

## Testability Patterns for Web Applications – an OWASP project

Luca Compagna, SAP Security Research

24 November 2023

PUBLIC



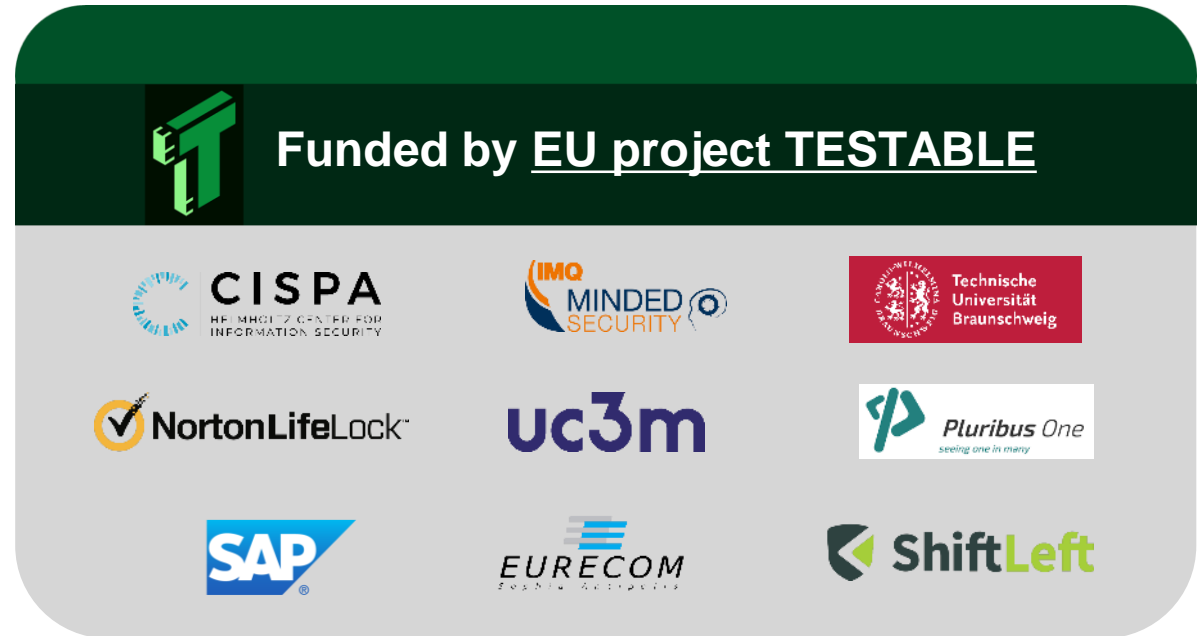
**TESTABLE**



# About me and others who contribute to this work



Luca Compagna  
**SAP** Security Research



<https://www.testable.eu/>

*joint work with:*

**Feras Al Kassar** (SAP), Giulia Clerici (SAP), Fabian Yamaguchi (SHIFTLEFT), Davide Balzarotti (EURECOM)

[1] Testability Tarpits: the Impact of Code Patterns on the Security Testing of Web Applications. Al Kassar et al. NDSS 2022

# An OWASP project: our journey started recently!

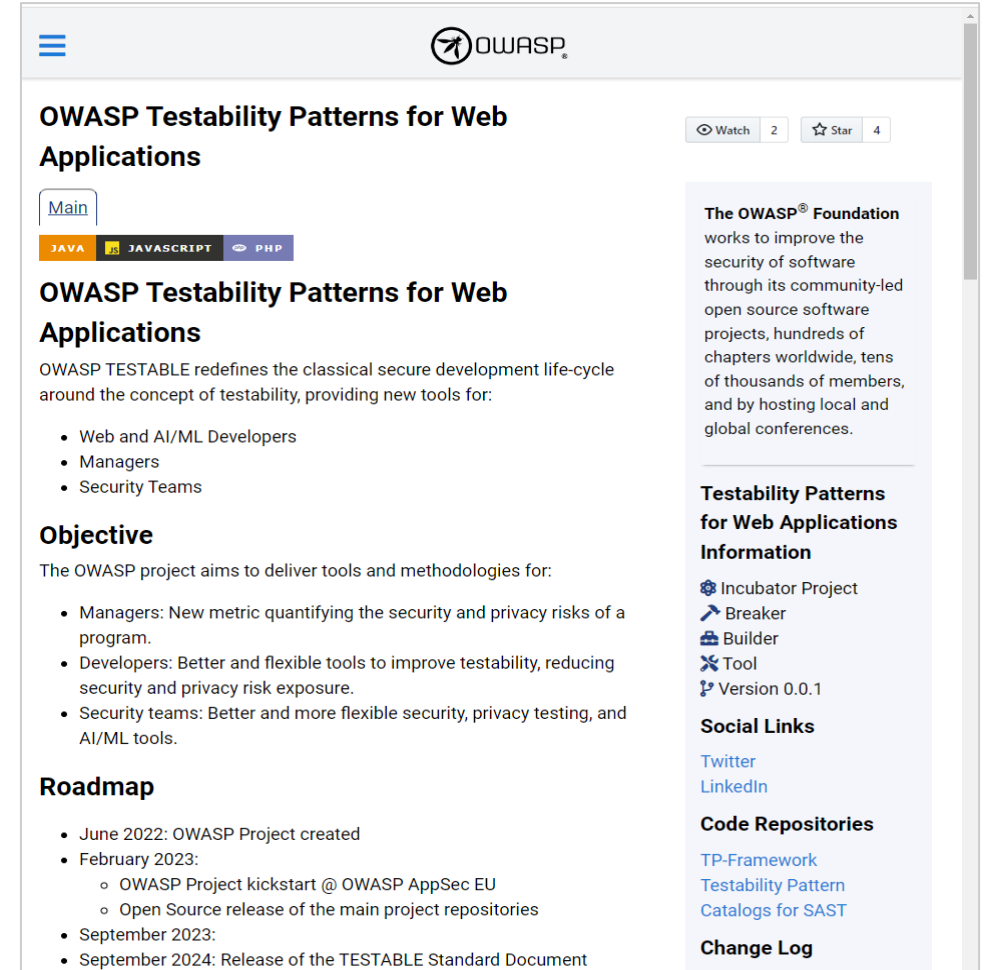
Security and privacy **testing** is core for protecting the Web  
**Many challenges** impact the **testing accuracy**!

Motivated by promising **research results**, we started a **new OWASP project** to precisely identify, measure and possibly remediate some of these challenges!

**Concrete results for SAST**

**Join** our project and our community to help achieving this vision!

<https://owasp.org/www-project-testability-patterns-for-web-applications/>



The screenshot displays the OWASP Testability Patterns for Web Applications project page. The header features the OWASP logo and navigation icons. The main title is "OWASP Testability Patterns for Web Applications", accompanied by "Watch" (2) and "Star" (4) buttons. Below the title, there's a "Main" tab and language filters for JAVA, JAVASCRIPT, and PHP. The page content includes a description of the project's goal to redefine the secure development life-cycle around testability, a list of target audiences (Web and AI/ML Developers, Managers, Security Teams), an objective to deliver tools and methodologies, and a roadmap with key milestones from June 2022 to September 2024. A right-hand sidebar provides additional context, including the OWASP Foundation's mission, project information (Incubator Project, Breaker, Builder, Tool, Version 0.0.1), social links (Twitter, LinkedIn), code repositories (TP-Framework, Testability Pattern, Catalogs for SAST), and a change log.

**OWASP Testability Patterns for Web Applications**

Watch 2 Star 4

Main

JAVA JAVASCRIPT PHP

**OWASP Testability Patterns for Web Applications**

OWASP TESTABLE redefines the classical secure development life-cycle around the concept of testability, providing new tools for:

- Web and AI/ML Developers
- Managers
- Security Teams

**Objective**

The OWASP project aims to deliver tools and methodologies for:

- Managers: New metric quantifying the security and privacy risks of a program.
- Developers: Better and flexible tools to improve testability, reducing security and privacy risk exposure.
- Security teams: Better and more flexible security, privacy testing, and AI/ML tools.

**Roadmap**

- June 2022: OWASP Project created
- February 2023:
  - OWASP Project kickstart @ OWASP AppSec EU
  - Open Source release of the main project repositories
- September 2023:
- September 2024: Release of the TESTABLE Standard Document

**The OWASP® Foundation** works to improve the security of software through its community-led open source software projects, hundreds of chapters worldwide, tens of thousands of members, and by hosting local and global conferences.

**Testability Patterns for Web Applications Information**

- Incubator Project
- Breaker
- Builder
- Tool
- Version 0.0.1

**Social Links**

- Twitter
- LinkedIn

**Code Repositories**

- TP-Framework
- Testability Pattern
- Catalogs for SAST

**Change Log**

# An OWASP project: our journey started recently!

Security and privacy **testing** is core for protecting the Web

**Many challenges** impact the **testing accuracy**!

Motivated by promising **research results**, we started a **new OWASP project** to precisely identify, measure and possibly remediate some of these challenges!

**Concrete results for SAST**

**Join** our project and our community to help achieving this vision!

<https://owasp.org/www-project-testability-patterns-for-web-applications/>



# Testability patterns for SAST

*(first concrete result of our OWASP project)*

# Testability patterns for SAST



chatGPT 3 (around 01/2023)



what is a testability pattern for static code analysis?



