

DOI: 10.15514/ISPRAS-2024-36(4)-2



Detecting Errors in the Pandas Software Module using the Svace Static Code Analyzer

M.A. Lapina, ORCID: 0000-0001-8117-9142 <mlapina@ncfu.ru>

M.I. Khodakov, ORCID: 0009-0007-6848-7184 <mxkhodakov@yandex.ru>

S.K. Grobova, ORCID: 0009-0000-2608-7712 <sofya.grobova@yandex.ru>

*North Caucasus Federal University,
1, Pushkina st., Stavropol, 355017, Russia.*

Abstract. The article deals with the urgent problem of software security at the early stage of its development. Special attention is paid to static code analysis, which is a key tool for detecting vulnerabilities at early stages of the software development life cycle. The article emphasizes the importance of integrating static analysis tools into the development process in order to detect and eliminate vulnerabilities early. The methods of static analyzers' error search are considered, as well as the main components of the Svace static analyzer developed at the Institute of System Programming of the Russian Academy of Sciences. Classification of analyses used in the Svace static analyzer is presented. Static analysis of source code in Python programming language is considered in detail. As a practical example the analysis of the Pandas 2.2.1 project performed with the help of Svace is given. The result was the detection of 241 vulnerabilities for 590709 lines of code, which shows a high density of warnings per million lines of code and confirms the effectiveness of static analysis in ensuring software security.

Keywords: static code analysis; static analyzer; lexical analyzers; lightweight analyzers; abstract syntax tree; code fragment.

For citation: Lapina M.A., Khodakov M.I., Grobova S.K. Detecting errors in the Pandas software module using the Svace static code analyzer. *Trudy ISP RAN/Proc. ISP RAS*, vol. 36, issue 4, 2024. pp. 17-26. DOI: 10.15514/ISPRAS-2024-36(4)-2.

Выявление ошибок в программном модуле Pandas с помощью статического анализатора Svacе

М.А. Лапина, ORCID: 0000-0001-8117-9142 <mlapina@ncfu.ru>

М.И. Ходаков, ORCID: 0009-0007-6848-7184 <mikhailhodakov@yandex.ru>

С.К. Грובה, ORCID: 0009-0000-2608-7712 <sofya.grobova@yandex.ru>

*Северо-Кавказский федеральный университет,
355017, Россия, г. Ставрополь, ул. Пушкина, д. 1.*

Аннотация. В статье рассматривается актуальная проблема обеспечения безопасности программного обеспечения на раннем этапе его разработки. Особое внимание уделяется статическому анализу кода, который является ключевым инструментом для выявления уязвимостей на ранних этапах жизненного цикла разработки программного обеспечения. Статья подчеркивает важность интеграции инструментов статического анализа в процесс разработки с целью раннего обнаружения и устранения уязвимостей. Рассмотрены методы поиска ошибок статических анализаторов, а также основные компоненты статического анализатора Svacе, разработанного в Институте системного программирования РАН. Представлена классификация анализов, используемых в статическом анализаторе Svacе. Детально рассмотрен статический анализ исходного кода на языке программирования Python. В качестве практического примера приведен анализ проекта Pandas 2.2.1, выполненный с помощью Svacе. Результатом послужило выявление 241 уязвимости на 590709 строк кода, что показывает высокую плотность предупреждений на миллион строк кода и подтверждает эффективность статического анализа в обеспечении безопасности программного обеспечения.

Ключевые слова: статический анализ кода; статический анализатор; лексические анализаторы; легковесные анализаторы; абстрактное синтаксическое дерево; фрагмент кода.

Для цитирования: Лапина М.А., Ходаков М.И., Грובה С.К. Выявление ошибок в программном модуле Pandas с помощью статического анализатора Svacе. Труды ИСП РАН, том 36, вып. 4, 2024 г., стр. 17–26 (на английском языке). DOI: 10.15514/ISPRAS-2024-36(4)-2

1. Introduction

Nowadays, software is used everywhere. Statistics shows that as the number of programs increases, so does the number of vulnerabilities in them. That's why special attention is paid to software analysis.

