

# **Software detective**

**Python in security research**

**Przemek Samsel, 19.08.25**

# Whoami

- Przemek
- Tietoevry
- Network security





# Agenda

- [github.com/cheshie/code-analysis](https://github.com/cheshie/code-analysis)
- Security research
- Methodology
- Techniques
- Tools
- Case study





# Security Research

## What is it

- Understand digital threats & attacks
- Discover vulnerabilities
- Improve system defenses



# Security Research

## Why is it important

- CVE-2025-47273
- setuptools (pip) v78.1
- Impact: write to arbitrary files (with python permission)

```
def _download_url(self, url, tmpdir):  
    # Determine download filename  
    name, _fragment = egg_info_for_url(url)  
    if name:  
        while '..' in name:  
            name = name.replace('..',  
                                '.' ).replace('\\', '_')  
        else:  
            ...  
            filename = os.path.join(tmpdir, name)
```



# Security Research

## Why is it important

- <https://github.com/advisories>

Q type:reviewed ecosystem:pip severity:high

1,487 advisories

Severity ▾CWE ▾Sort ▾

Python-Future Module Arbitrary Code Execution via Unintended Import of test.py

High

CVE-2025-50817 was published for future (pip) 4 days ago

Keras vulnerable to CVE-2025-1550 bypass via reuse of internal functionality

High

CVE-2025-8747 was published for keras (pip) last week

Picklescan has pickle parsing logic flaw that leads to malicious pickle file bypass

High

GHSA-9gvj-pp9x-gcfr was published for picklescan (pip) last week

PyLoad vulnerable to SQL Injection via API /json/add\_package in add\_links parameter

High

CVE-2025-55156 was published for pyload-ng (pip) last week

GitHub reviewed advisories	
All reviewed	23,501
Composer	4,829
Erlang	36
GitHub Actions	33
Go	2,447
Maven	5,843
npm	4,065
NuGet	723
pip	3,866
Pub	12
RubyGems	943
Rust	1,010
Swift	39

