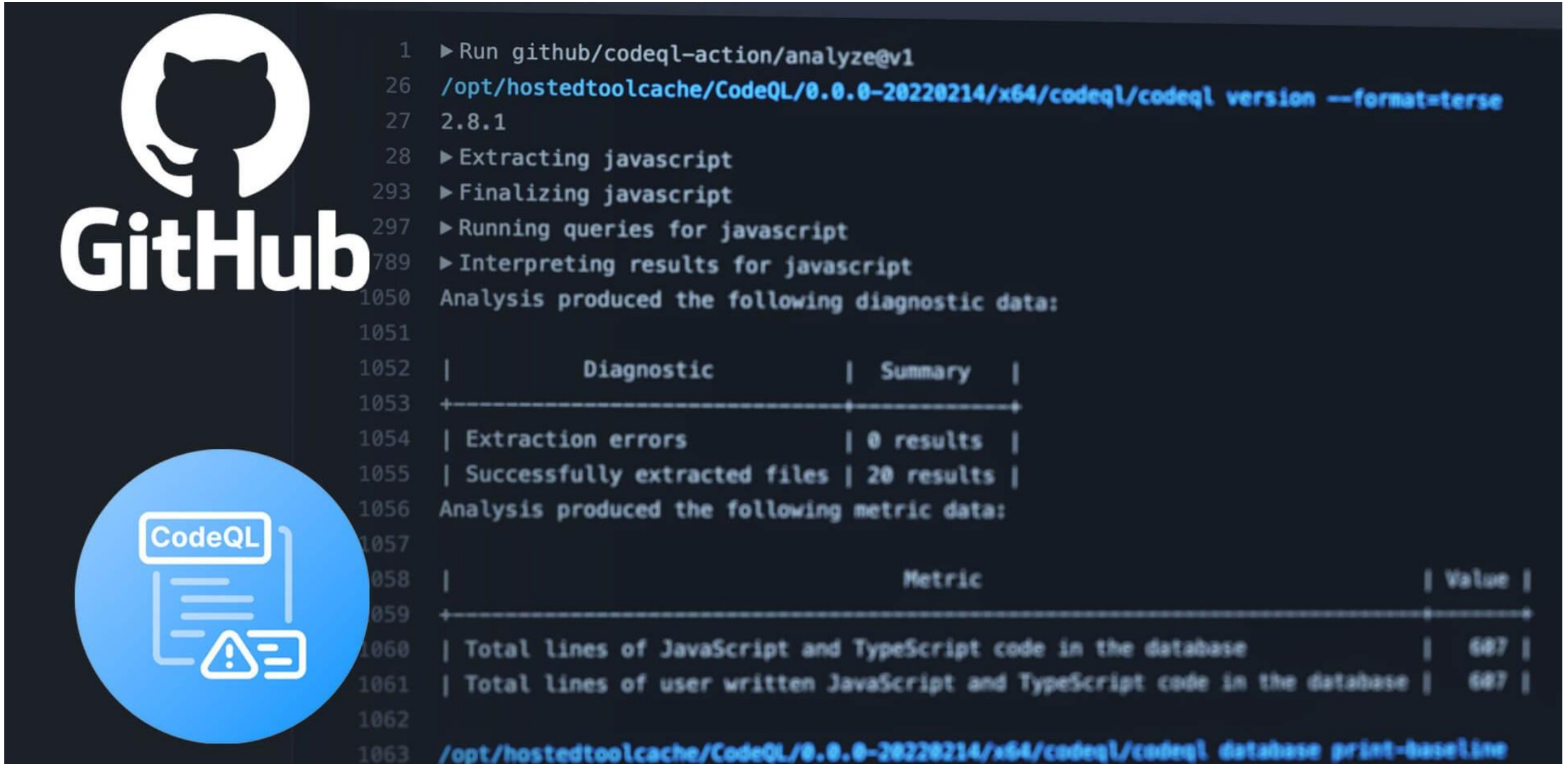
EVERYWHERE

SAST

*SAST – Static Application Security Testing

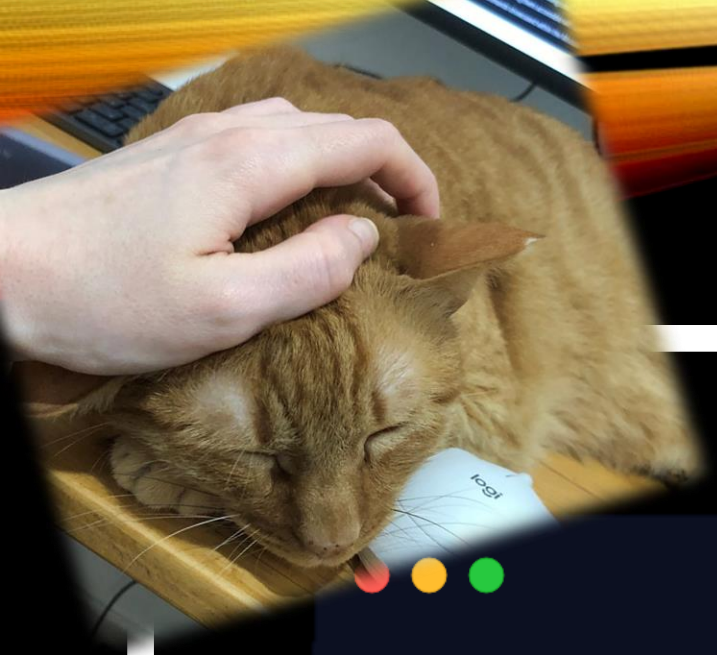
EVERYWHERE

Bounce SECURITY



EVERYWHERE