At present there are two most popular methods of code analysis – dynamic analysis and static analysis. It is impossible to imagine software development without using static analysis tools. Static analysis is a type of analysis when the program is not executed but its whole code is analyzed [1]. This method is most often implemented at the initial stages of development, which helps to detect vulnerabilities earlier and eliminate them faster. Static analysis of source code can be performed in two ways - manually and with the help of special software tools.

In manual analysis of source code, checks such as code review or code inspection are performed. This approach is actively used because a human is able to detect defects in source code that software analyzers cannot do yet.

Software analysis is performed using various analyzers. Many methods of static error search in programming have developed from the sphere of program compilation and operate with abstractions. Depending on the abstractions used, error search methods [2] can be divided into several groups:

- Lexical analyzers are designed to break down the source code of a program into individual tokens or tokens. They can be used to find only the simplest types of defects [3].
- Lightweight analyzers, also known as first-level analyzers, check the text for compliance with some grammar and build a parse tree (an abstract syntactic tree) by a linear sequence of tokens of this text, which is well suited for further processing and analysis of the text [4].

- More sophisticated parsers (Level 2 parsers), which use more complex algorithms and methods for parsing related to phases after syntactic analysis [5].

The disadvantage [6] of this method is that the analyzer may consider absolutely safe code fragments suspicious. Excessive suspiciousness leads to an increase of the false/true alarms ratio [7].

In this article we will consider the Svace static analyzer developed at the Ivannikov Institute for System Programming of the Russian Academy of Sciences (ISP RAS) and analyze the Pandas 2.2.1 project using it. This analyzer detects a large number of vulnerabilities in code, has a high level of true positives, fine-tuning and a large number of supported programming languages were the reasons for choosing this product.

2. Description of Svace

Svace is a tool developed at the Institute of System Programming of the Russian Academy of Sciences that performs static error search using several types of analyzers. It combines the key qualities of foreign analogs (Synopsis Coverity Static Analysis, Perforce Klocwork Static Code Analysis, Fortify Static Code Analyzer) with the unique use of open industrial compilers in order to maximize support for new programming language standards [8]. Supported languages are C, C++, C#, Java, Kotlin, Go, Python, Scala.

2.1 Svace analysis scheme

The schematic in (Fig. 1) shows the general analysis scheme of the Svace static analyzer [9].

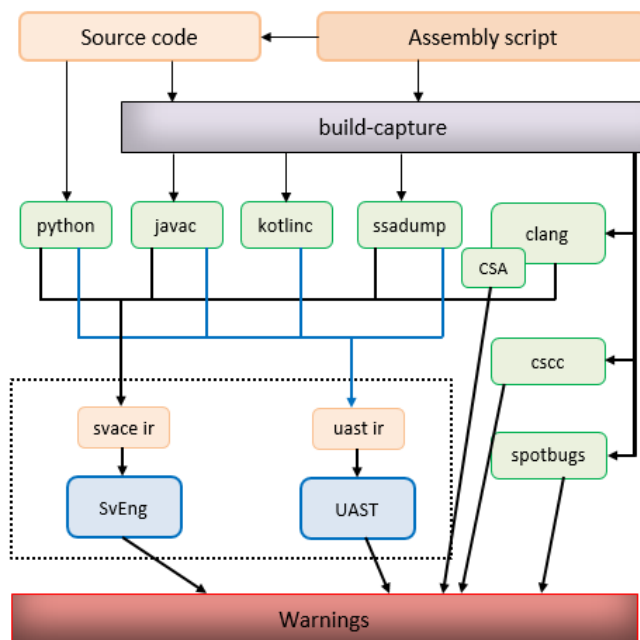


Fig.1. Schematic of the Svace static analyzer [9].

2.2 Svace components

The Svace static analyzer includes five components – SvEng, UAST, SpotBugs, CSA and csc.

SvEng is the main component of Svace. It uses the following types of static analysis: pattern search in AST-trees, data flow analysis, interprocedural symbolic execution with state merging and using method summaries, static analysis of labeled data for security errors.

