Technical Report

Symfony YAML

Security Assessment

Prepared for:
OSTIF

SHIELDER
WEB SECURITY

1. Document Details

Classification	**Public** - CC BY-SA 4.0
Last review	June 30, 2026
Author	Pietro Tirenna

1.1. Version

Identifier	Date	Author	Note
v1.0	December 23, 2025	Pietro Tirenna	First version
v1.1	December 23, 2025	Abdel Adim Oisfi	Peer Review
v1.2	June 30, 2026	Pietro Tirenna	Public release

1.2. Contacts Information

Company	Name	Position	Contact
Shielder	Abdel Adim `smaury` Oisfi	CEO	abdeladim.oisfi@shielder.it
Shielder	Pietro Tirenna	Security Researcher	pietro.tirenna@shielder.it
OSTIF	Derek Zimmer	Executive Director	derek@ostif.org
OSTIF	Amir Montazery	Managing Director	amir@ostif.org
OSTIF	Helen Woeste	Communications and Community Manager	helen@ostif.org
OSTIF	Tom Welter	Project Manager	tom@ostif.org
Symfony	Nicolas Grekas	Symfony Developer	nicolas.grekas@symfony.com
Symfony	Fabien Potencier	Symfony YAML Core Developer	fabien@symfony.com

1.3. About OSTIF

The **Open Source Technology Improvement Fund (OSTIF)** is dedicated to resourcing and managing security engagements for open source software projects through partnerships with corporate, government, and non-profit donors. We bridge the gap between resources and security outcomes, while supporting and championing the open source community whose efforts underpin our digital landscape.

Over the past ten years, OSTIF has been responsible for the discovery of over 800 vulnerabilities, (121 of those being Critical/High), over 13,000 hours of security work, and millions of dollars raised for open source security. Maximizing output and security outcomes while minimizing labor and cost for projects and funders has resulted in partnerships with multi-billion dollar companies, top open source foundations, government organizations, and respected individuals in the space. Most importantly, we've helped over 120 projects and counting improve their security posture.

Our directive is to support and enrich the open source community through providing public-facing security audits, educational resources, meetups, tooling, and advice. OSTIF's experience positions us to be able to share knowledge of auditing with maintainers, developers, foundations, and the community to further secure our infrastructure in a sustainable manner.

We are a small team working out of Chicago, Illinois. Our website is ostif.org. You can follow us on social media to keep up to date on audits, conferences, meetups, and opportunities with OSTIF, or feel free to reach out directly at contactus@ostif.org or our Github.

Derek Zimmer, Executive Director
Amir Montazery, Managing Director
Helen Woeste, Communications and Community Manager
Tom Welter, Project Manager

2. Summary

3. Executive Summary

The document aims to highlight the findings identified during the “Security Assessment” against the "Symfony YAML" product described in section “3.2 Context and Scope”.

For each detected findings, the following information are provided:

- **Severity**: findings score ("3.7 Findings Severity Classification").
- **Affected resources**: vulnerable components.
- **Status**: remediation status ("3.8 Remediation Status Classification").
- **Description**: type and context of the detected finding.
- **Impact**: loss of confidentiality, data integrity and/or availability in case of a successful exploitation and conditions necessary for a successful attack.
- **Proof of Concept**: evidence and/or reproduction steps.
- **Suggested remediation**: configurations or actions needed to mitigate the finding.
- **References**: useful external resources.

3.1. Overview

From December 9, 2025 to December 23, 2025 **Shielder** was hired by the **Open Source Technology Improvement Fund (OSTIF)** to perform a *Security Audit* of the **Symfony YAML Component**, a PHP library to load and dump YAML files.

YAML – *YAML Ain't Markup Language* – is a human-friendly data serialization language for all programming languages. It is a popular format for configuration files, balancing readability with advanced features.

The YAML component provides:

- A `Parser` to load YAML file into PHP.
- A `Dumper` to serialize PHP structures to YAML.
- A `LintCommand` CLI to validate the correct syntax of YAML files.

A team of 1 (one) Shielder researcher worked on this project for a total of 10 (ten) person-days of effort.