Bounce SECURITY



```
/^([a-z\d\•–]+)@([a-z\d-]+)\.([a-z]{2,8})(\.[a-z]{2,8})?$/
```

EVERYWHERE

Bounce SECURITY

BUT WHAT DOES THAT MEAN?

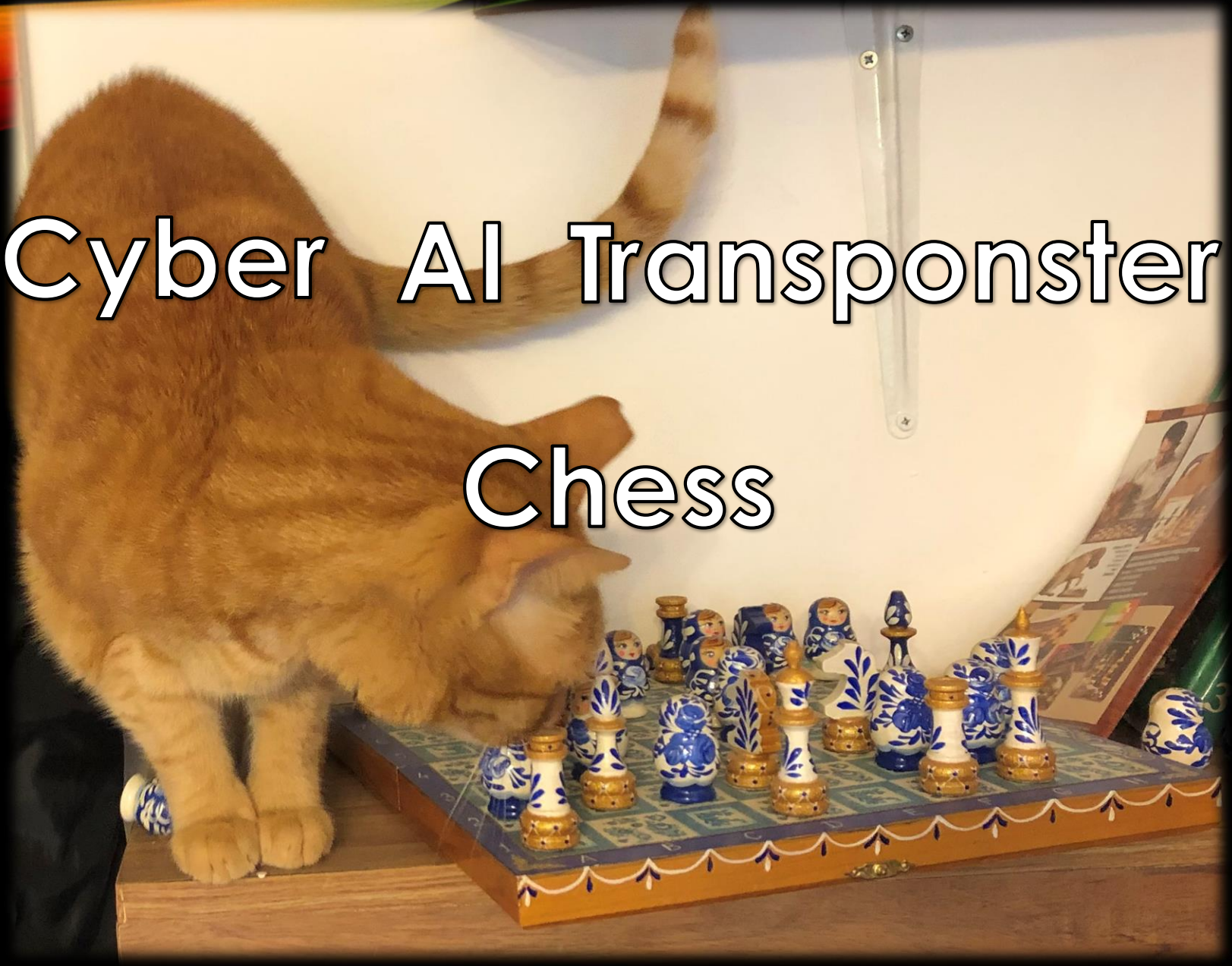
We want to enforce:

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

Bounce SECURITY

Cyber AI Transponster

Chess



Bounce SECURITY

EXAMPLES

<u>4 examples</u>:

1. Wrong Overload
2. Security Decorators
3. Unsanitised Input
4. Vulnerable Library

Bounce SECURITY

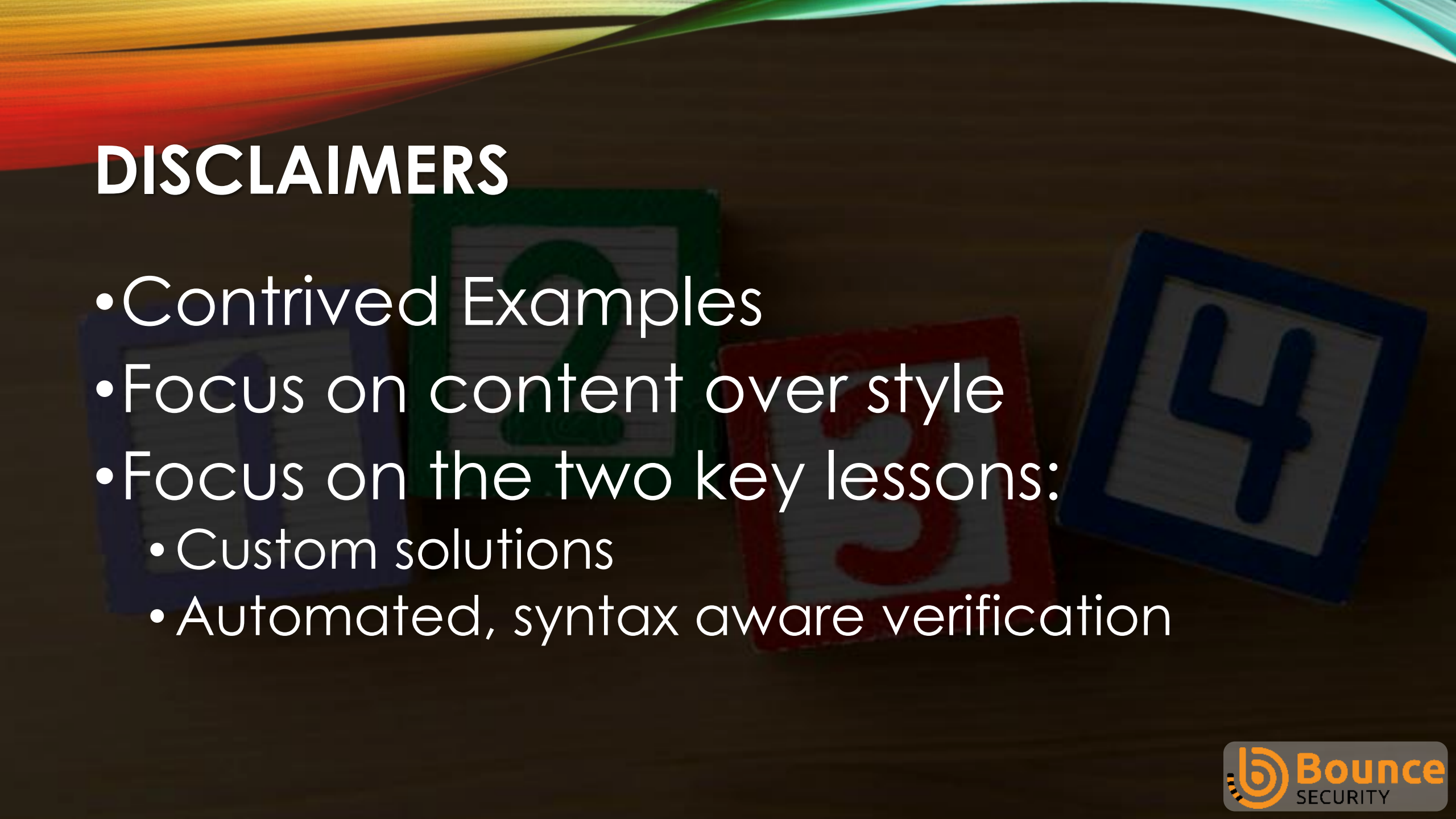
EXAMPLES

Each example:

- Has a security impact
- Could easily lead to becoming vulnerable again
- Is specific to our organization
- Needs something cleverer than Ctrl-F

DISCLAIMERS

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



WRONG OVERLOAD

The problem we're addressing:

- Sending money transactions
- Custom function to carry this out

```
class CallService:
    def send_transaction(self, cat, value, msg):
        # Do transaction sending stuff
        print("Sent to '{0}'".format(cat))
```

WRONG OVERLOAD


```
class CallService:
    def send_transaction(self, cat, value, msg):
        # Do transaction sending stuff
        print("Sent to '{0}'".format(cat))
```

- Vulnerable to replay attack (no transaction signing)
- Cannot eliminate the current functionality
- Cannot change function name

WRONG OVERLOAD

Proposed security solution:

- Create “overloaded” function
- Add an optional signature to authenticate

WRONG OVERLOAD - CODE EXAMPLE

```
class CallService:
    def send_transaction_old(self, cat, value, msg):
        # Do transaction sending stuff
        print("Sent to '{0}'".format(cat))

    def send_transaction(self, cat, value, msg, checksum=None):
        if checksum == None:
            self.send_transaction_old(cat, value, msg)
        else:
            # Do transaction sending stuff but this time
            # with a checksum
            print("Sent to '{0}', but safely!".format(cat))
```

WRONG OVERLOAD - CODE EXAMPLE

```
class CallService:
    def send_transaction_old(self, cat, value, msg):
        # Do transaction sending stuff
        print("Sent to '{0}'".format(cat))

    def send_transaction(self, cat, value, msg, checksum=None):
        if checksum == None:
            self.send_transaction_old(cat, value, msg)
        else:
            # Do transaction sending stuff but this time
            # with a checksum
            print("Sent to '{0}', but safely!".format(cat))
```

WRONG OVERLOAD - CODE EXAMPLE

```
class CallService:
    def send_transaction_old(self, cat, value, msg):
        # Do transaction sending stuff
        print("Sent to '{0}'".format(cat))

    def send_transaction(self, cat, value, msg, checksum=None):
        if checksum == None:
            self.send_transaction_old(cat, value, msg)
        else:
            # Do transaction sending stuff but this time
            # with a checksum
            print("Sent to '{0}', but safely!".format(cat))
```

WRONG OVERLOAD

Proposed security solution:

- Create “overloaded” function
- Add an optional signature to authenticate

Challenge:

- How do we ensure the old version is not being used?

Bounce SECURITY

WRONG OVERLOAD - CODE EXAMPLE

```
28  def main():
29
30      CatRequest = CallService()
31      CatRequest.name = "HacmeBank"
32
33      cat_num = "12146123"
34      msg = "Invoice 24234"
35      some_checksum = "D97EA6CA"
36
37      # ruleid: py_ex1_wrong_overload
38      CatRequest.send_transaction(cat_num, 4489, msg)
39
40      # ok: py_ex1_wrong_overload
41      CatRequest.send_transaction(cat_num, 4489, msg, some_checksum)
```

WRONG OVERLOAD - CODE EXAMPLE

```
28  def main():
29
30      CatRequest = CallService()
31      CatRequest.name = "HacmeBank"
32
33      cat_num = "12146123"
34      msg = "Invoice 24234"
35      some_checksum = "D97EA6CA"
36
37      # ruleid: py_ex1_wrong_overload
38      CatRequest.send_transaction(cat_num, 4489, msg)
39
40      # ok: py_ex1_wrong_overload
41      CatRequest.send_transaction(cat_num, 4489, msg, some_checksum)
```

Bounce SECURITY

WRONG OVERLOAD

Why do we need custom verification:

- Specific to our company
- Understanding of function parameters required

```
28  def main():
29
30      CatRequest = CallService()
31      CatRequest.name = "HacmeBank"
32
33      cat_num = "12146123"
34      msg = "Invoice 24234"
35      some_checksum = "D97EA6CA"
36
37      # ruleid: py_ex1_wrong_overload
38      CatRequest.send_transaction(cat_num, 4489, msg)
39
40      # ok: py_ex1_wrong_overload
41      CatRequest.send_transaction(cat_num, 4489, msg, some_checksum)
```

Bounce SECURITY

WRONG OVERLOAD - RULE

```yaml
rules:
  - id: py_ex1_wrong_overload
    pattern: $CLASS.send_transaction($PARAM1, $PARAM2, $PARAM3)
    message: The wrong overload of send_transaction has been
    used on this line.
    languages:
      - python
    severity: ERROR