A testability pattern for static code analysis is a set of guidelines or best practices for designing code in a way that makes it easy to test using static analysis tools. This may include things like writing clear, modular code with well-defined interfaces, using dependency injection to decouple components, and following established coding conventions and design patterns. The goal is to make it easier to automatically identify and analyze code for potential bugs or other issues.



**2021 knowledge base!**

Technical paper by Feras Al Kassar et al. [Testability obstacles: the Impact of Code Patterns on the Security Testing of Web Applications](#). NDSS 2022



# Testability



chatGPT 3.5 (today)



**You**  
what is a testability pattern for static code analysis?



**ChatGPT**  
Testability patterns in static code analysis are design patterns that improve the testability of software through architectural principles. One common pattern is the "Dependency Injection" pattern.

Dependency Injection (DI) involves separating the creation of dependencies from the class that uses them. This allows the class to be tested in isolation by injecting mock dependencies.

For instance, if a class relies on an external service, you can inject a mock service or a fake data source. This enables better isolation of the code under test, allowing for more comprehensive and efficient testing processes.

By utilizing DI in static code analysis, you can identify areas where dependencies are tightly coupled or instantiated within classes, making it challenging to isolate components for testing. Refactoring code to follow DI principles can greatly enhance testability, leading to more comprehensive and efficient testing processes.



**You**  
are you aware of the work "Testability Obstacles: The Impact of Code Patterns on the Security Testing of Web Applications"?



**ChatGPT**  
Yes, that work explores the impact of code patterns on the security testing of web applications. It identifies several obstacles to security testing, such as tightly coupled dependencies, excessive complexity, and lack of modularity.

The study likely examines how these patterns can hinder the creation of meaningful test cases for targeted testing. It may also discuss strategies to mitigate these obstacles and improve the overall testability of web applications.

The goal of such research is to provide insights into the challenges faced by security testers and to propose strategies to address them. This can help in creating more robust and secure web applications by improving their testability.



**You**  
can you refine the definition of testability pattern for static code analysis as presented in that work?



**ChatGPT**  
In the context of the work "Testability Obstacles: The Impact of Code Patterns on the Security Testing of Web Applications," the definition of a testability pattern for static code analysis might encompass specific recurring code structures or design characteristics within web applications that impede effective security testing when analyzed statically.

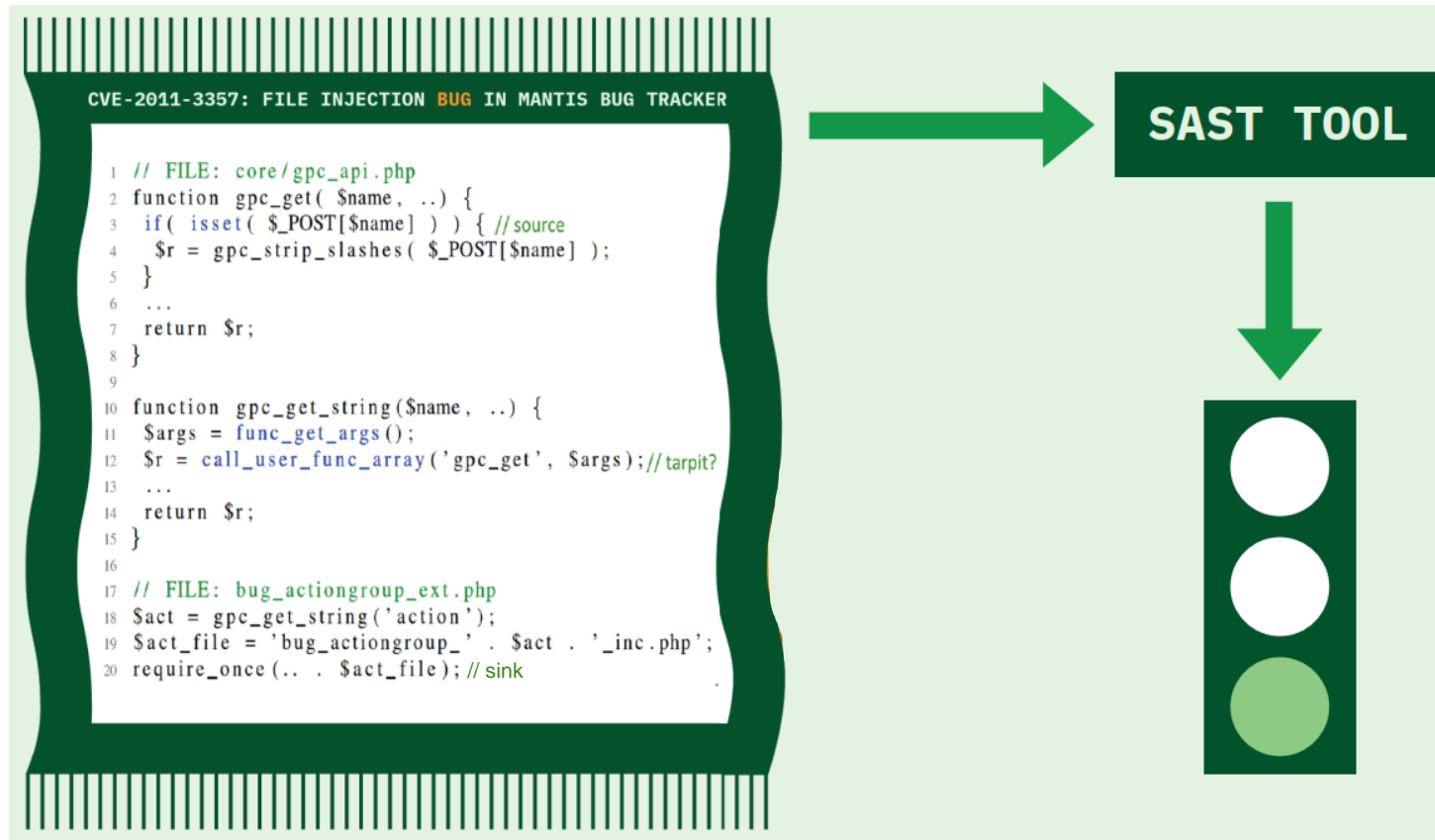
These patterns could include:

1. **Tightly Coupled Dependencies:** Identifying code sections where dependencies are tightly coupled, making it challenging to isolate components for testing or security analysis.
2. **Excessive Complexity:** Highlighting code structures that are overly complex, making it difficult to understand and create comprehensive tests or conduct thorough security analysis.
3. **Lack of Modularity:** Recognizing areas where the code lacks modularity, hindering the ability to test individual components independently or assess security vulnerabilities in isolation.
4. **Inadequate Input Validation:** Identifying patterns where input validation is insufficient or improperly implemented, leading to potential security vulnerabilities that are hard to detect through static analysis alone.
5. **Obscure Control Flow:** Noting instances where control flow within the code is convoluted or hard to trace, making it challenging to identify potential security weaknesses or create effective test cases.

The refinement of testability patterns in this context involves pinpointing these recurring code structures or design characteristics that pose obstacles to effective static analysis for security testing. By recognizing and addressing these patterns, the aim is to enhance the overall testability and security posture of web applications.

# Context: SAST and testability

Static application security testing (SAST) is widely used in industry to detect vulnerabilities [1]