# Security Research

## How to start

- **Build knowledge**

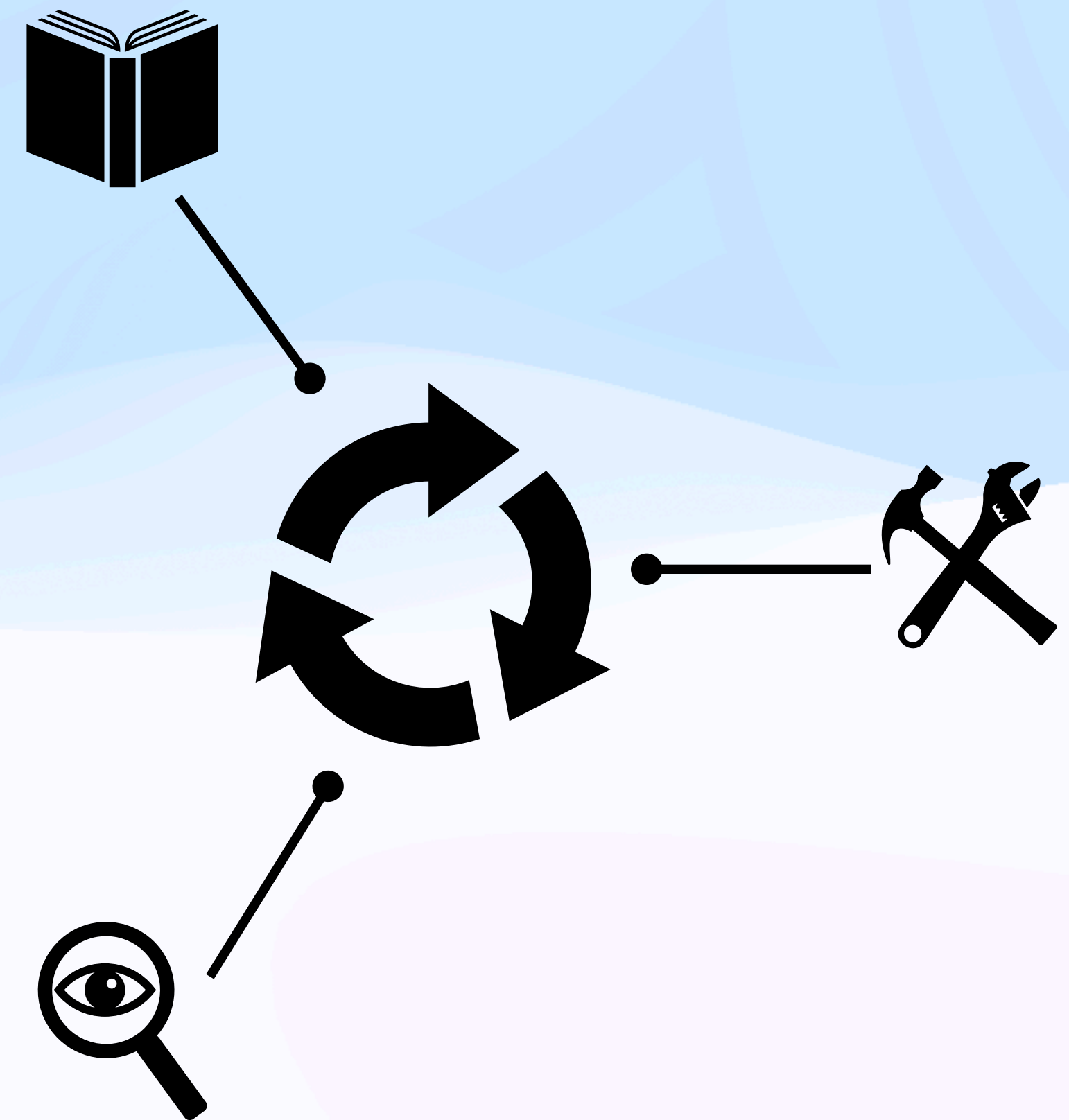
Read published research, experiment

- **Learn codebase**

Use software, understand code

- **Refine skills**

Solve challenges, identify patterns



# Methodology

## How to approach

- User interaction
- Threat model
- Attack scenarios

```
import ...

def list_products():
    ...

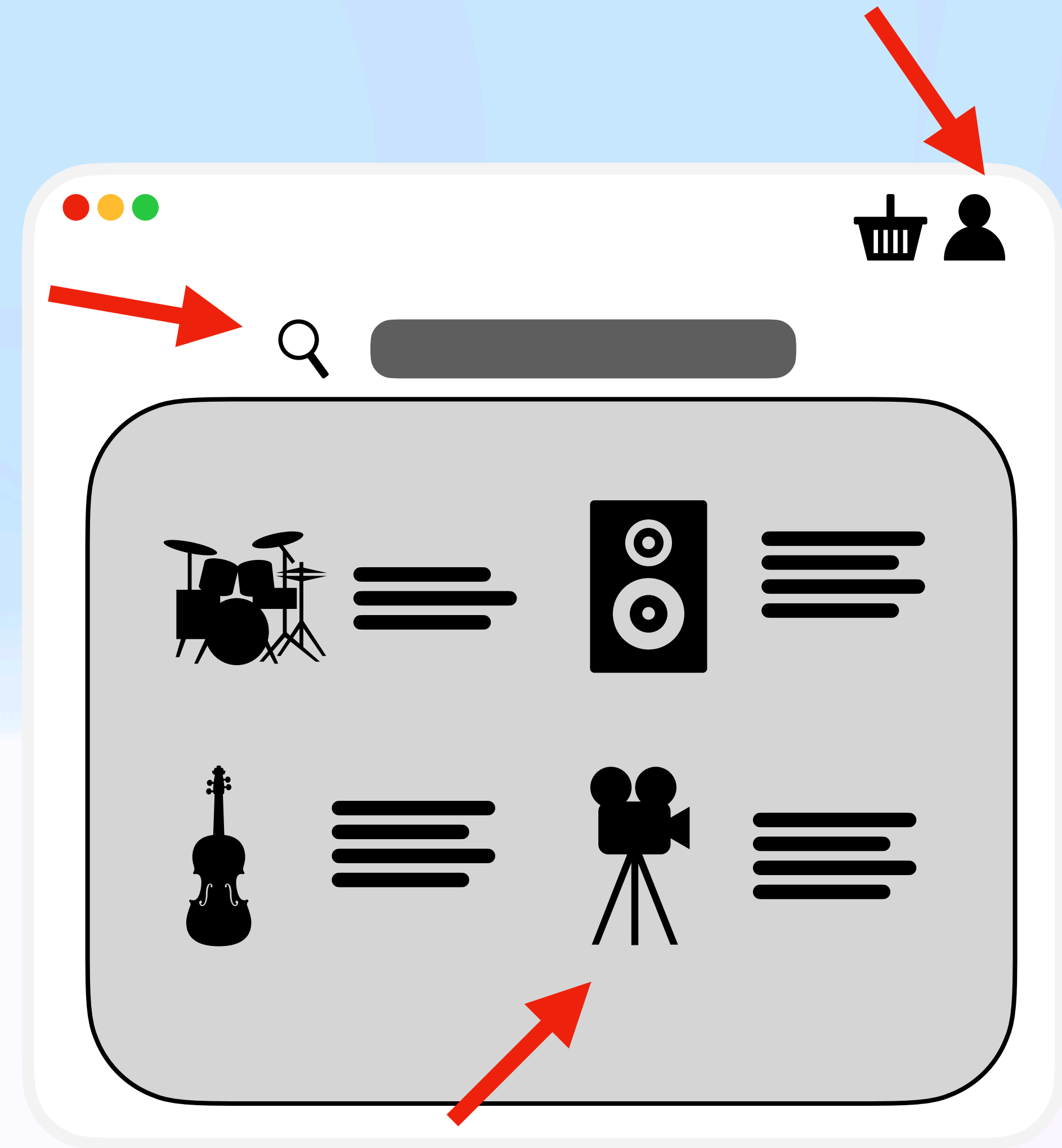
def add_to_basket():
    ...

def archive_product():
    ...
```



# Methodology

- **User interaction**
- Threat model
- Attack scenarios

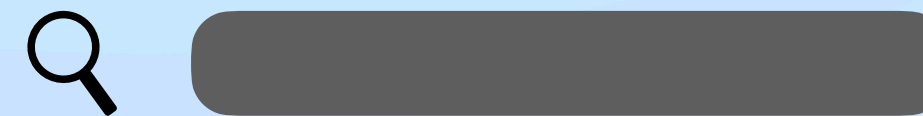


# Methodology

- User interaction
- **Threat model**
- Attack scenarios

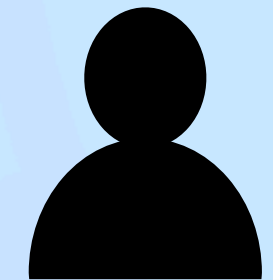
Can I...

list hidden products?



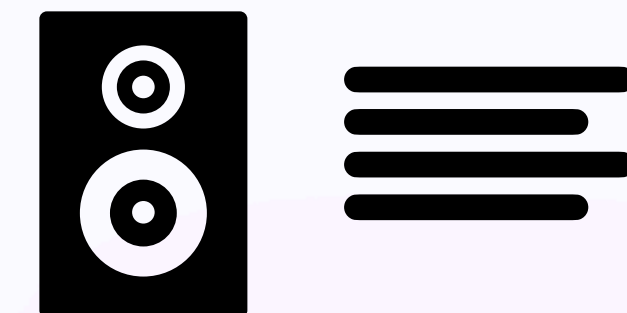
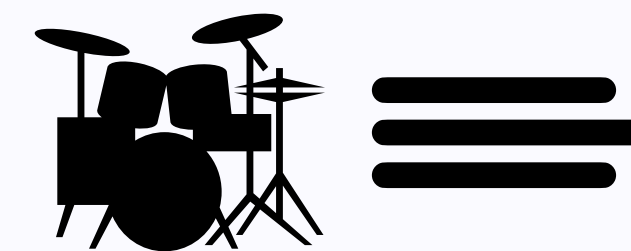
list other information?

Change my data  
without password?