```

Bounce SECURITY

WRONG OVERLOAD

```
28  def main():
29
30      CatRequest = CallService()
31      CatRequest.name = "HacmeBank"
32
33      cat_num = "12146123"
34      msg = "Invoice 24234"
35      some_checksum = "D97EA6CA"
36
37      # ruleid: py_ex1_wrong_overload
38      CatRequest.send_transaction(cat_num, 4489, msg)
39
40      # ok: py_ex1_wrong_overload
41      CatRequest.send_transaction(cat_num, 4489, msg, some_checksum)
```

Bounce SECURITY

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
- Don't want complex logic inside each function

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 {0}".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 {0}".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 {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)
```

Bounce SECURITY

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)
```

Bounce SECURITY

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)
```

Bounce SECURITY

SECURITY DECORATORS

Why do we need custom verification:

- Understanding of decorators/ functions is required

```
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)
```

Bounce SECURITY

SECURITY DECORATORS - RULE

```yaml
rules:
  - id: py_ex2_auth_decorators
    patterns:
      - pattern: |
          def $FUNC(...):
            ...
      - focus-metavariable:
          - $FUNC
      - pattern-not-inside: |
          @require_update_purrrmission
          def $FUNC(...):
            ...
      - pattern-inside: |
          class SensitiveFunctions:
            ...
    message: Function is missing the authorization decorator
    languages:
      - python
    severity: ERROR

```

Bounce SECURITY

SECURITY DECORATORS

```
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)
```

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 {0}".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 {0}".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)
```

Bounce SECURITY

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 {0}".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)
```

Bounce SECURITY

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 {0}".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)
```

Bounce SECURITY

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?

Bounce SECURITY

DANGEROUS FUNCTION - CODE EXAMPLE

```
15  new_address = get_input()
16
17  #ruleid: py_ex3_unsanitized_input
18  run_ping(new_address)
19
20  new_address = sanitize_it(new_address)
21
22  #ok: py_ex3_unsanitized_input
23  run_ping(new_address)
24
```

Bounce SECURITY

DANGEROUS FUNCTION - CODE EXAMPLE

```
15  new_address = get_input()
16
17  #ruleid: py_ex3_unsanitized_input
18  run_ping(new_address)
19
20  new_address = sanitize_it(new_address)
21
22  #ok: py_ex3_unsanitized_input
23  run_ping(new_address)
24
```

Bounce SECURITY

DANGEROUS FUNCTION - CODE EXAMPLE

```
15  new_address = get_input()
16
17  #ruleid: py_ex3_unsanitized_input
18  run_ping(new_address)
19
20  new_address = sanitize_it(new_address)
21
22  #ok: py_ex3_unsanitized_input
23  run_ping(new_address)
24
```

Bounce SECURITY

DANGEROUS FUNCTION - CODE EXAMPLE

```
15  new_address = get_input()
16
17  #ruleid: py_ex3_unsanitized_input
18  run_ping(new_address)
19
20  new_address = sanitize_it(new_address)
21
22  #ok: py_ex3_unsanitized_input
23  run_ping(new_address)
24
```

Bounce SECURITY

DANGEROUS FUNCTION - CODE EXAMPLE

```
15  new_address = get_input()
16
17  #ruleid: py_ex3_unsanitized_input
18  run_ping(new_address)
19
20  new_address = sanitize_it(new_address)
21
22  #ok: py_ex3_unsanitized_input
23  run_ping(new_address)
24
```

Bounce SECURITY

DANGEROUS FUNCTION

Why do we need custom verification:

- Recognizing custom sanitizers

```
15  new_address = get_input()
16
17  #ruleid: py_ex3_unsanitized_input
18  run_ping(new_address)
19
20  new_address = sanitize_it(new_address)
21
22  #ok: py_ex3_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 {0}".format(address)
13     subprocess.Popen(cmd, shell=True)
14
```

DANGEROUS FUNCTION - RULE

```yaml
rules:
  - id: py_ex3_unsanitized_input
    mode: taint
    pattern-sources:
      - pattern: input(...)
    pattern-sinks:
      - pattern: run_ping(...)
    pattern-sanitizers:
      - pattern: sanitize_it(...)
    message: run_ping is being used without first sanitizing
    languages:
      - python
    severity: ERROR

```

Bounce SECURITY

DANGEROUS FUNCTION

```
15  new_address = get_input()
16
17  #ruleid: py_ex3_unsanitized_input
18  run_ping(new_address)
19
20  new_address = sanitize_it(new_address)
21
22  #ok: py_ex3_unsanitized_input
23  run_ping(new_address)
24
```

VULNERABLE LIBRARY

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

Bounce SECURITY

VULNERABLE LIBRARY

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

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

Bounce SECURITY

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)
```

Bounce SECURITY

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)
```

Bounce SECURITY

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)
```

Bounce SECURITY

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

Bounce SECURITY

VULNERABLE LIBRARY - RULE