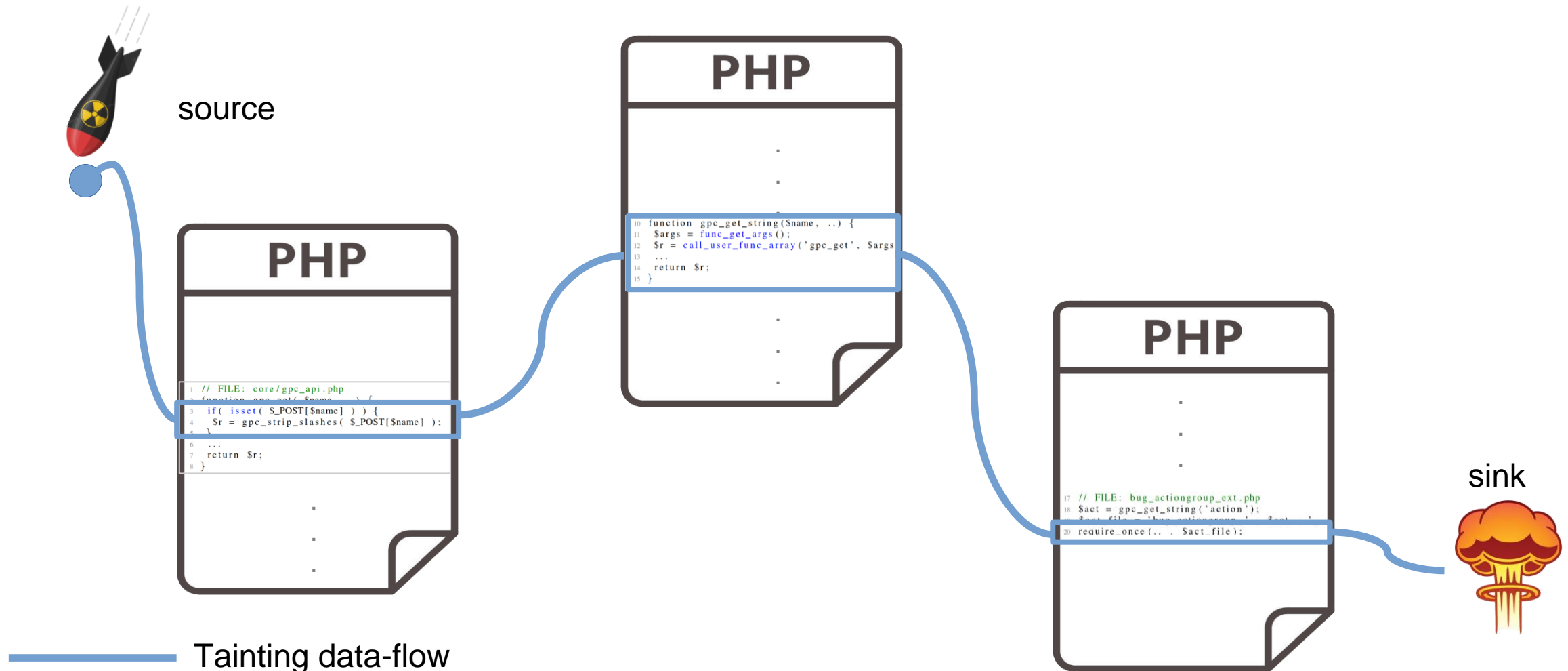


[1] [OWASP Code Review Guide v2.0](#), cf. Figure 1 and Figure 2



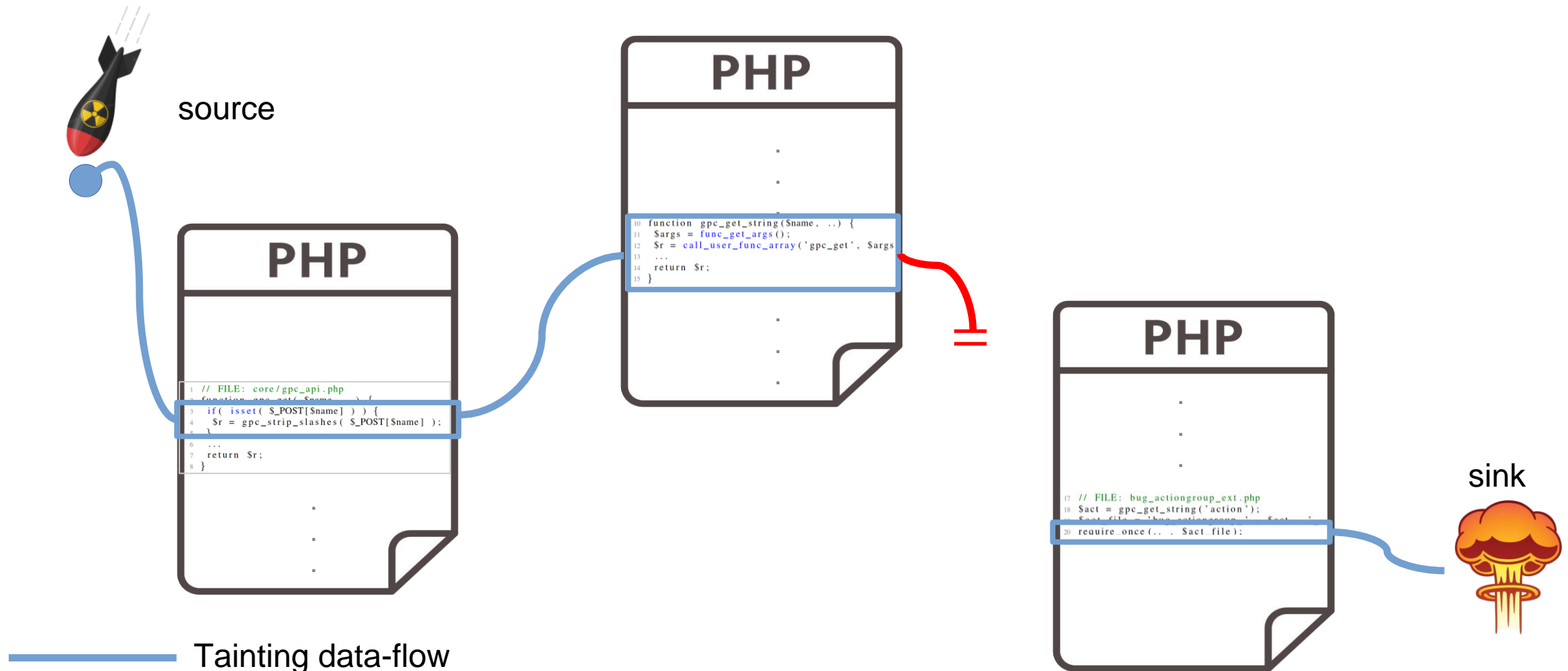
# Context: Injection vulnerabilities

Any attacker-controlled data (source) flowing in a dangerous operation (sink) without sanitization?



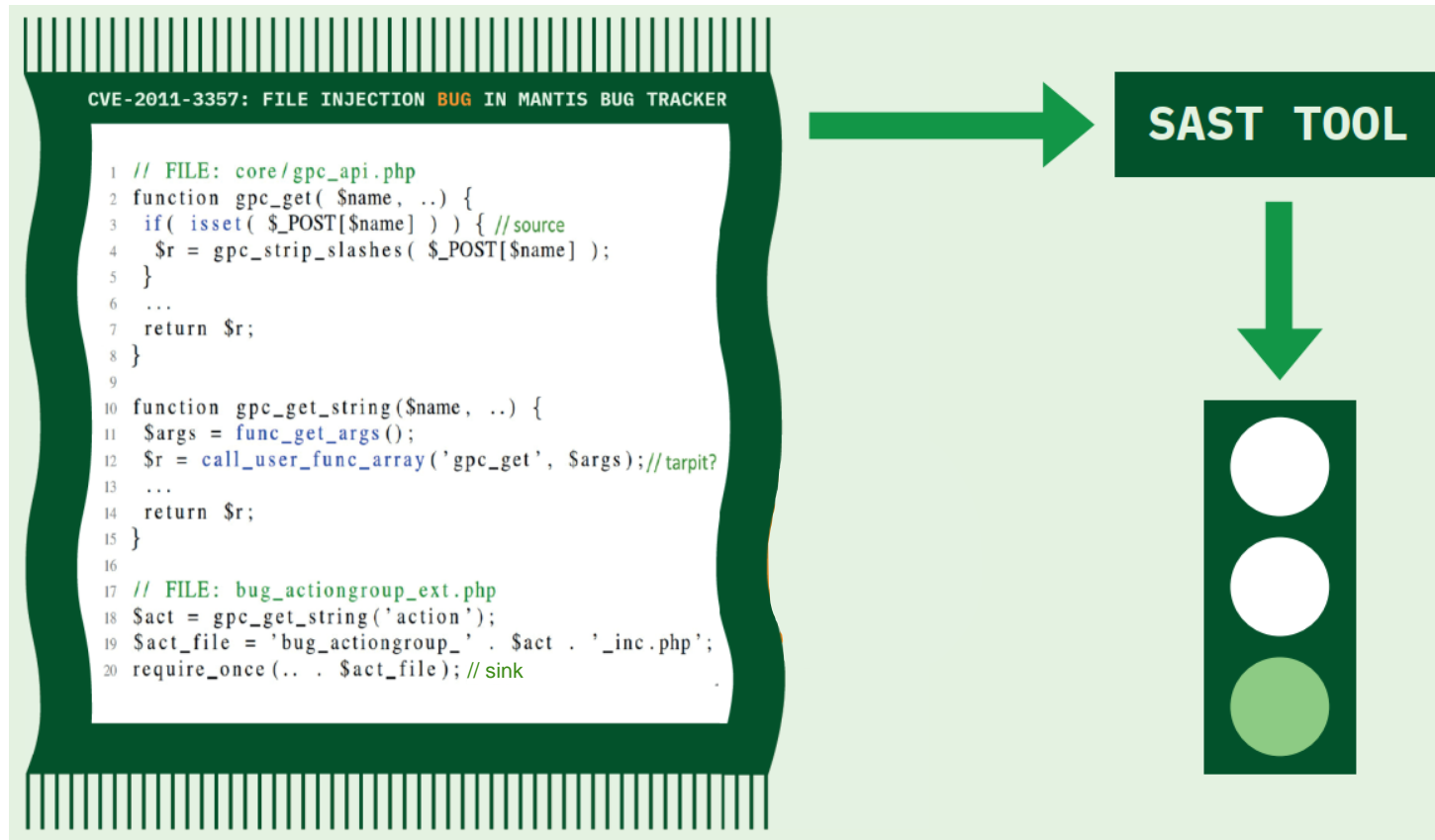
# Context: Injection vulnerabilities

Any attacker-controlled data (source) flowing in a dangerous operation (sink) without sanitization?