3.2. Context and Scope

The YAML component is shipped by default together with Symfony, an industry leading PHP framework for building web applications. As such, developers of Symfony applications will likely use it to build interfaces with YAML files. Nonetheless, the component can also be included as a standalone PHP library.

The audit was mainly focused on:

- Assessing risks of passing an untrusted YAML content to the Parser.

- Assessing lack of proper security considerations in the documentation and in the provided examples.

3.3. Methodology

Due to the limited size of codebase, the audit was mostly performed following a Manual Source Code Review approach.

A fuzzing campaign using the experimental PHP-Fuzzer was established. However, due to some limitations of the tool and of the harnesses developed, this simple fuzzing campaign did not lead to interesting results.

The code review was mainly directed towards the following threats:

- Code deserialization gadgets triggered during YAML parsing.
- Leaks of sensitive information in the output PHP after YAML parsing.
- Resource starvation or program hangs occurring during YAML parsing.

Finally, in order to establish the adherence of the Symfony YAML component to the YAML spec, a script was implemented to test the implementation against the YAML Test Suite. The logic of the script was essentially to pass the `in.yaml` file present in each test case against the parser, then producing an output JSON via the PHP `json_encode` function, and finally to compare the resulting JSON with the `in.json` file provided in the test.

3.4. Audit Summary

The **Symfony YAML Component** is adequately robust and well designed in terms of security, but there is still room for improvement in some areas.

The Shielder team was able to identify of **3** (three) low and **2** (two) informational findings.

The main threats included in this report are caused by unmitigated resource starvations with complex data, and by a lack of proper security documentation.

3.5. Recommendations

Security Documentation

The documentation of the Symfony YAML Component currently lacks a "Security" documentation to inform the users about the known vulnerabilities, the risks attached with the different configuration properties, and general recommendations.

Hardening Against Parsing Differentials

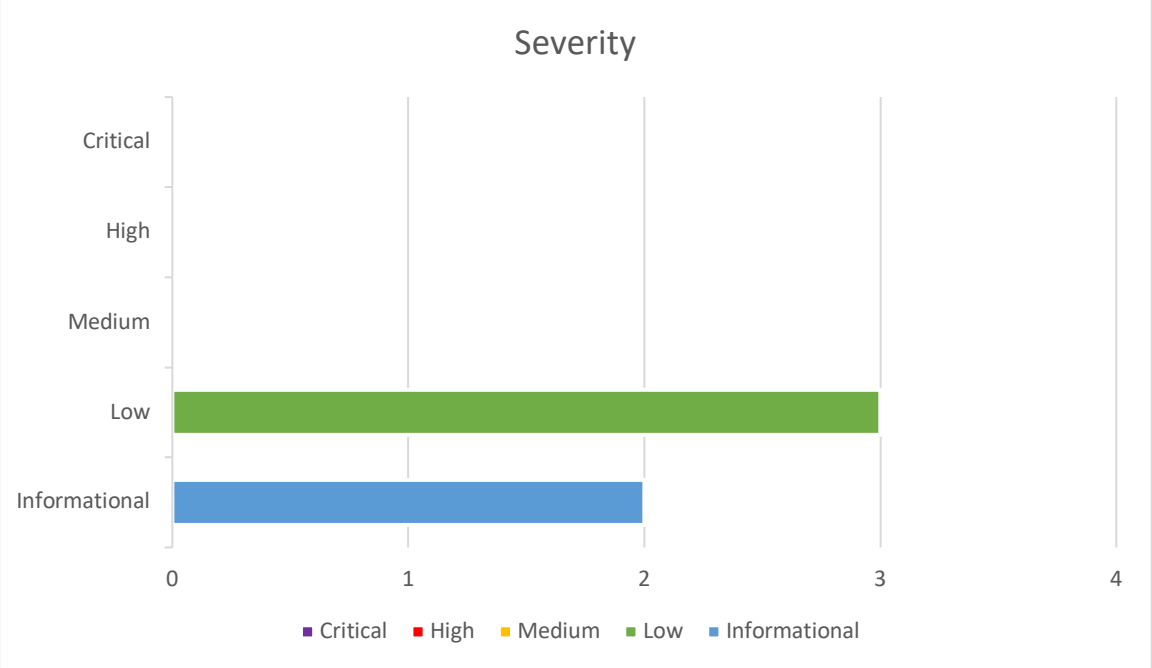
The Symfony YAML component, when tested against the YAML test suite, exhibits multiple inconsistencies with the spec. This could lead to parser differentials attack, for example in case developers pass the same data to Symfony YAML for validation/sanitization and then to other parsers for actuation.