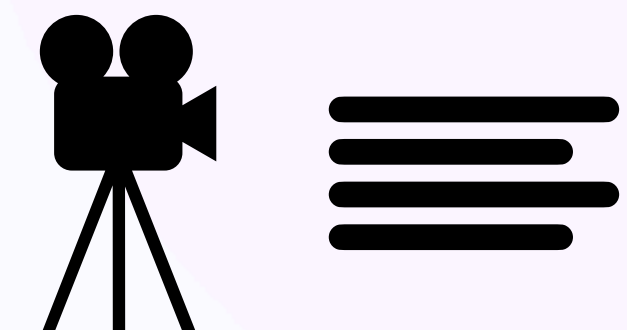
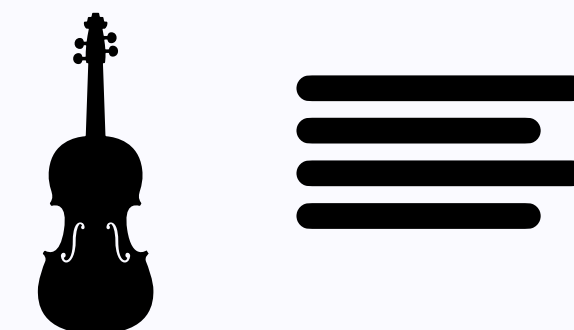


Access other users?

Modify products?



Change price?



Access basket of another user?



# Methodology

- User interaction
- Threat model
- **Attack scenarios**

## User account

Change email address ➡ Requires password?

Access user information ➡ Requires authorization?

Provide invalid login ➡ Descriptive error?

## Search products

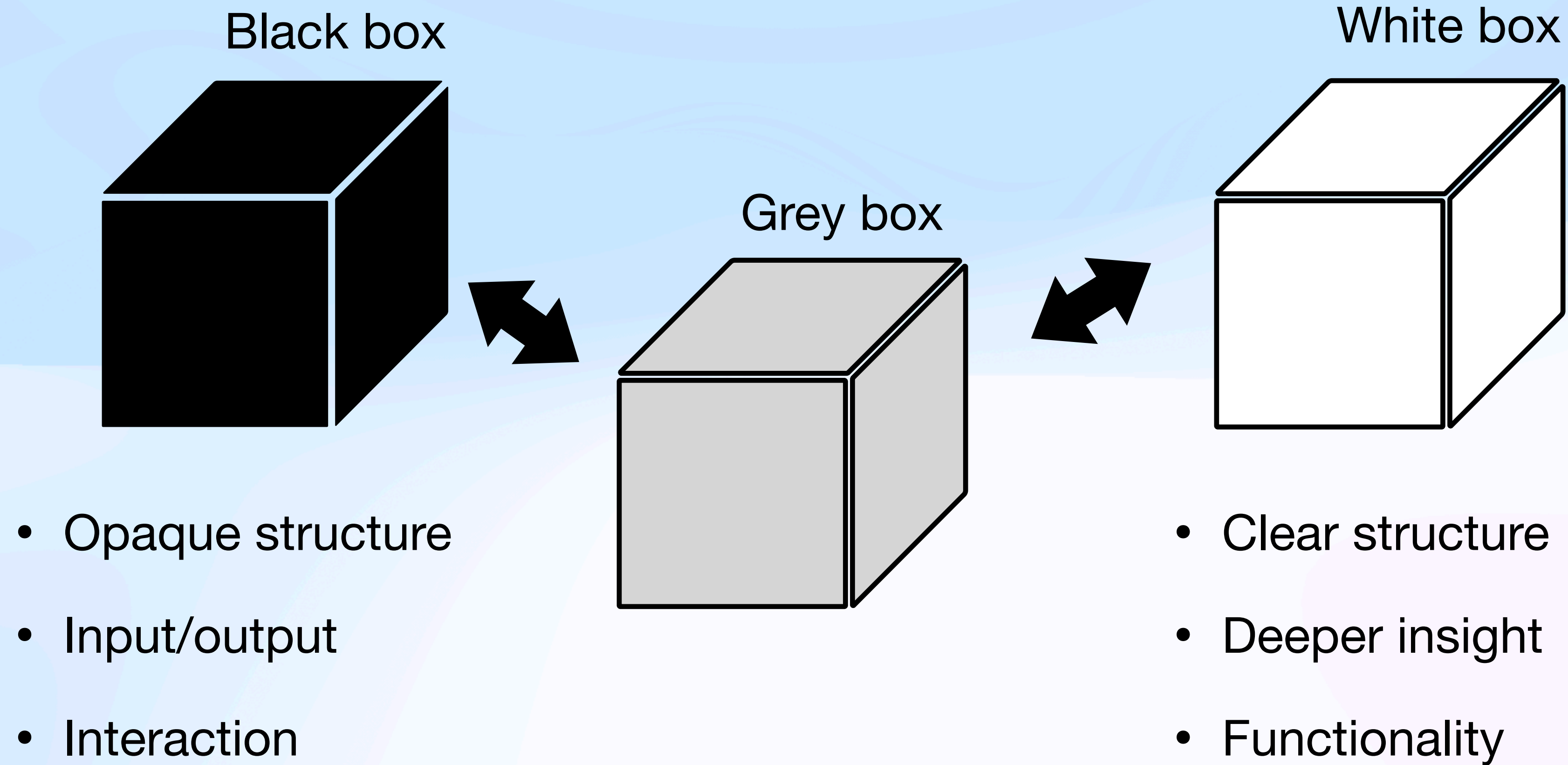
Use special characters ➡ Affects query results?

## Purchase products

Change price in basket ➡ Are changes reflected?

# Methodology

## Two perspectives

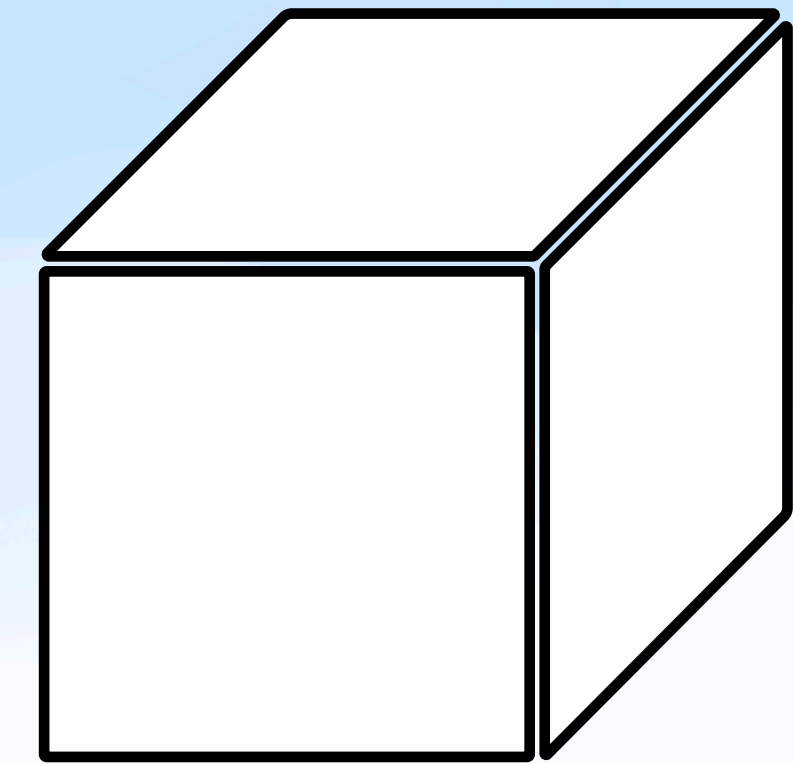




# Methodology

```
import ...  
  
def list_products():  
    ...  
  
def add_to_basket():  
    ...  
  
def archive_product():  
    ...
```