# Context: SAST and testability

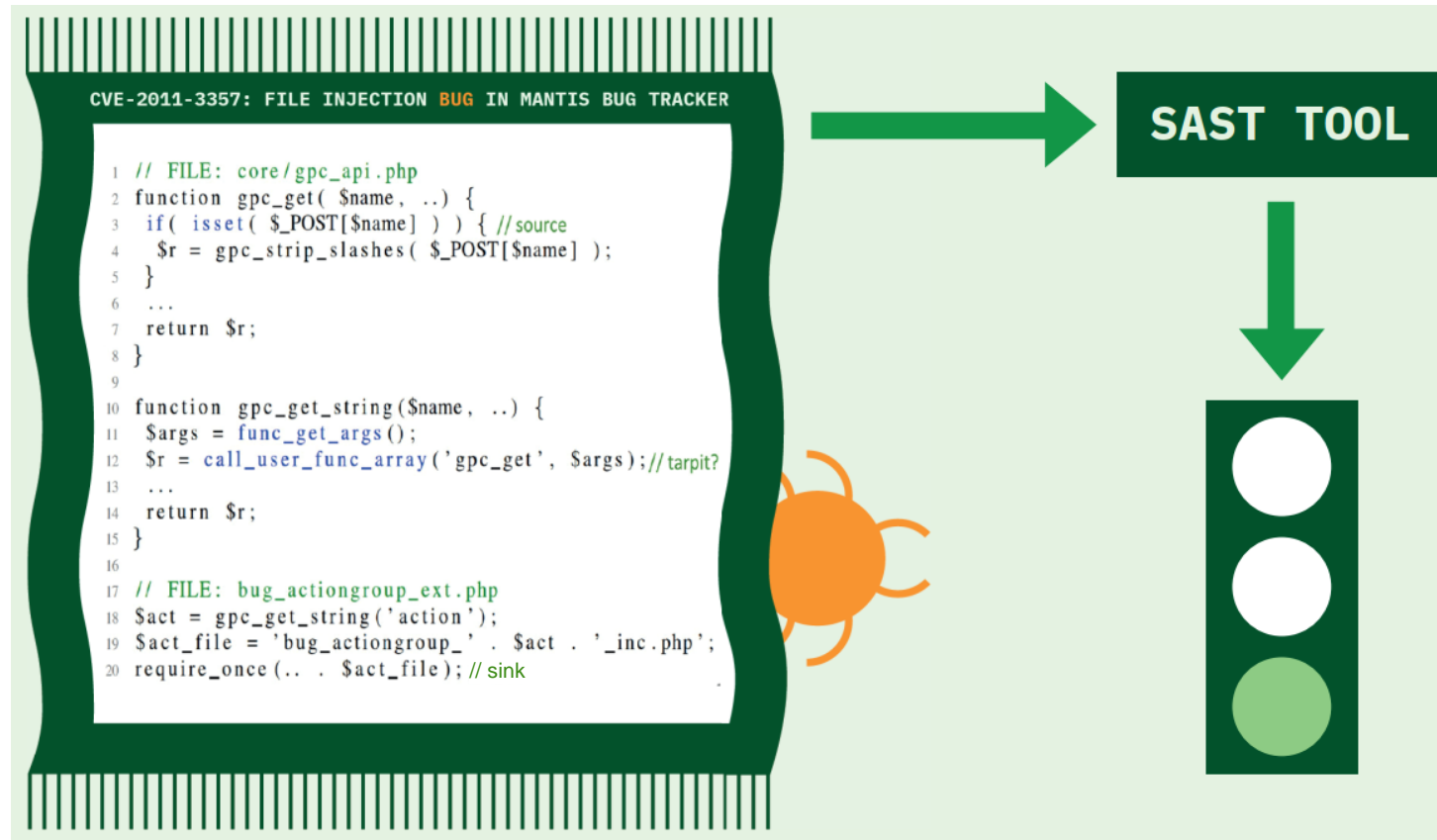
Static application security testing (SAST) is widely used in industry to detect vulnerabilities [1]



[1] [OWASP Code Review Guide v2.0](#), cf. Figure 1 and Figure 2

# Context: SAST and testability

Static application security testing (SAST) is widely used in industry to detect vulnerabilities [1]



**Was all the code analyzed?**  
**No bugs under the carpet?**

## Technical questions

Code obstacles impacting SAST?  
Can we measure these obstacles?  
Can we discover / remediate them?

[1] [OWASP Code Review Guide v2.0](#), cf. Figure 1 and Figure 2

# CVE-2011-3357: File inclusion in mantis bug tracker

```
20 require_once (.. . $act_file);
```

**EXECUTE THE CONTENT OF A FILE < ABUSABLE FUNCTION > SINK LIST**

# CVE-2011-3357: File inclusion in mantis bug tracker

```
20 require_once (.. . $act_file); // sink
```



# CVE-2011-3357: File inclusion in mantis bug tracker

```
17 // FILE: bug_actiongroup_ext.php
18 $act = gpc_get_string('action');
19 $act_file = 'bug_actiongroup_' . $act . '_inc.php';
20 require_once(.. . $act_file); // sink
```

**FUNCTION CALL < PROPAGATES ...> TAINTING CAPABILITIES**

**STRING CONCAT < PROPAGATES THE CONTROL OF  
A STRING TO ANOTHER > TAINTING CAPABILITIES**

# CVE-2011-3357: File inclusion in mantis bug tracker

```
10 function gpc_get_string($name, ...) {  
11     $args = func_get_args();  
12     $r = call_user_func_array('gpc_get', $args);  
13     ...  
14     return $r;  
15 }  
16  
17 // FILE: bug_actiongroup_ext.php  
18 $act = gpc_get_string('action');  
19 $act_file = 'bug_actiongroup_' . $act . '_inc.php';  
20 require_once(.. . $act_file); // sink
```

Diagram illustrating the flow of data from the `call_user_func_array` function call in line 12 to the `gpc_get_string` function call in line 18, which leads to the file inclusion in line 19.

??? < PROPAGATES ... > TAINTING CAPABILITIES

# CVE-2011-3357: File inclusion in mantis bug tracker

```

1  // FILE: core/gpc_api.php
2  function gpc_get( $name, .. ) {
3      if( isset( $_POST[$name] ) ) {
4          $r = gpc_strip_slashes( $_POST[$name] );
5      }
6      ...
7      return $r;
8  }
9
10 function gpc_get_string( $name, .. ) {
11     $args = func_get_args();
12     $r = call_user_func_array( 'gpc_get', $args );
13     ...
14     return $r;
15 }
16
17 // FILE: bug_actiongroup_ext.php
18 $act = gpc_get_string( 'action' );
19 $act_file = 'bug_actiongroup_' . $act . '_inc.php';
20 require_once( .. . $act_file ); // sink

```

**\$\_POST <TAKEN FROM UNTRUSTED SOURCE> SOURCE LIST**

# CVE-2011-3357: File inclusion in mantis bug tracker

```
1 // FILE: core/gpc_api.php
2 function gpc_get( $name, .. ) {
3     if( isset( $_POST[$name] ) ) { // source
4         $r = gpc_strip_slashes( $_POST[$name] );
5     }
6     ...
7     return $r;
8 }
9
10 function gpc_get_string( $name, .. ) {
11     $args = func_get_args();
12     $r = call_user_func_array( 'gpc_get', $args );
13     ...
14     return $r;
15 }
16
17 // FILE: bug_actiongroup_ext.php
18 $act = gpc_get_string( 'action' );
19 $act_file = 'bug_actiongroup_' . $act . '_inc.php';
20 require_once( .. . $act_file ); // sink
```

POST <https://mantisb.com/service>  
action=<ATTACKER\_PAYLOAD>

# Toward testability patterns for SAST

Many **SAST tools** (including commercial ones) do **not find** that **File inclusion**

```

1 // FILE: core/gpc_api.php
2 function gpc_get( $name, ..) {
3   if( isset( $_POST[$name] ) ) { // source
4     $r = gpc_strip_slashes( $_POST[$name] );
5   }
6   ...
7   return $r;
8 }
9
10 function gpc_get_string($name, ..) {
11   $args = func_get_args();
12   $r = call_user_func_array( 'gpc_get', $args ); // obstacle?
13   ...
14   return $r;
15 }
16
17 // FILE: bug_actiongroup_ext.php
18 $act = gpc_get_string( 'action' );
19 $act_file = 'bug_actiongroup_' . $act . '_inc.php';
20 require_once( .. . $act_file ); // sink

```

vulnerable app

pattern  
creation (1)



```

// replace with
// code companion for the obstacle
//
$a = $_GET["p1"]; // source
$b = $a // replace with obstacle!
echo $b; // sink

```

testability pattern skeleton (baseline XSS)

# Toward testability patterns for SAST

Many **SAST tools** (including commercial ones) do **not find** that **File inclusion**

```

1 // FILE: core/gpc_api.php
2 function gpc_get( $name, .. ) {
3   if( isset( $_POST[$name] ) ) { //source
4     $r = gpc_strip_slashes( $_POST[$name] );
5   }
6   ...
7   return $r;
8 }
9
10 function gpc_get_string($name, ..) {
11   $args = func_get_args();
12   $r = call_user_func_array('gpc_get', $args); //obstacle?
13   ...
14   return $r;
15 }
16
17 // FILE: bug_actiongroup_ext.php
18 $act = gpc_get_string('action');
19 $act_file = 'bug_actiongroup_' . $act . '_inc.php';
20 require_once( .. $act_file ); //sink

```

vulnerable app

pattern  
creation (1)



```

function F($var) { // code companion
  return $var;    // for the obstacle
}                //
$a = $_GET["p1"]; // source
$b = call_user_func_array("F", [$a]); // obstacle
echo $b; // sink

```

testability pattern instance

pattern  
transformation (3)



12 \$r = gpc\_get(...\$args); // no obstacle anymore

SAST  
measurement (2)



SAST	Correct
RIPS	NO
phpSAFE	NO
WAP	NO
Progpilot	YES
Comm_2	NO
Comm_1	YES

After that **transformation**, commercial tool Comm\_2 **finds** the **File inclusion**!