UAST – component in which errors are searched in a unified abstract syntax tree.
SpotBugs - component intended for analyzing programs in Java programming language.
CSA – intended for analyzing software in C/C++ languages.
cscC – intended for analyzing programs in C# language. This component is based on the Roslyn compiler. For searching errors, it uses the same types of analysis as SvEng.

2.3 Classification of analyses used at Sspace

The analyses used in the Sspace static analyzer [9] can be divided into the following groups:

- Analyses based on pattern search in the abstract syntax tree (ASD);
- Interprocedural analyses based on summaries;
- Statistical analyses;
- Special function analyses;
- Symbolic execution-based analyses;
- Labeled function analyses.

3. Analyzing a Python project with Sspace

Analyzing the source code of a Python application does not require building the project because it is an interpreted language. Two components are involved in the analysis: SvEng and UAST.

Pandas was chosen as a project written in Python. 590709 lines of code were checked and 241 vulnerabilities were detected as a result of the analysis. The density of warnings per million code lines is 407.98. Table 1 shows the analysis results.

Fig. 2 shows a code fragment where the INVARIANT_RESULT warning was detected in the project Pandas pandas/tests/window/moments/test_moments_consistency_ewm.py. The essence of this error is that the value of the expression does not depend on the values of its parts, or a part of the expression can be omitted without changing the result.

Table 1. Results of the analysis.

Warning	Number of warnings
INVARIANT_RESULT	93
DIVISION_BY_ZERO.EX	23
BAD_COPY_PASTE	8
CATCH.NO_BODY	107
MUTABLE_DEFAULT_ARGUMENT	1
SIMILAR_BRANCHES	7
WRONG_ARGUMENTS_ORDER	2

```
35     if adjust:
36         count = 0
37         for i in range(len(s)):
38             if s.iat[i] == s.iat[i]:# INVARIANT_RESULT
39                 w.iat[i] = pow(1.0 / (1.0 - alpha), count)
40                 count += 1
41             elif not ignore_na:
42                 count += 1
```

Fig. 2. Error INVARIANT_RESULT.

Fig. 3 shows the `DIVISION_BY_ZERO.EX` error in the project Pandas `pandas/tests/io/parser/test_skiprows.py`. This error occurs whenever the program tries to divide an interval or a numeric value by 0.

The following types of errors were detected on the abstract syntax tree [10] using the component UAST (unified abstract syntax tree).

Fig. 4 shows a code fragment with `BAD_COPY_PASTE` error in the project Pandas `pandas/tests/extension/base/constructors.py`. An error occurs if the code has been copied and reproduced, but not all necessary changes have been made. Fig. 5 shows the `CATCH.NO_BODY` error in the project Pandas `pandas/tests/plotting/test_converter.py`. This error occurs if an exception was caught but not handled by the corresponding except block.

```
285 def test_skip_rows_bad_callable(all_parsers):
286     msg = "by zero"
287     parser = all_parsers
288     data = "a\n1\n2\n3\n4\n5"
289
290     with pytest.raises(ZeroDivisionError, match=msg):
291         parser.read_csv(StringIO(data), skiprows=lambda x: 1 / 0)
```

Fig. 3. Error `DIVISION_BY_ZERO.EX`.

```
28 if hasattr(result._mgr, "blocks"): # Original code
29     assert isinstance(result._mgr.blocks[0], EABackedBlock)
30     assert result._mgr.array is data
31
32
33 result2 = pd.Series(result)
34 assert result2.dtype == data.dtype
35 if hasattr(result._mgr, "blocks"): # Bad copy paste
36     assert isinstance(result2._mgr.blocks[0], EABackedBlock)
```

Fig. 4. Error `BAD_COPY_PASTE`.

```
37 try:
38     from pandas.plotting._matplotlib import converter
39 except ImportError: # CATCH.NO_BODY
40
41
42     pass
```

Fig. 5. Error `CATCH.NO_BODY`.

Fig. 6 shows a code fragment with the `MUTABLE_DEFAULT_ARGUMENT` error in the project Pandas `pandas/io/formats/style_render.py`. The error occurs if we use some modifiable object (list, dictionary) as a default value for a function argument. If we change the value of the argument inside the function, we will have to change the original value because they both refer to the same object, in this case it is the `css_props` argument. This can lead to unexpected results and errors in your code.