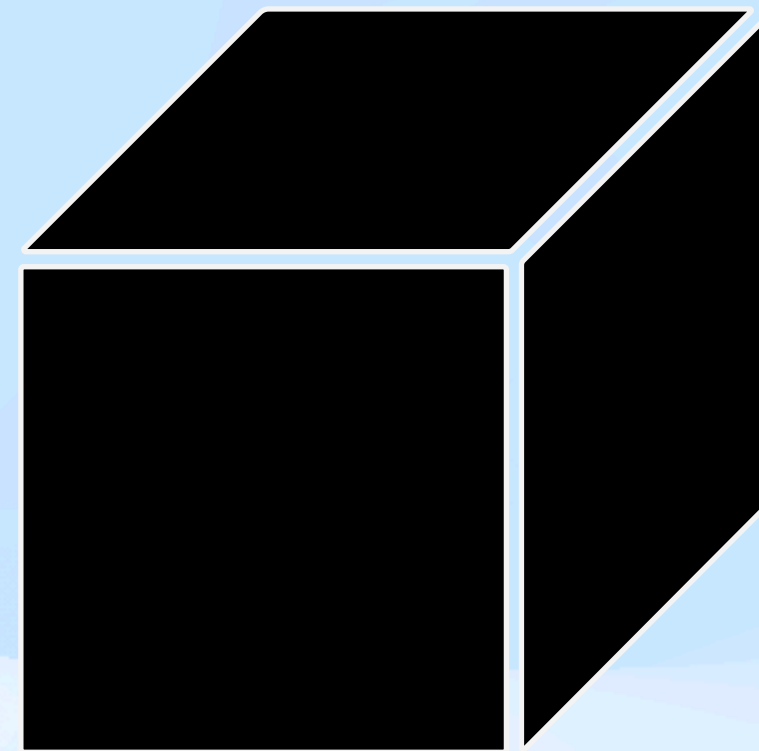
White box



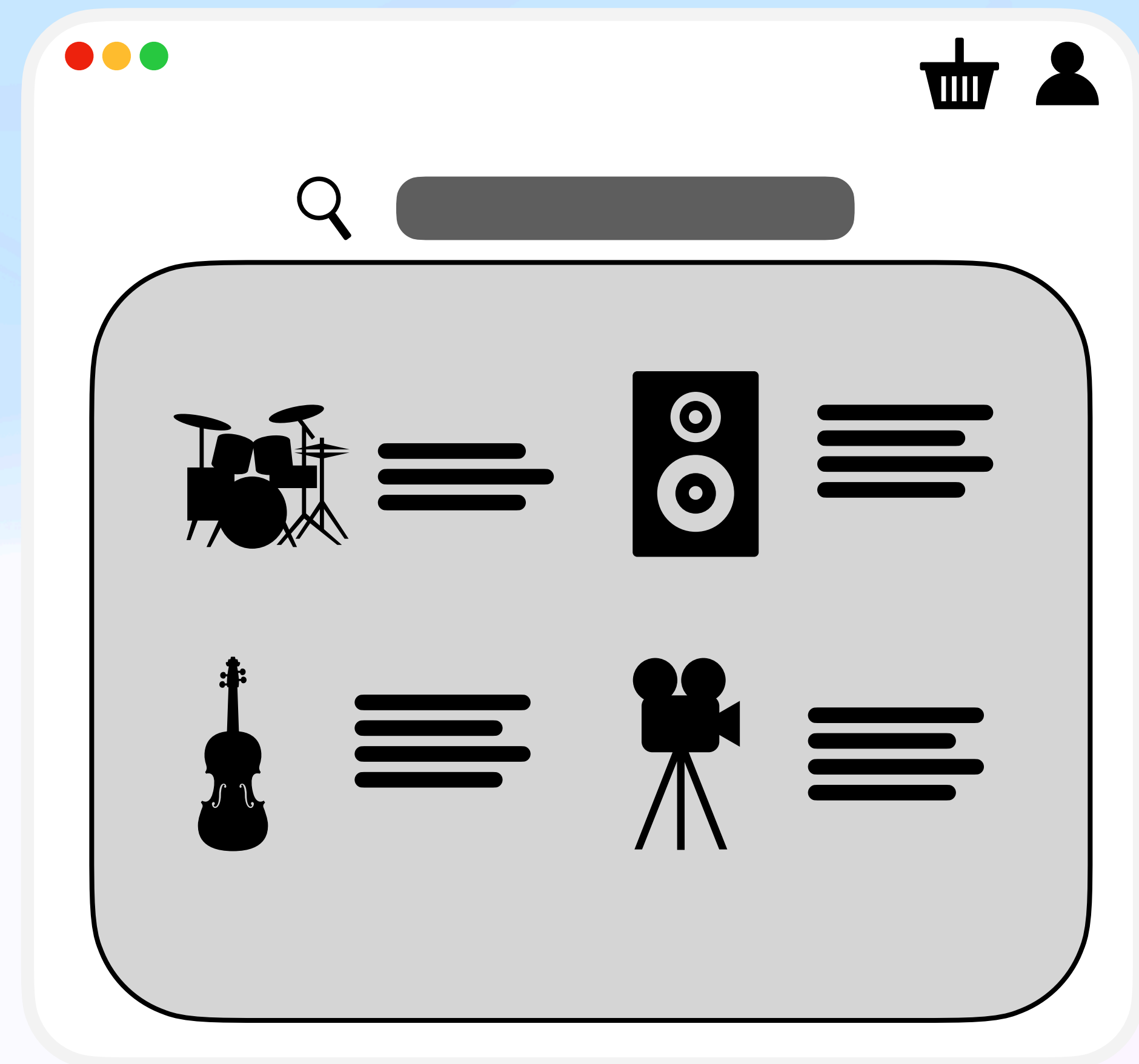
- Clear structure
- Deeper insight
- Functionality