```yaml
rules:
  - id: py_ex4_vulnerable_library
    mode: taint
    pattern-sources:
      - pattern: py7zr.$FUNC(...)
    pattern-sinks:
      - pattern: $O.extractall(...)
    message: Semgrep found a match
    languages:
      - python
    severity: ERROR

```

VULNERABLE LIBRARY

```
 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)
```

Bounce SECURITY

SEMGREP LIMITATIONS FOR PYTHON

- No cross-file scanning (for now)
- No support for types (for now)

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

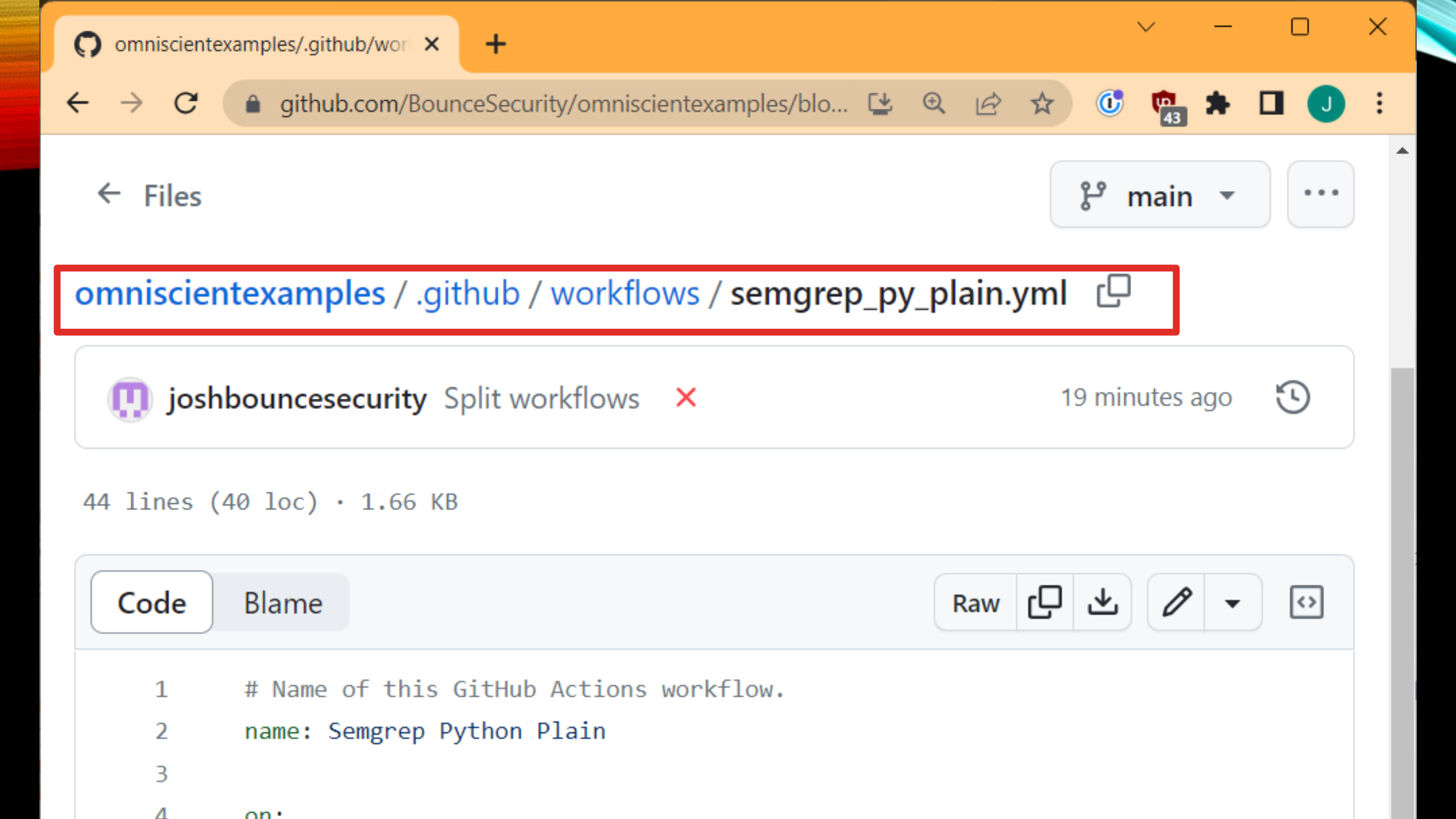
Bounce SECURITY

SO, HOW DO I DO THIS ALL THE TIME?

- Let's see a demo!

ALL THE TIME

Bounce SECURITY


omniscientexamples/.github/wor
github.com/BounceSecurity/omniscientexamples/blo...
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:

```yaml
on:
  # Scan changed files in PRs (diff-aware scanning):
  #pull_request: {}
  # Scan on-demand through GitHub Actions interface:
  workflow_dispatch: {}
  # Scan mainline branches and report all findings:
  push:
    branches: ["master", "main"]
  # Schedule the CI job (this method uses cron syntax):
  #schedule:
  #  - cron: '20 13 * * *' # Sets Semgrep to scan every day at 17:20 UTC.
  #  # It is recommended to change the schedule to a random time.

jobs:
  semgrep:
    # User-definable name of this GitHub Actions job:
    name: semgrep/ci
    # If you are self-hosting, change the following `runs-on` value:
    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            env:
39              # Add the rules that Semgrep uses by setting the SEMGREP_RULES enviro
40              SEMGREP_RULES: >
41                semgrep_rules/py_ex1_wrong_overload.yml
42                semgrep_rules/py_ex2_auth_decorators.yml
43                semgrep_rules/py_ex3_unsanitized_input.yml
44                semgrep_rules/py_ex4_vulnerable_library.yml
```

DEMO

- Results in Workflow log

ALL THE TIME

Split workflows · BounceSecurity/

github.com/BounceSecurity/omniscientexamples/actions/runs/54351970...

semgrep/ci

failed 27 minutes ago in 20s

Run semgrep --pro --error --no-rewrite-rule-ids 1s

```
26
27    code/python/ex1_wrong_overload.py
28       py_ex1_wrong_overload
29          The wrong overload of send_transaction has been used on this line.
30
31             38┆ CatRequest.send_transaction(cat_num, 4489, msg)
32              ⋮┆----------------------------------------
33             44┆ CatRequest.send_transaction("12146123", 4489, msg)
34              ⋮┆----------------------------------------
35             47┆ CatRequest.send_transaction(cat_num, 4342, msg)
36
37    code/python/ex2_auth_decorators.py
38       py_ex2_auth_decorators
39          Function is missing the authorization decorator
40
41             31┆ def next_move_cat(cat, move):
```

DEMO

- Results uploaded to GitHub

ALL THE TIME

Split workflows · BounceSecurity/

github.com/BounceSecurity/omniscientexamples/actions/runs/54351970...

semgrep/ci

failed 33 minutes ago in 29s

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

```
29  Scan was limited to files tracked by git.
30
31  Ran 4 rules on 4 files: 7 findings.
32  Error: Process completed with exit code 1.
```

Upload SARIF file for GitHub Advanced Security Dashboard 6s

```
1   ▶ Run github/codeql-action/upload-sarif@v2
10  /usr/bin/docker exec  f06586f5148f3056d9257266a10da5eb214be8c1bb1745b6f372f958cf791a56 sh -c "cat
    /etc/*release | grep ^ID"
11  ▶ Uploading results
15  ▶ Waiting for processing to finish
```

Post Run actions/checkout@v3 0s

Stop containers 0s

DEMO

- Results uploaded to GitHub

ALL THE TIME

Security Overview · BounceSecuri

github.com/BounceSecurity/omniscientexamples/security

BounceSecurity / omniscientexamples

Code | Pull requests | Actions | Security 7 | Insights | Settings

Overview

Reporting

Policy

Advisories

Vulnerability alerts

Dependabot

Code scanning 7

Secret scanning

Security overview

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
Enable vulnerability reporting

DEMO

- Results uploaded to GitHub

ALL THE TIME

Code scanning alerts · BounceSec

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_ex1_wrong_overload Error
#51 opened 5 days ago • Detected by **Semgrep** in **code/python/ex1_wrong_overload.py**:47 — main

Semgrep Finding: py_ex4_vulnerable_library Error
#50 opened 5 days ago • Detected by **Semgrep** in **code/python/ex4_vulnerable_library.p...**:13 — main

Semgrep Finding: py_ex4_vulnerable_library Error
#49 opened 5 days ago • Detected by **Semgrep** in **code/python/ex4_vulnerable_library.p...**:9 — main

Semgrep Finding: py_ex3_unsanitized_input Error — main

DEMO

- Results uploaded to GitHub

ALL THE TIME

Semgrep Finding: py_ex1_wrong_

github.com/BounceSecurity/omniscientexamples/security/code-scanning/51

Code scanning alerts / #51

Semgrep Finding: py_ex1_wrong_overload

Dismiss alert

Open in main 46 minutes ago

code/python/**ex1_wrong_overload.py**:47

```
44      CatRequest.send_transaction("12146123", 4489, msg)
45
46      # ruleid: wrong_overload
47      CatRequest.send_transaction(cat_num, 4342, msg)
```

The wrong overload of send_transaction has been used on this line.

Semgrep