It this recommended to improve adherence to the spec and resiliency to parser differentials by:

- Implementing a Symfony YAML interface for https://play.yaml.com/, so that differences with other popular parsers are easier to quantify
- Improving success ratio against the YAML test suite
- Include a callout in the documentation to inform developers of parser differentials risk when using different multiple parsers for validation/sanitization and then for operations

3.6. Results Summary

The following chart shows the number of findings found per severity:



ID	Finding	Severity	Status
1	Denial of Service (DoS) via Infinite Recursion of Nested YAML Blocks	LOW	Closed
2	Denial of Service (DoS) via Unbounded Alias Resolution	LOW	Closed
3	Regular Expression Denial of Service (ReDoS) in Parser::cleanup	LOW	Closed
4	Lack of Security Warning for Object Deserialization	INFORMATIONAL	Closed
5	Lack of Security Warning for Constant Resolution	INFORMATIONAL	Closed

3.7. Findings Severity Classification

Severity	Description
CRITICAL	Vulnerability that allows to compromise the whole application, host and/or infrastructure. In some cases, it allows access, in read and/or write, to highly sensitive data, totally impacting the resources in terms of confidentiality, integrity and availability. Usually, such vulnerabilities can be exploited without the need of valid credentials, without considerable difficulty and with the possibility of automated, highly reliable, and remotely triggerable attacks. Vulnerabilities marked with this severity must be resolved quickly, especially in production environment.
HIGH	Vulnerability that significantly affects the confidentiality, integrity, and availability of confidential and sensitive data. However, the prerequisites for the attack affect its likelihood of success, such as the presence of controls or mitigations and the need of a certain set of privileges.
MEDIUM	Vulnerability that allows to obtain only a limited or less sensitive set of data, partially compromising confidentiality. In some cases, it may affect the integrity and availability of the information, but with a lower level of severity. In addition, the chances of success of such vulnerability may depend on external factors and/or conditions outside the attacker's control.
LOW	Vulnerability resulting in a limited loss of confidentiality, integrity, and availability of data. In some cases, it depends on conditions not aligned to a real scenario or requires that the attacker has access to credentials with a high level of privileges. In addition, a low severity vulnerability may provide useful information to successfully exploit a higher impact vulnerability.
INFORMATIONAL	Problems that do not directly impact confidentiality, integrity, and availability. Usually, these problems indicate the absence of security mechanisms or the improper configuration of them. Mitigation of this type of problem increases the general level of security of the system and allows in some cases to prevent potential new vulnerabilities and/or limit the impact of existing ones.

3.8. Remediation Status Classification

Status	Description
Open	Vulnerability not mitigated or insufficient mitigation.
Not reproducible	Vulnerability not reproducible due to environment changes or to mitigation of other vulnerabilities required in the reproduction steps.
Closed	Vulnerability mitigated. The security patch applied is reasonably robust.

4. Findings Details

Analysis results are discussed in this section.

4.1. Denial of Service (DoS) via Infinite Recursion of Nested YAML Blocks

Severity	LOW
Affected Resources	Parser.php
Status	Closed

Update

The vulnerability was reported to the `security@symfony.com` mailing list. It was then moved to the GitHub advisory GHSA-c2p3-7m5p-cv8x and assigned the CVE-2026-45133 number.

Description

The `doParse()` and `parseBlock()` methods in `Parser.php` exhibit mutual recursion: when `doParse()` encounters an indented line, it calls `parseBlock()`, which instantiates a new parser and calls `doParse()` on the indented content. Since the code does not enforce a limit on the maximum depth of the content, this cycle can continue indefinitely for deeply nested structures, consuming more resources for each indentation level, eventually leading to a Fatal Error.

Impact

An attacker might use this vulnerability to crash the PHP process that uses the YAML parser.

Attack Complexity

The attacker would need to be able to feed arbitrary YAML input to the parser.