# Methodology

Black box



- Opaque structure
- Input/output
- Interaction





# Techniques

What to use

## DAST

Dynamic Application Security Testing



## SAST

Static Application Security Testing



# Techniques

## DAST

Dynamic Application Security Testing



## Activities

- Behavioral analysis
- Fuzzing
- Instrumentation

## Tools

- BurpSuite
- Fuzzotron
- Frida



# Techniques

## Activities

- Code review
- Signature analysis
- Analyze Abstract Syntax Tree (AST)

## Tools

- CodeQL
- Bandit
- Semgrep
- Nuclei

## SAST

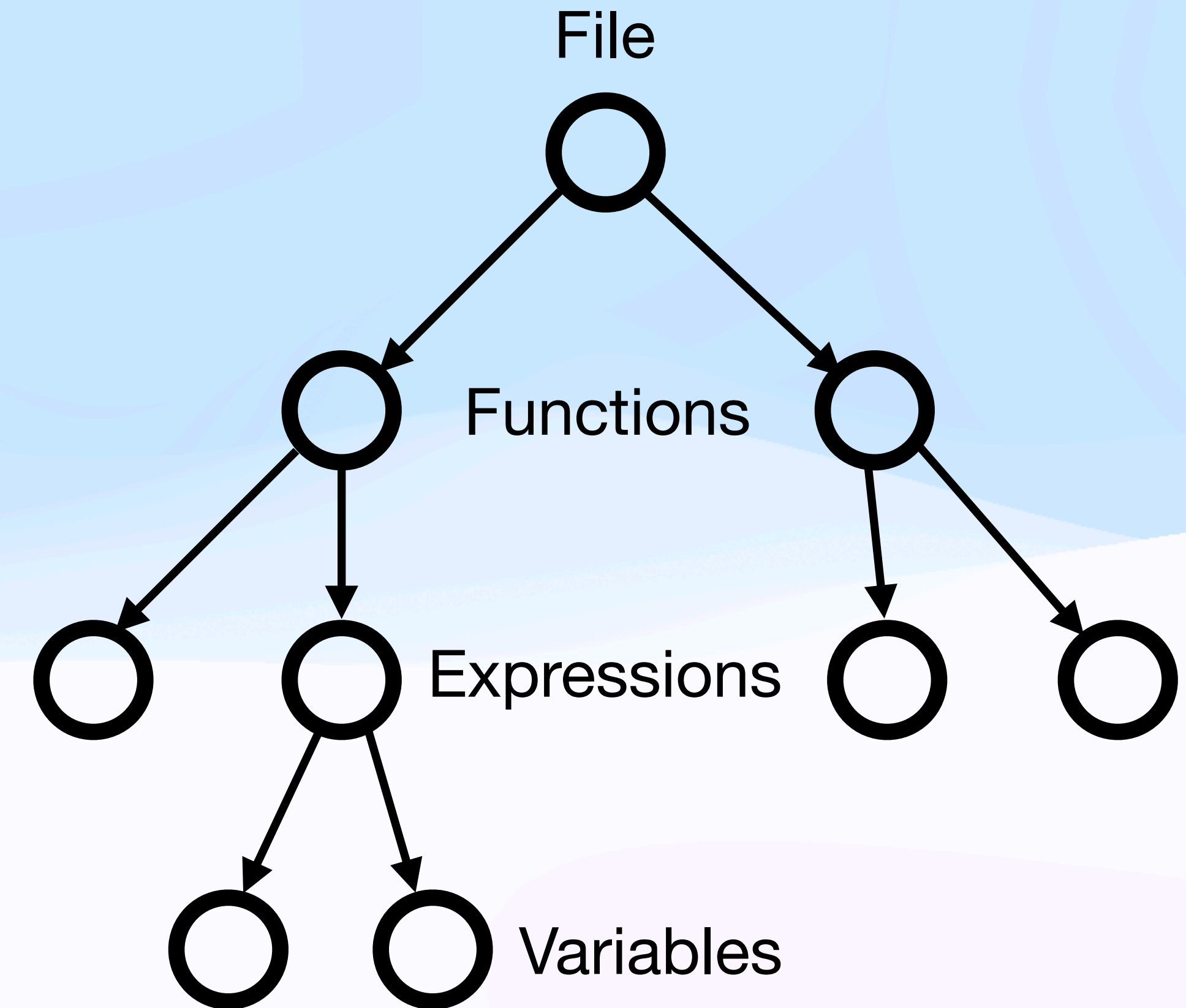
Static Application Security Testing



# Abstract Syntax Trees

## Basic information

- Data structure
- Graphical representation
- Abstract syntactic structure
- Used in compilers



# Abstract Syntax Trees

- Data structure
- Graphical representation
- Abstract syntactic structure
- Used in compilers