# Many variants of that pattern can be created...

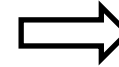
```
function F($var) { // code companion
    return $var;    // for the obstacle
}                //
$a = $_GET["p1"]; // source
$b = call_user_func_array("F", [$a]); // obstacle
echo $b; // sink
```

instance 1



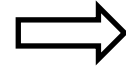
```
function F($var) { // code companion
    return $var;    // for the obstacle
}                //
$a = $_GET["p1"]; // source
$f = "F"; // obstacle
$b = call_user_func_array($f, [$a]); // obstacle
echo $b; // sink
```

instance 2

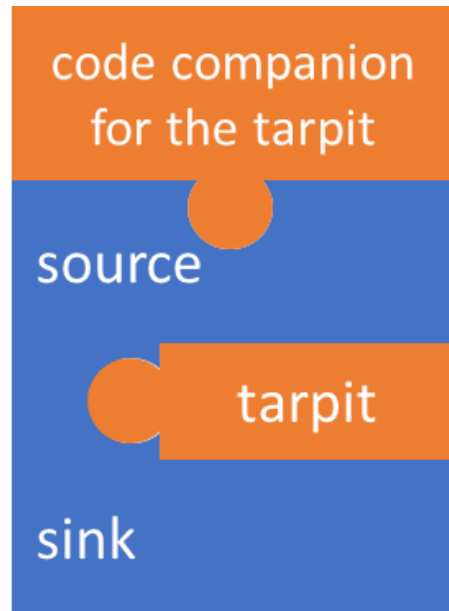


```
function F_1($var) { // code companion
    return $var;    // for the obstacle
}                //
function F_2($var) { //
    return "foo";    //
}                //
$a = $_GET["p1"]; // source
$f = $_GET["Func_id"]; // obstacle
$b = call_user_func_array("F_". $f, [$a]); // obstacle
echo $b; // sink
```

instance 3



...



# Testability Patterns for SAST

## Targeted 3 popular languages

- **PHP**: ~120 pattern instances (*mature: 8/10*)
- **JS**: ~150 pattern instances (*mature: 6/10*)
- **Java**: ~200 pattern instances (*mature: 3/10*)

We inspected the entire language manual

What do we want to do with them?

- SAST tools **measurement**
- **Discover** patterns into applications
- Remediate patterns to increase testability

The screenshot shows the GitHub repository page for 'testable-eu/sast-testability-patterns'. The repository is in the 'master' branch. The file list on the left includes: .github/ISSUE\_TEMP..., .vscode, JAVA, JS, PHP, docs, pattern\_template, .gitignore, LICENSE, and README.md. The README.md file is selected, showing the title 'Testability Pattern Catalogs for SAST' and a description: 'This repository includes catalogs of SAST testability patterns for the OWASP Testability Patterns project. Testability Patterns (TPs) are problematic code instructions that affect the capability of code analysis tools for security testing. Due to TPs, SAST tools may not detect an existing vulnerability, or conversely, report a false alarm. This project has a OWASP website available at: https://owasp.org/www-project-testability-patterns-for-web-applications/. Quick Start' followed by links: 'What is a testability pattern?', 'Testability patterns structure', 'Example pattern template', and 'Adding a pattern'. The right sidebar shows repository statistics: 2 stars, 1 watching, 0 forks, and 0 releases. It also lists contributors and a language usage chart: JavaScript 30.6%, Java 26.4%, Shell 19.4%, Scala 10.1%, HTML 7.7%, PHP 4.0%, and Python 1.8%. At the bottom right, there is a 'Suggested Workflows' section with a 'Publish Node.js Package to GitHub Packages' workflow.

<https://github.com/testable-eu/sast-testability-patterns>



# Testability Patterns Framework for SAST

Framework provides operations for

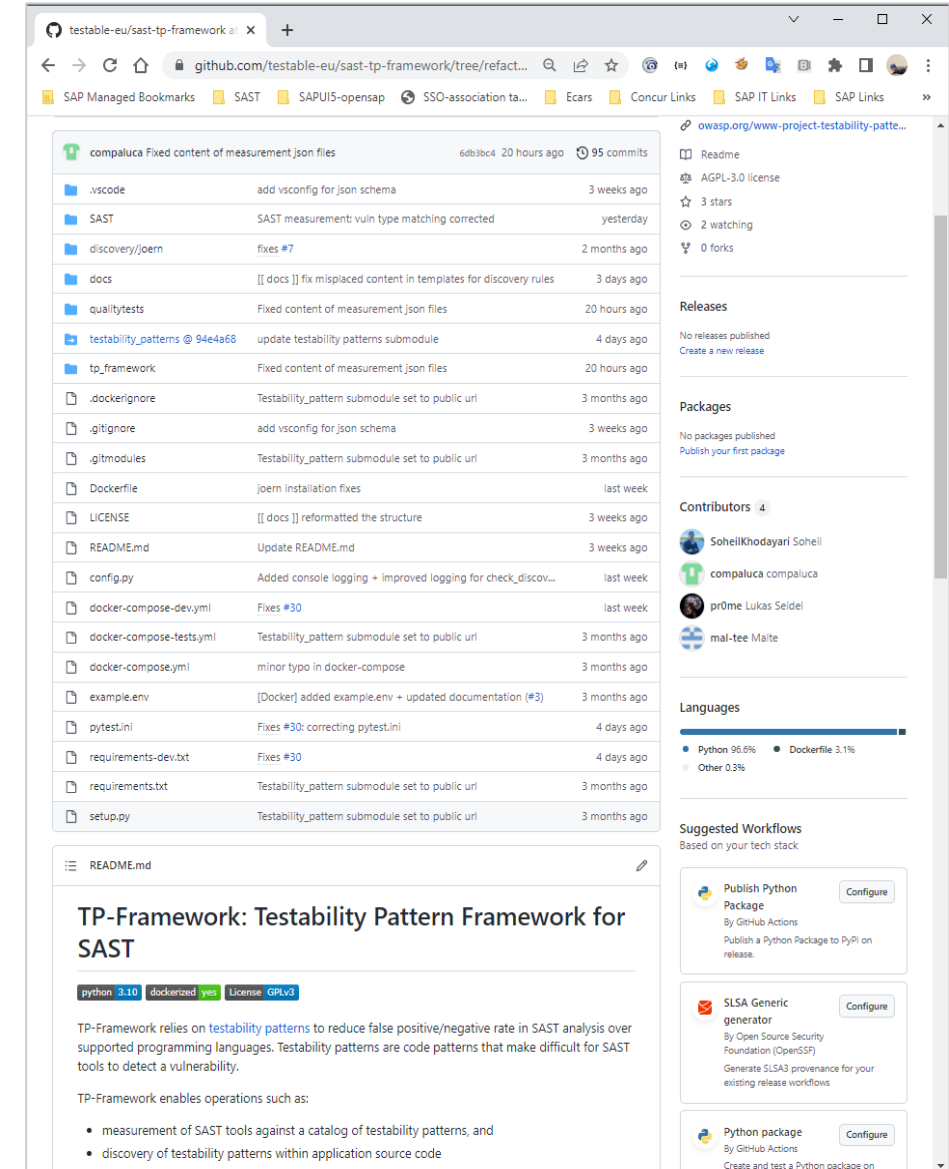
- SAST tools **measurement** (e.g., codeql integrated)
- **Discover** patterns into apps (via Joern and Scala queries)

```
tpframework measure -l JS -p 1 2 --tools codeql:2.13.1
```

```
tpframework discovery -t MYAPP/ -l PHP -a --tools T1:V1 T2:V2
```

## Results spoiler

1. Measurement: many SAST tools struggle on our patterns
2. Discovery: many apps in Github use these patterns
3. Testability for SAST can be improved!



The screenshot displays the GitHub repository for 'testable-eu/sast-tp-framework'. The repository is a public project with 95 commits and 3 stars. The README.md file is open, showing the title 'TP-Framework: Testability Pattern Framework for SAST' and a description of the framework's purpose and supported languages. The right sidebar shows repository statistics, releases, packages, contributors, and suggested workflows.