Related Issues

N/A

Proof of Concept

The vulnerability can be reproduced by passing the attached `test_recursion.yaml` to the following test code:

```
# test_parser.php
<?php
use Symfony\Component\Yaml\Yaml;

$yamlFile = $argv[1];
$yamlContent = file_get_contents($yamlFile);
```

```
$result = Yaml::parse($yamlContent, 0);
var_dump($result);
```

The test can be simply executed by running:

```
php test_parser.php test_recursion.yaml
```

which clearly shows, in the stack trace, the recursive pattern.

Suggested Remediations

Implement a maximum depth of recursion, when parsing nested YAML blocks.

References

- https://cwe.mitre.org/data/definitions/674.html

4.2. Denial of Service (DoS) via Unbounded Alias Resolution

Severity	LOW
Affected Resources	Parser.php
Status	Closed

Update

The vulnerability was reported to the `security@symfony.com` mailing list. It was then moved to the GitHub advisory GHSA-4qpc-3hr4-r2p4 and assigned the CVE-2026-45304 number.

Description

The parser supports YAML anchors (`&foo`) and aliases (`*foo`) without limiting the number of expansions of the resulting structure.

When parsing a malicious YAML that defines a structure that expands exponentially (a "Billion Laughs" attack), this will force PHP to instantiate a huge amount of data, triggering an OOM error and crashing the application.

Impact

An attacker might use this vulnerability to crash the PHP process that uses the YAML parser.

Attack Complexity

The attacker would need to be able to feed arbitrary YAML input to the parser.

Related Issues

N/A

Proof of Concept

The vulnerability can be reproduced by passing the attached `test_billion_laughs.yaml` to the following test code:

```
# test_parser.php
<?php
use Symfony\Component\Yaml\Yaml;

$yamlFile = $argv[1];
$yamlContent = file_get_contents($yamlFile);

$result = Yaml::parse($yamlContent, 0);
json_encode($result);
```

The test can be simply executed by running:

```
php test_parser.php test_billion_laughs.yaml
```

Notice, in the stack trace, the error on `json_encode` when trying to allocate too many bytes.

It's worth mentioning that, in this case, the program does not crash on `parse`, but on `json_encode`. This is due to lazy allocation in PHP, so that the memory allocation – and thus the crash – only happens when the structure is visited to produce the string representation.

Suggested Remediations

Implement suitable limits when expanding components, and/or provide a flag to enable/disable anchors and aliases.

References

- https://cwe.mitre.org/data/definitions/776.html

4.3. Regular Expression Denial of Service (ReDoS) in Parser::cleanup

Severity	LOW
Affected Resources	Parser.php
Status	Closed

Update

The vulnerability was reported to the `security@symfony.com` mailing list. It was then moved to the GitHub advisory GHSA-9frc-8383-795m and assigned the CVE-2026-45305 number.

Description

In the `Parser::cleanup` function, used to trim spaces, headers and comments from the YAML content before parsing, the following regular expression is used to search and replace all the YAML headers:

```
#^\%YAML[: ][\d\.]+.*\n#u
```

In the regular expression, the `[\d\.]+` and `.*` quantified subpatterns can match the same input. With greedy quantifiers such as `+` and `*`, this can lead to "catastrophic backtracking", eventually leading to Denial of Service.

Impact

An attacker might abuse this vulnerability to stall or crash a PHP application using the Symfony YAML parser.

Attack Complexity

The attacker would need to be able to feed arbitrary YAML input to the parser.

Related Issues

N/A

Proof of Concept

The vulnerability can be reproduced by passing the attached `test_redos.yaml` to the following test code:

```
# test_parser.php
<?php
use Symfony\Component\Yaml\Yaml;

$yamlFile = $argv[1];
$yamlContent = file_get_contents($yamlFile);

$result = Yaml::parse($yamlContent, 0);
```

```
var_dump($result);
```

Suggested Remediations

Remove the overlap from the regular expression, or replace the greedy quantifiers with possessive quantifiers.

References

- https://owasp.org/www-community/attacks/Regular_expression_Denial_of_Service_-_ReDoS

4.4. Lack of Security Warning for Object Deserialization

Severity	INFORMATIONAL
Affected Resources	Inline.php
Status	Closed

Update

Documentation on the security risks attached to the deserialization were added on commit aeeba0f.