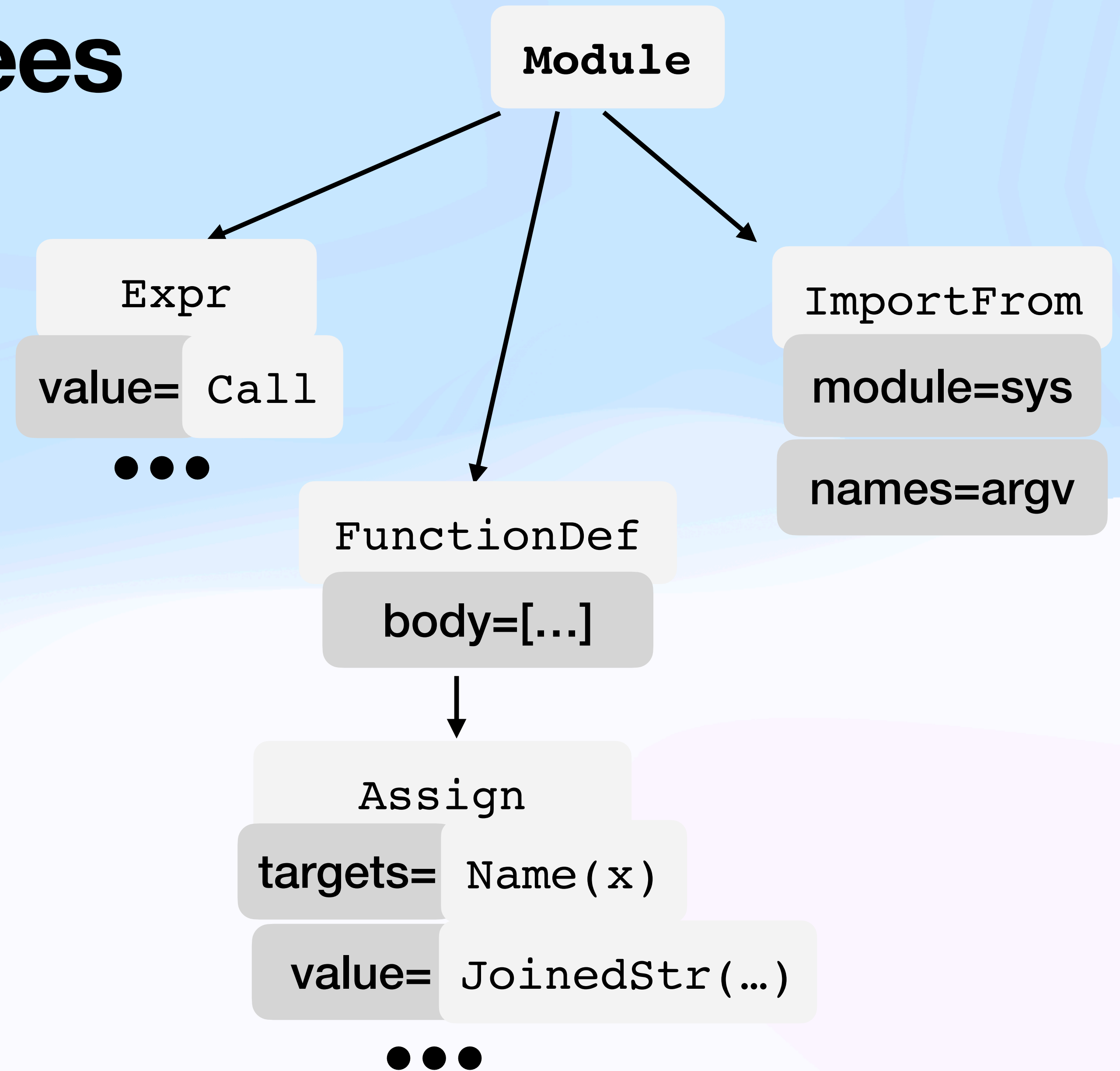
```
from ast import parse, dump
from sys import argv
from astprint import pprint

file = open(argv[1])
tree = parse(file.read())
print(pprint(tree))
```



# Abstract Syntax Trees

```
1 from sys import argv
2
3 def hello(name):
4     x = f"hello {name}!"
5     return x
6
7 print(hello(argv[1]))
```



# Abstract Syntax Trees

## Exercise

Exercises/ex1.py

```
1 from this import s
2
3 words = [x for x in s.split()]
4 print(len(words))
```

```
$ get_ast.py Exercises/ex1.py
```

# Bandit

## Overview

- Open-source security linter for Python
- Builds AST -> checks against set of defined plugins
- Active, occasionally updated, several open issues
- <http://bandit.readthedocs.io/>

### Install

```
$ python -m venv bvenv  
$ source bvenv/bin/activate  
(bvenv)$ pip install bandit
```



# Bandit

## Basic use

- List default plugins
- Run plugins for specific issue type
- Run specific plugin

```
$ bandit -r file.py
```

```
$ bandit -r file.py -s MEDIUM
```

```
$ bandit -r file.py -t B703
```

# Bandit

## Implementation

- Process target file -> build AST -> run set of plugins against nodes
- Uses a variant of the NodeVisitor ([bandit/core/node\\_visitor.py](#)) paradigm
- Reporting vulnerabilities, categorizing their impact and filtering ([bandit/core/issue.py](#))

# Bandit

## Plugins

- Rules are written with Python using the Bandit API
- Default - reports severity levels of **medium** and higher
- Use `# nosec` on line above specific issue to ignore it

Test results:

Issue: [`<ID>`] Use of weak MD5 for hashing.

Severity: Low

Confidence: High

Location: `example.py:5`

More Info: <https://bandit.readthedocs.io/plugins/...>

MD5 is a known broken hash algorithm. Avoid using it.



# Bandit

## Plugins

**Bandit informs user about vulnerabilities as follows**

- **File path** - Python file where the issue was detected
- **Line number** - where the issue is located
- **Test ID** - plugin unique identifier related to vulnerability type
- **Severity** - severity level of the issue (low, medium, high)
- **Confidence** - if issue is actually a security problem (low, medium, high)
- **Message** - brief description of the security issue

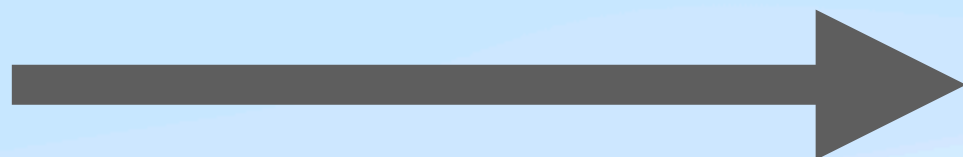
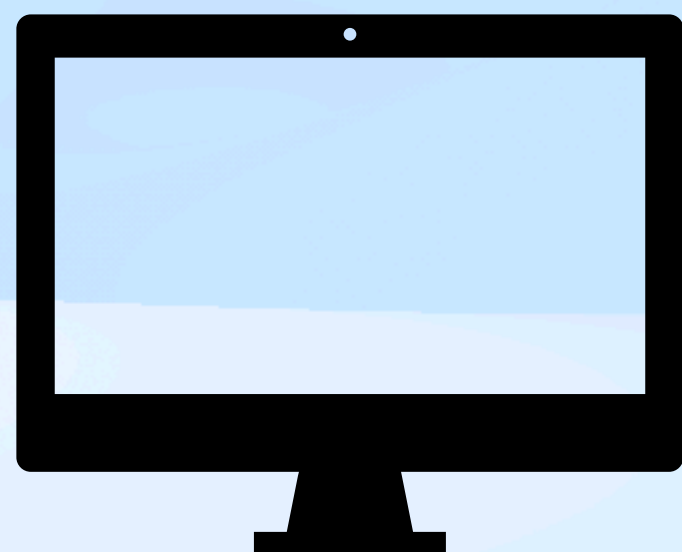
# SQL Injection