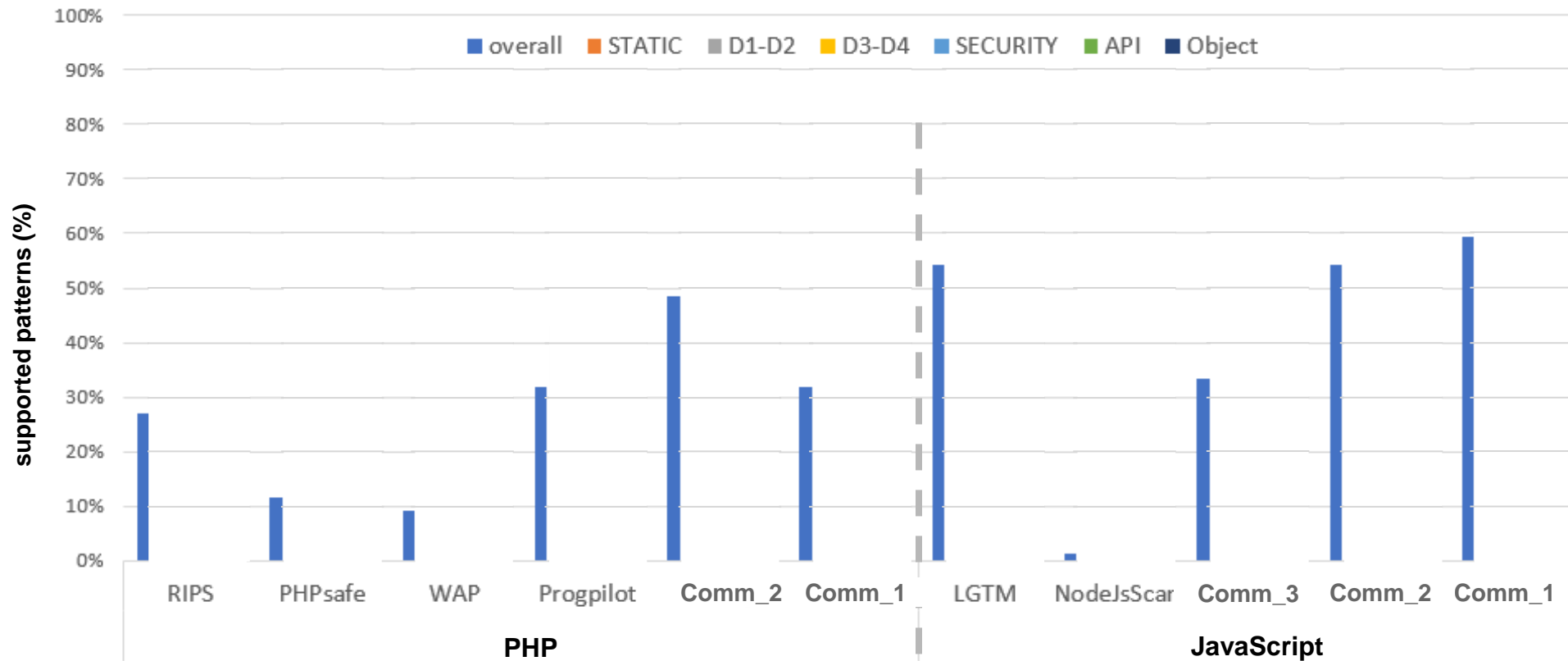
<https://github.com/testable-eu/sast-tp-framework>



# 1. Measurement: many SAST tools struggle on our patterns

**Measured** our pattern instances against **SAST**

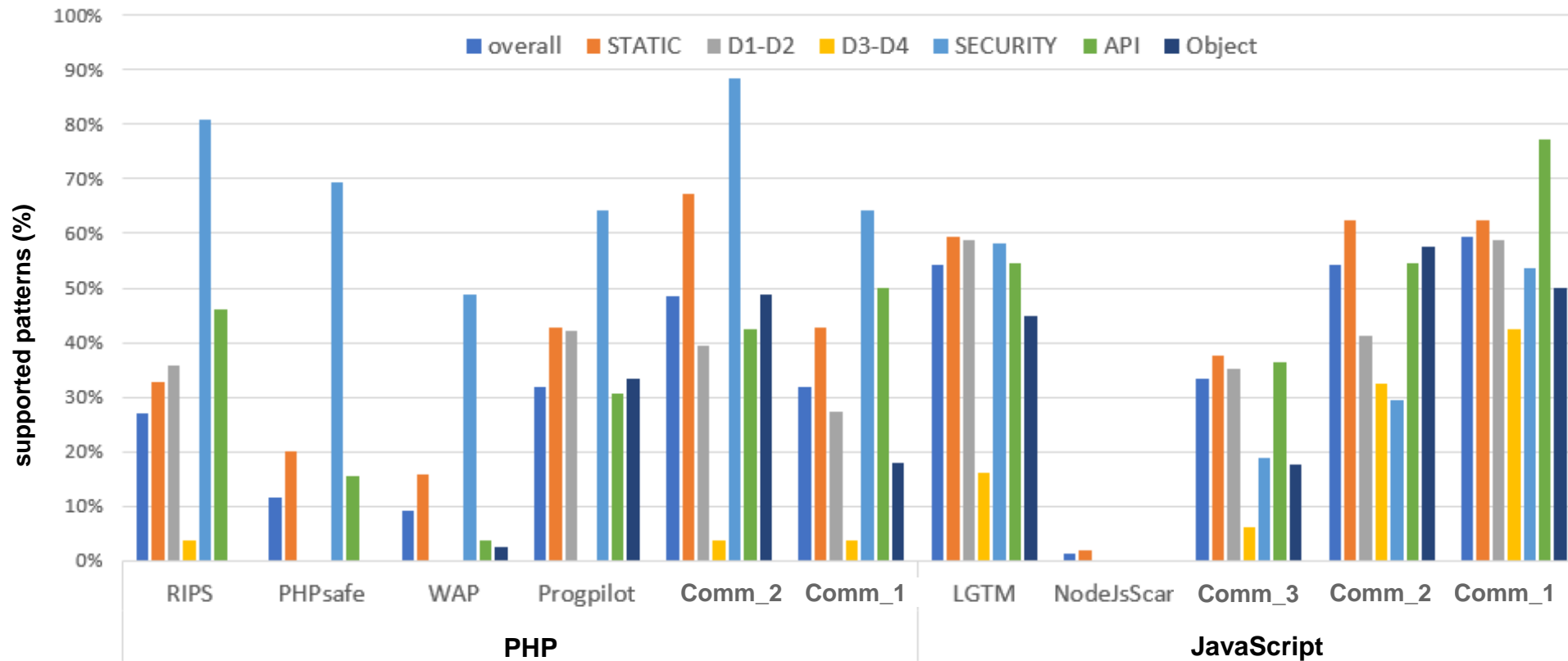
- Overall: <50% support for PHP and <60% for JS
- Tools Combination: 66% PHP, 82% JS



# 1. Measurement: many SAST tools struggle on our patterns

**Measured** our pattern instances against **SAST**

- Overall: <50% support for PHP and <60% for JS
- Tools Combination: 66% PHP, 82% JS
- Studied **different dimensions** (internal API vs Feature, security-related, ...)

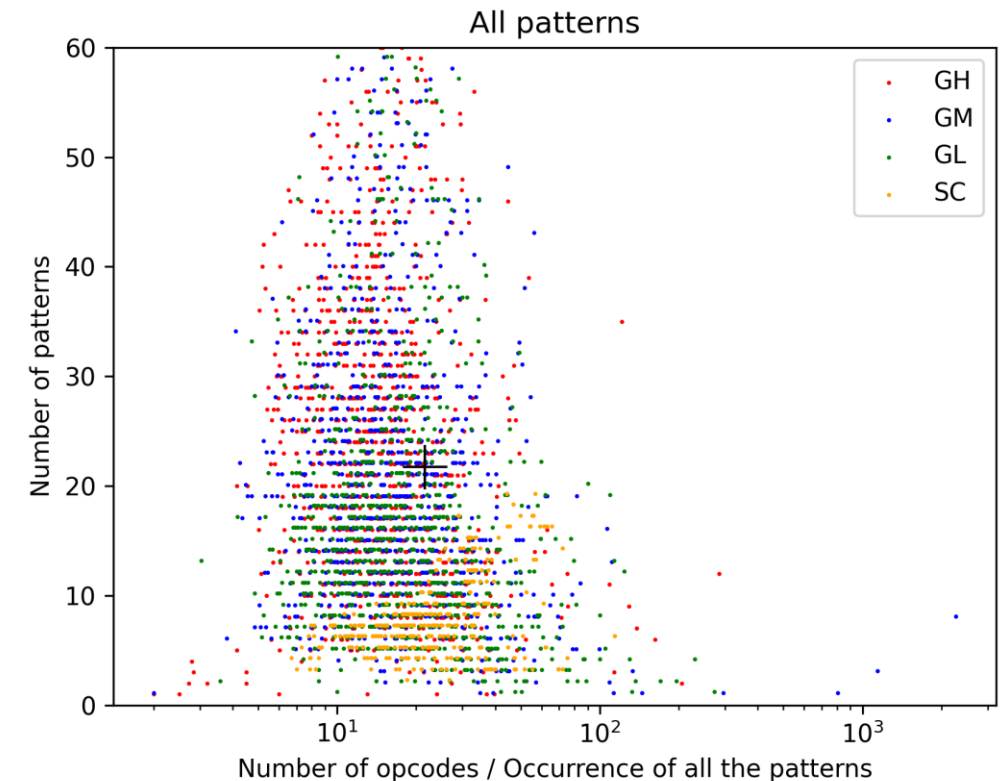


## 2. Discovery: many apps in Github use these patterns

**PHP**: created **discovery rules** for our patterns and run them over **>3000** open-source PHP **apps** (from Github and Sourcecodester)