Description

By enabling the `Yaml::PARSE_OBJECT` flag, users can specify arbitrary PHP objects in YAML file, which will be then parsed by the component via the `unserialize` PHP function.

Arbitrary deserialization, in PHP, constitutes a serious risk that can be abused to compromise applications.

However, the documentation of the component does not mention the security issues associated with enabling object support in the parser.

Impact

Users unknowingly enabling parsing of custom object in Symfony YAML would expose their application to various risks, with the worst scenario being Remote Code Execution.

Attack Complexity

The victim application would need to enable object parsing by passing the `YAML::PARSE_OBJECT` flag to the parser, and the parser would have to accept untrusted, user-controlled YAML.

Related Issues

N/A

Proof of Concept

This is a code-level finding. The vulnerable snippet is reported below:

```
# Inline.php
case str_starts_with($scalar, "!php/object"):
    if (self::$objectSupport) {
        if (!isset($scalar[12])) {
            throw new ParseException(
                'Missing value for tag "!php/object".',
                self::$parsedLineNumber + 1,
                $scalar,
                self::$parsedFilename,
            );
```

```
        }

        return unserialize(
            self::parseScalar(substr($scalar, 12)),
        );
    }
```

Notice the usage of the unsafe `unserialize` function.

Suggested Remediations

Include a "security callout" in the documentation of the component, informing users of the attached risks.

References

- https://www.php.net/manual/en/function.unserialize.php

4.5. Lack of Security Warning for Constant Resolution

Severity	INFORMATIONAL
Affected Resources	Inline.php
Status	Closed

Update

Documentation on the security risks attached to the constant resolution were added on commit aeeba0f.

Description

By enabling the `Yaml::PARSE_CONSTANT` flag, users can specify constants in the YAML file, which will be resolved as proper PHP constants.

This process might be abused by attackers to leak information from the target system via built-in PHP constants, or even exfiltrate critical information from custom constants defined in the application (such as secrets).

However, the documentation of the component does not mention the security issues associated with enabling constant support in the parser.

Impact

Users unknowingly enabling parsing of constants in Symfony YAML would expose their application to various risks, with the worst scenario being exfiltration of sensitive secrets.

Attack Complexity

The victim application would need to enable constant parsing by passing the `YAML::PARSE_CONSTANT` flag to the parser, and the parser would have to accept untrusted, user-controlled YAML.

Related Issues

N/A

Proof of Concept

This is a code-level finding. The vulnerable snippet is reported below:

```
# Inline.php
case str_starts_with($scalar, "!php/const"):
    if (self::$constantSupport) {
        if (!isset($scalar[11])) {
            throw new ParseException(
                'Missing value for tag "!php/const".',
                self::$parsedLineNumber + 1,
                $scalar,
                self::$parsedFilename,
```

```
        );
    }

    $i = 0;
    if (
        \defined(
            $const = self::parseScalar(
                substr($scalar, 11),
                0,
                null,
                $i,
                false,
            ),
        )
    ) {
        return \constant($const);
    }
```

Notice that the constant is created by invoking `/constant`, which will successfully resolve any constant currently defined in the runtime.

Suggested Remediations

Include a "security callout" in the documentation of the component, informing users of the attached risks.

References

N/A

5. Attachments

The following is the list of attachments, together with their SHA256 integrity hashes:

5.1. *test_billion_laughs.yaml*

Link: https://github.com/ShielderSec/poc/blob/main/symfony-yaml/CVE-2026-45304.yaml

SHA256: `97dc82045e17b27376d3baf96aa4795f345735a8bfc4958c7bac23481bfc1b88`

5.2. *test_recursion.yaml*

Link: https://github.com/ShielderSec/poc/blob/main/symfony-yaml/CVE-2026-45133.yaml

SHA256: `f797a9e5f66b97e4b9e9fe0e73f2546f665a507bd68c11fe7bfb9caa764b9224`

5.3. *test_redos.yaml*

Link: https://github.com/ShielderSec/poc/blob/main/symfony-yaml/CVE-2026-45305.yaml

SHA256: `9d61d0606ae811ddef63ee3d7205badbb235ac984a65cbb9e2c21ca70909adbd`