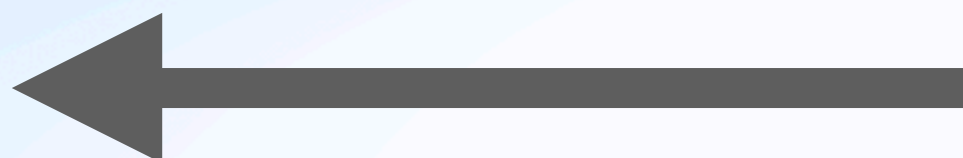
# SQL Injection

## Database query

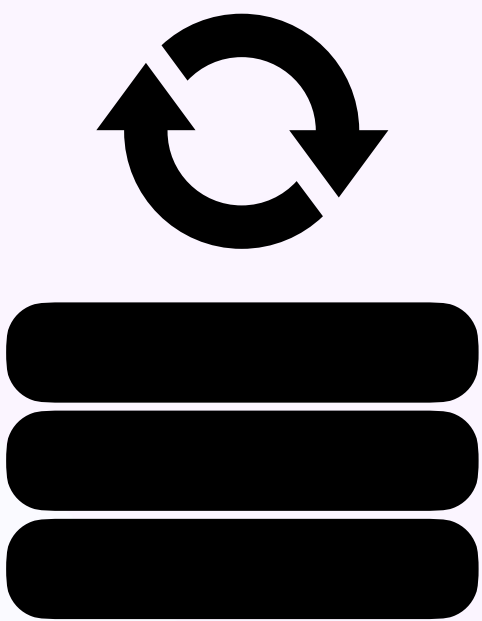
**SELECT \* FROM Products WHERE name = "Guitar"**



ID	Name	Category	Price
1	Guitar	Music	200
2	Bow	Sport	500
3	Desk	Furniture	800
...	...	...	...



1	Guitar	Music	200
---	--------	-------	-----

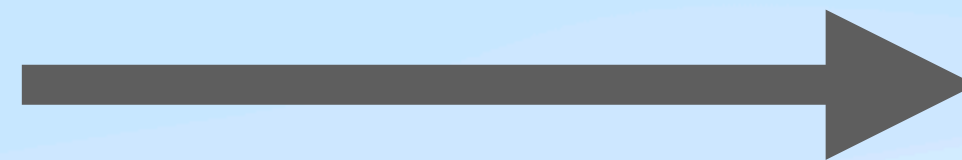




# SQL Injection

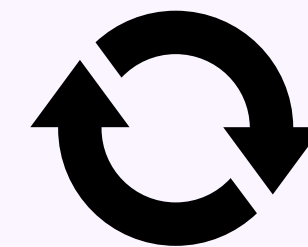
## Attack idea

**SELECT \* FROM Products WHERE name = "" OR 1=1;-- "**



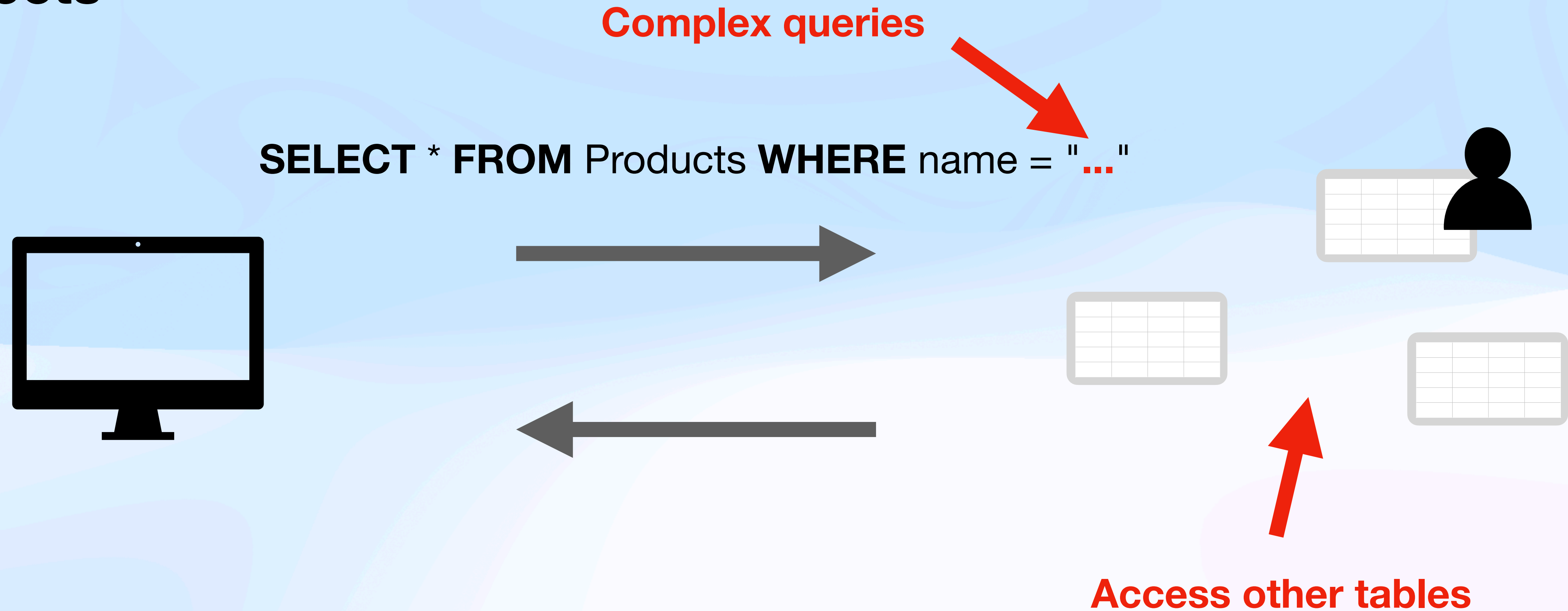
List of all products...

ID	Name	Category	Price
1	Guitar	Music	200
2	Bow	Sport	500
3	Desk	Furniture	800
...	...	...	...