Our patterns are very **prevalent** in the real world

	Unique obstacles per Project	obstacles per LoC
AVG	21	20



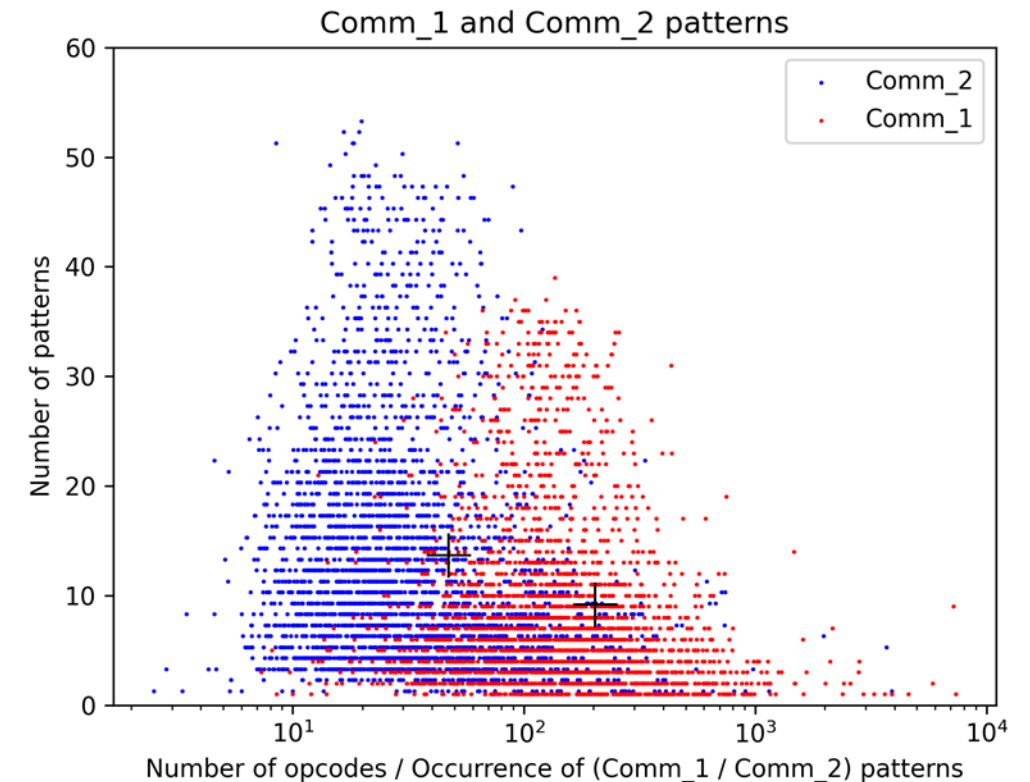


## 2. Discovery: many apps in Github use these patterns

**PHP**: created **discovery rules** for our patterns and run them over **>3000** open-source PHP **apps** (from Github and Sourcecodester)

Our patterns are very **prevalent** in the real world

	Unique obstacles per Project	obstacles per LoC
AVG	21	20
Comm_1	8	203
Comm_2	13	47



## 2. Discovery: many apps in Github use these patterns

ID	Pattern	#i	API	SEC	Dyn	OOP	Neg	Tools	SC		$G_L$		$G_M$		$G_H$	
									prj	med	prj	med	prj	med	prj	med
1	static_variables	1			S			-----	50	13	443	4	635	7	712	14.0
2	global_variables	1			S			-S--XY	89	10	203	7	213	10	210	12.0
3	global_array	1			S			--WP-Y	33	6	138	4.0	162	5.0	179	7
4	conditional_assignment	1			S			R-WPXY	221	74	795	18	890	31.5	908	59.0
5	combined_operator	1			S			R-WPXY	335	170	919	33	942	64.0	934	97.5
6	coalesce	1			S			RS--XY	0	0	0	0	280	6.0	433	11
7	string_arithmetic_operations	1		✓	S			RSW-XY	277	10	523	6	636	9.0	707	11
8	simple_reference	1			S			----X-	42	39	163	9	231	5	292	6.0
9	reference_argument	1			S			----XY	208	14	387	7	399	10	486	9.0
10	return_by_reference	1			S			-----	19	11	83	4	95	4	132	4.0
11	foreach_with_reference	1			S			-----	41	7	182	3.0	238	4.0	321	4
12	make_ref	2			S		✓	---P-Y	25	6	116	6.0	134	4.0	180	4.0
13	assign_static_prop_ref	1			S			---PX-	9	1.0	19	1	10	1.0	17	1
14	object_assigned_by_reference	1			S			----X-	22	21	100	6.5	107	7	155	5
15	nested_function	1			S			-SWPX-	66	3.5	166	4.0	222	4.0	283	5
16	variadic_functions	1			S			----XY	1	12	59	3	143	3	239	3
17	get_arguments	1			S			R-----Y	23	5.0	106	4.0	137	4	191	4
18	send_unpack	1			S			RS--X-	1	31	73	3	146	3.0	264	4.0
19	closures	2			D2			----XY	36	1	543	7	733	11	782	25.5
20	use_with_closures	2			D2		✓	----XY	25	1	321	4	524	5.0	614	12.0
21	simple_object	1			S	✓		---PXY	336	199	968	350.5	977	863	974	1536.5
22	assign_object	1			S	✓		--W-X-	30	3	138	4.0	212	4.0	325	4
23	object_argument	1			S	✓		----X-	119	30	591	23	718	53.5	804	79.5
24	new_self	1			S	✓		----X-	41	6	162	2.0	249	3	351	3
25	clone	1			S	✓		---PX-	41	6	147	3	238	5.0	338	5.0
26	late_static_binding	2			D2	✓	✓	----X-	13	1.0	165	4	279	5	386	8.0
27	get_called_class	1			D2	✓		-----	0	0	16	1.0	28	1.0	34	2.0
28	static_methods	1			S	✓		---PX-	119	17	792	29.0	865	61	898	126.5

⋮

80	callback_functions	1			D1			---P-Y	41	2	128	3.0	159	3	208	2.5
----	--------------------	---	--	--	----	--	--	--------	----	---	-----	-----	-----	---	-----	-----

`call_user_func_array('gpc_get', $args);`

### 3. Testability for SAST can be improved!

**Remediation 1:** Two **transformation experiments** targeting PHP and JS

- transformations intended as code rewriting for obstacles
- **>9000 new alerts: 370 true positives** in 48 apps (over ~2700 alerts inspected)
- **182** true positives **already confirmed** from 31 projects: 38 impacting popular Github projects

**Remediation 2:** **improve SAST tools**

- Force SAST tools to collaborate to reduce false negatives, see our paper at USENIX 2023, [2]
- SAST tool vendors improving their tool on top of the testability patterns, e.g., our little story with codeQL

**Remediation 3:** provide **custom rules** to make SAST tools overcoming obstacles (*on-going work*)

[2] WHIP: Improving Static Vulnerability Detection in Web Application by Forcing tools to Collaborate. Feras Al-Kassar et al. USENIX Sec. 2023

# Story 1: SAST tool improvement

SAP Security Research

- measuring SAST tools
- internal dissemination

SAP Security Testing Team

- processing the results
- selection of some results
- feedback to SAST tools
- e.g., codeql tickets

codeQL

- processing the results
- improving the tool
- releasing new version

SAP Security Research

- measuring new SAST tools' versions
- evaluating the improvement

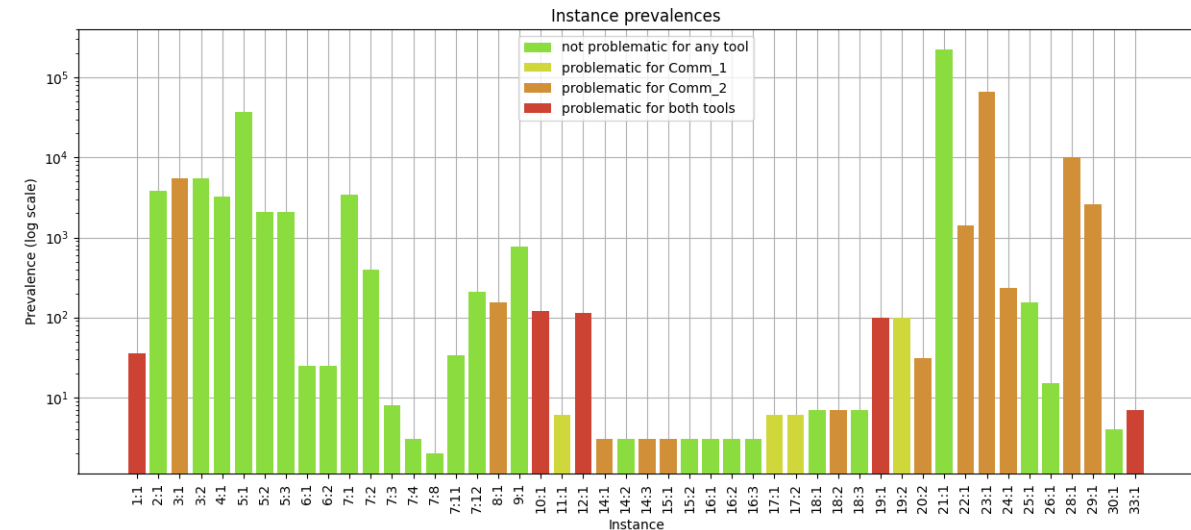
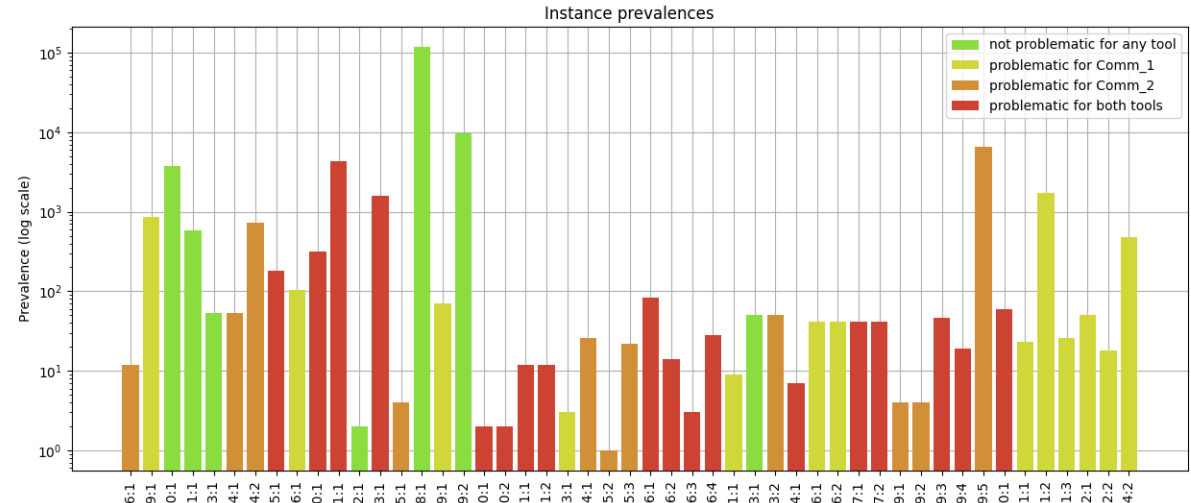
# Story 2: Usage at SAP

Testability of a SAP application wrt SAST

Which patterns are there?