```
48
49      # ruleid: ok
50      CatRequest.send_transaction(cat_num, 4489, "Invoice 78896", some_che
```

Severity

Error

Affected branches

main 2

Tool

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

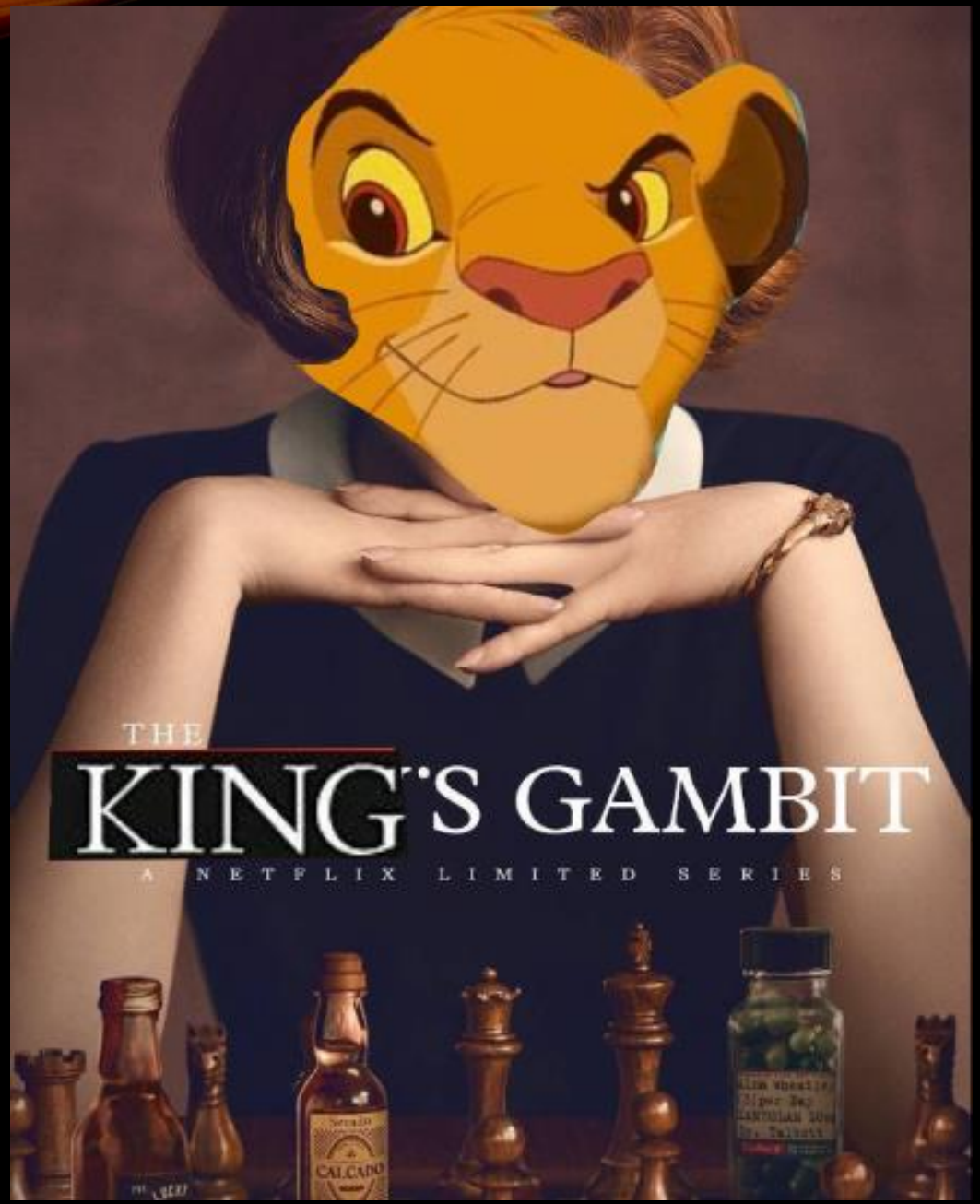
ALL THE TIME

Bounce SECURITY

KEY TAKEAWAYS

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

Bounce SECURITY



Credit: Emette Cohen Douglen

Bounce SECURITY

QUESTIONS?

- Everything
  - Vulnerabilities without generic solution
  - Some vulnerabilities are custom to us
- Everywhere
  - Simple but syntax-aware search rules
  - Scan the whole codebase or just relevant files
- All the time
  - Simple one-liner to run the scan
  - Easy to integrate and alert from regular CI

https://appsecg.host/pycon

Michal Kamensky
michal@bouncesecurity.com
@Kam3nskyM

Josh Grossman
josh@bouncesecurity.com
@JoshCGrossman

Bounce SECURITY