# SQL Injection

## Effects



# SQL Injection

## Vulnerable implementation

```
@app.route( '/product' , methods=[ 'GET' ] )
def products():
    name = request.form[ 'name' ]
    conn = get_db_connection()
    products = conn.execute(
        "SELECT * FROM products WHERE product_name = '" + name + "'"
    ).fetchall()
    conn.close()
    ...
```



# Bandit Detection

1. Evaluate AST - find instances of string concatenation
2. Extract string statement, compare to regular expression
3. Check methods that use specific statement

# Bandit

## 1. Evaluate AST - find instances of string concatenation

```
def _evaluate_ast(node):  
    if isinstance(node._bandit_parent, ast.BinOp):  
        ...  
    elif isinstance(  
        node._bandit_parent, ast.Attribute  
    ) and node._bandit_parent.attr in ("format", "replace"):  
        ...  
    elif hasattr(ast, "JoinedStr") and isinstance(  
        node._bandit_parent, ast.JoinedStr  
    ):  
        ...
```

# Bandit

```
conn.execute("SELECT * FROM products WHERE product_name = '" + name + "'")
```

```
if isinstance(node._bandit_parent, ast.BinOp):  
    out = utils.concat_string(node, node._bandit_parent)  
    wrapper = out[0]._bandit_parent  
    statement = out[1]
```

<b>node</b>	➡	SELECT * FROM products WHERE product_name = '
<b>node-&gt;parent</b>	➡	ast.BinOp (ast.Add)
<b>wrapper</b>	➡	ast.Call (execute)
<b>statement</b>	➡	"SELECT * FROM products WHERE product_name = ' '"



# Bandit

## 2. Extract string statement, compare to regular expression

```
SIMPLE_SQL_RE = re.compile(  
    r"(select\s.*from\s|"   
    r"delete\s+from\s|"   
    r"insert\s+into\s.*values\s|"   
    r"update\s.*set\s)",  
    re.IGNORECASE | re.DOTALL,  
)  
  
def _check_string(data):  
    return SIMPLE_SQL_RE.search(data) is not None
```

# Bandit

## 3. Check methods that use specific statement

```
if isinstance(wrapper, Call):  
    names = ["execute", "executemany"]  
    name = utils.get_called_name(wrapper)  
    return (name in names, statement, str_replace)  
else:  
    return (False, statement, str_replace)
```

# Bandit

```
@test.checks("Str")
@test.test_id("B667")
def hardcoded_sql_expressions(context):
    execute_call, statement, str_replace = _evaluate_ast(context.node)
    if _check_string(statement):
        return bandit.Issue(
            severity=bandit.MEDIUM,
            confidence=(
                bandit.MEDIUM
                if execute_call and not str_replace
                else bandit.LOW
            ),
            cwe=Cwe.SQL_INJECTION,
            text="Possible SQL injection ...")
```



# Bandit Plugins

ID	Description	Count	Examples	CWE
B1xx	misc tests	13	hardcoded credentials, unsafe functions	<a href="#">CWE-703</a> , <a href="#">CWE-78</a> , <a href="#">CWE-732</a>
B2xx	misconfiguration	2	configuration, archive extraction	<a href="#">CWE-94</a> , <a href="#">CWE-22</a>
B3xx/B4xx	blacklists (calls/imports)	1	Insecure cryptography	<a href="#">CWE-327</a>
B5xx	cryptography	9	Insecure configuration, missing validations	<a href="#">CWE-295</a> , <a href="#">CWE-327</a> , <a href="#">CWE-326</a>
B6xx	injection	15	code injection	<a href="#">CWE-78</a> , <a href="#">CWE-89</a> , <a href="#">CWE-94</a>
B7xx	XSS	4	Cross-Site Scripting	<a href="#">CWE-94</a>

# Add own plugins



# Bandit

## Plugin structure

- **context** (bandit/core/context.py) - object of analysis
- **helper tools** (bandit/core/utils.py) - make evaluation of AST nodes
- **@checks** (bandit/core/test\_properties.py) - determine type of AST node

```
import ...

@test.checks("<AST TYPE>")
@test.test_id("B###")
def main_check(context):
    if ...:
        return Issue(
            severity=bandit.LOW,
            confidence=bandit.HIGH,
            cwe=Cwe.###,
            text=f"Vulnerable ...")
```



# Bandit

## Context

Main object of analysis

- Function properties (*name, arguments, keywords, named argument*)
- Import properties (*is it imported name? is it aliased?*)
- AST node
- Bytes representation
- File-related properties

# Bandit

## Python package

bandit\_plugin

└── bandit\_plugin

| └── init.py

| └── myplugin.py

└── setup.py

```
from setuptools import setup
```

```
setup(
    name='bandit_plugin_test',
    version='0.0.1',
    description='...',
    packages=['bandit_plugin'],
    author='',
    install_requires=['bandit'],
    entry_points={
        'bandit.plugins': [
            'os_getcwd =
bandit_plugin.myplugin:function',
        ],
    })
```

# Bandit

## Adding plugin

- Create plugin module
- Add test ID -> `@test_properties.test_id`
- Install local package
- Provide plugin ID to use it
- Modify installed package

```
$ python -m pip install ./bandit_plugin
```

```
$ bandit -t B### -f custom <file.py>
```

```
bvenv/lib/python3.13/site-packages/bandit/plugins/
```

# Exercises





# Bandit

## Future plans

- Extend plugin coverage
- Introduce taint tracking
- Implement VS Code plugin

# Static analysis tools

## Comparison

### **Bandit**

Open source

Python API

<50 plugins

### **Semgrep**

Open source

YAML

~350 rules

### **CodeQL**

Closed source

Free for research

~50 queries

[GitHub - Python CodeQL queries](#)

[Semgrep - Python security rules](#)

[Bandit - security plugins](#)



# Summary

- Security research aims to discover software vulnerabilities
- Whitebox - focus on code structure
- AST - code structure visualisation
- Static analysis provides helps to analyze relationships between code entities





# References

## Security research

- <https://portswigger.net/research/top-10-web-hacking-techniques-of-2024>
- <https://www.offsec.com/blog/>
- <https://blog.trailofbits.com/>
- <https://pagedout.institute/>

## Code analysis

- <https://github.blog/security/vulnerability-research/codeql-zero-to-hero-part-4-gradio-framework-case-study/>
- <https://github.blog/security/vulnerability-research/cybersecurity-researchers-digital-detectives-in-a-connected-world/>

## Security tutorials

- <https://tryhackme.com/careers/quiz/>
- <https://portswigger.net/web-security>
- <https://www.hackthebox.com/>



# Thank you