Which patterns are more problematic?

Anything we can do to remediate them?



# Call for action

Join our project and our community!

## Contribute

- <https://github.com/testable-eu/sast-testability-patterns>
- <https://github.com/testable-eu/sast-tp-framework>

There is a lot to do

- testability patterns: create new ones, mature existing, target other programming languages...
- framework (python): improve features, integrate a new SAST tool, implement new features, ...
- ...

The screenshot shows the OWASP Testability Patterns for Web Applications website. The page has a light blue header with the OWASP logo and navigation links. The main content area is titled "OWASP Testability Patterns for Web Applications" and includes a "Main" tab, language filters for Java, JavaScript, and PHP, and a description of the project's mission. The "Objective" section lists the target audiences: Web and AI/ML Developers, Managers, and Security Teams. The "Roadmap" section outlines the project's timeline from June 2022 to later 2024. A right sidebar contains additional information, including the OWASP Foundation's mission, project details, social links, and code repositories.

OWASP Testability Patterns for Web Applications

Watch 2 Star 4

Main

JAVA JAVASCRIPT PHP

**OWASP Testability Patterns for Web Applications**

OWASP TESTABLE redefines the classical secure development life-cycle around the concept of testability, providing new tools for:

- Web and AI/ML Developers
- Managers
- Security Teams

**Objective**

The OWASP project aims to deliver tools and methodologies for:

- Managers: New metric quantifying the security and privacy risks of a program.
- Developers: Better and flexible tools to improve testability, reducing security and privacy risk exposure.
- Security teams: Better and more flexible security, privacy testing, and AI/ML tools.

**Roadmap**

- June 2022: OWASP Project created
- February 2023:
  - OWASP Project kickstart @ OWASP AppSec EU
  - Open Source release of the main project repositories
- September 2023:
- September 2024: Release of the TESTABLE Standard Document
- Later 2024: Active and self sustained community

**The OWASP® Foundation** works to improve the security of software through its community-led open source software projects, hundreds of chapters worldwide, tens of thousands of members, and by hosting local and global conferences.

**Testability Patterns for Web Applications Information**

- Incubator Project
- Breaker
- Builder
- Tool
- Version 0.0.1

**Social Links**

- Twitter
- LinkedIn

**Code Repositories**

- TP-Framework
- Testability Pattern
- Catalogs for SAST

**Change Log**

- changes



## E.g. Developer of a SAST Tool



Lukas Seidel

Qwiet AI  
*formerly known as*  
 **ShiftLeft**

### Benefits you get

Measure SAST tools against Testability Patterns:  
identify strengths / weakness (B1)

Discover patterns in open source apps (B2)

(B1)+(B2): Prioritize which patterns to support better

### Incentives to contribute

Define/improve patterns on which your tool out-perform

Question/improve patterns on which your tool under-perform

## E.g. Web Developer



Malte Wessels



### Benefits you get

Identify testability obstacles in your app (pattern discovery)

Remediate these obstacles to make your app more testable

Select which SAST tool to use on your app

### Incentives to contribute

Capture as testability patterns the challenges you are facing with SAST

Question/improve patterns that cannot be removed from your app

## E.g. Security central team



**Me** on behalf of  
colleagues



### Benefits you get

Identify testability patterns in the entire organization (pattern discovery)

Devise remediation strategies for top 10 patterns

- code refactoring practices can be introduced and disseminated to dev teams in the organization
- channel request for improvements to SAST tools

Select which SAST tool(s) to buy (also depending on the SAST measurement on the top 10 patterns)

### Incentives to contribute

Capture as testability patterns the challenges the organization is facing with SAST

Question/improve patterns that cannot be removed from your organization

## E.g. Auditors



**Matteo Meucci**



### Benefits you get

Identify testability patterns in their customer apps (pattern discovery)

Consult their customers to make their apps more testable

Select which SAST tool(s) to use on which customer app

### Incentives to contribute

Capture as testability patterns the challenges experienced while using SAST on customer projects

Define/improve/remediate patterns to demonstrate thought leadership to their customers

# OWASP Top 10 testability patterns for SAST?

## Preliminary results for PHP

### SAST measurement score

- weighted score of SAST tools failing on the pattern
- top performer tools (often commercial) get higher weight
- e.g., 3.9 == all the 6 SAST tools failed

### Pattern discovery score

- how many findings of that pattern in a reasonable large and representative dataset
- e.g., for PHP
  - Github low popularity: 1,000 projects – [20 , 70] stars
  - Github medium popularity: 1,000 projects – [200 , 700] stars
  - Github high popularity: 1,000 projects – > 1,000 stars
  - Sourcecodester: 341 Projects (customizable web apps)

	pattern name	SAST measurement score	Pattern Discovery score
1	throw_exception	3,9	2189
2	static_variables	3,9	1802
3	variable_variables	3,9	1464
4	function_variable	3,9	1400
5	object_to_array	3,9	1182
6	methods_variable	3,9	834
7	foreach_with_reference	3,9	758
8	static_instance	3,9	699
9	new_from_variable	3,9	691
10	array_map	3,9	678
11	buffer	3,9	548
12	object_callable	3,9	487
13	dirname	3,9	347
14	dynamic_include	3,9	347
15	return_by_reference	3,9	317
16	extract	3,9	289
17	get_overloading	3,9	207
18	array_walk	3,9	181
19	call_overloading	3,9	178
20	callback_functions	3,9	178

# An OWASP project: journey started, join us!

Targeting the **Testability** dimension

First concrete result: **Testability Patterns for SAST**

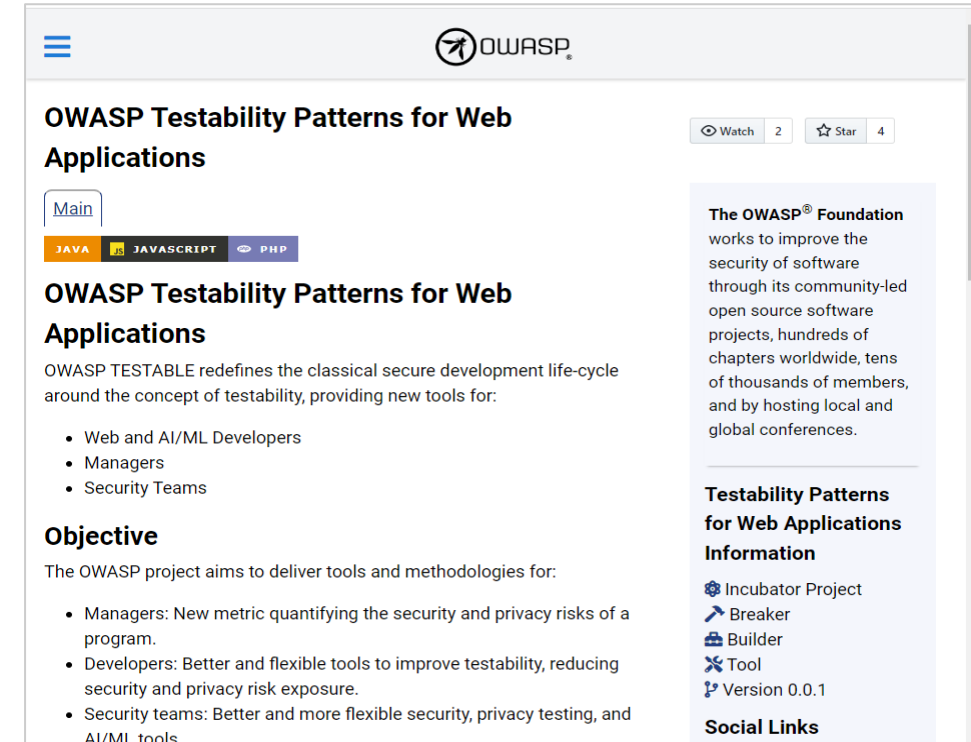
- <https://github.com/testable-eu/sast-testability-patterns>
- <https://github.com/testable-eu/sast-tp-framework>

Help us with your valuable expertise

<https://owasp.org/www-project-testability-patterns-for-web-applications/>

Let us devise OWASP top 10 testability patterns for SAST!

Can we do the same for DAST, Privacy, ML?



# Thank you!

## People

20 Security Researchers

+ PhDs **(positions available)**

+ interns **(positions available)**



## Portfolio