Fig. 7 shows a code fragment with `SIMILAR_BRANCHES_ARGUMENT` error in the project Pandas `pandas/tests/indexes/datetime/test_date_range.py`. This error occurs when executing the same instructions regardless of the condition.

Fig. 8 shows a code fragment with `WRONG_ARGUMENTS_ORDER` error in the project Pandas `pandas/_testing/__init__.py`. The error occurs when method arguments are passed in the wrong order. In this example, left and right are mixed up in the wrong order.

```
2154     def __init__(
2155         self,
2156         css_props: CSSProperties = [ #MUTABLE_DEFAULT_ARGUMENT
2157             ("visibility", "hidden"),
2158             ("position", "absolute"),
2159             ("z-index", 1),
2160             ("background-color", "black"),
2161             ("color", "white"),
2162             ("transform", "translate(-20px, -20px)"),
2163     ],
```

Fig. 6. Error `MUTABLE_DEFAULT_ARGUMENT`.

```
71     elif inclusive_endpoints == "both":
72         expected_range = both_range[:] # First branch
73     else:
74         expected_range = both_range[:] # Second branch
```

Fig. 7. Error `SIMILAR_BRANCHES`.

```
495     if isinstance(left, np.ndarray) and isinstance(right, np.ndarray):
496         return np.shares_memory(left, right)
497     elif isinstance(left, np.ndarray):
498
499         return shares_memory(right, left) # WRONG_ARGUMENTS_ORDER
```

Fig. 8. Error `WRONG_ARGUMENTS_ORDER`.

4. Classification of errors

Errors in software can be classified by the degree of their influence on the program operation [11]. Table 2 presents the table of error classification and their description. Errors obtained as a result of static analysis were classified by the level of their influence on the program code. Table 3 presents the error classification table.

The `DIVISION_BY_ZERO.EX` error belongs to the "Critical error" class because division by zero in the program code may cause unpredictable behavior of the command and eventually lead to its crash. Fig. 9 shows the scenario when this error occurs.

The `MUTABLE_DEFAULT_ARGUMENT` error belongs to the "Major error" class because it can cause unexpected results when using a variable data type as a default value for an argument. Fig.10 shows a scenario where this error occurs. In this code, the item function uses the variable data type (list) as the default value for the `item_list` argument. As a result, each time the `add_item` function is called, items are added to the same `item_list` instead of a new list. This is the `MUTABLE_DEFAULT_ARGUMENT` error.

The `WRONG_ARGUMENTS_ORDER` error is a "Major error" because if the order of arguments in a function call is incorrect, it can lead to unexpected program behavior. Fig. 11 shows the scenario of this error. In this example, the main function expects the first argument to be "x" and the second argument to be "y". However, when the function is called, these arguments are specified in reverse order, which causes an error.

The errors `INVARIANT_RESULT`, `BAD_COPY_PASTE`, `SIMILAR_BRANCHES`, `CATCH.NO_BODY` are not serious and belong to the "Minor error" class, but they require corrections to prevent unexpected program behavior.

Fig. 12 shows the `SIMILAR_BRANCHES` error scenario.

In this example, the if and else branches contain the same instructions $y = x * 2$, which may affect the efficiency of the program.

Table 2. Classification of errors and their description.

Error class	Description
Critical errors	This class of errors leads to an emergency situation that makes the program operation impossible.
Major errors	Errors that lead the program to unexpected results.
Minor errors	Errors that reduce the efficiency of the program and its performance.

Table 3. Classification of errors.

Class	Error name
Critical error	<code>DIVISION_BY_ZERO.EX</code>
Major error	<code>MUTABLE_DEFAULT_ARGUMENT</code>
	<code>WRONG_ARGUMENTS_ORDER</code>
Minor error	<code>INVARIANT_RESULT</code>
	<code>BAD_COPY_PASTE</code>
	<code>SIMILAR_BRANCHES</code>
	<code>CATCH.NO_BODY</code>

```
1 def divide_numbers(num1, num2):
2     return num1 / num2
3
4 def main():
5     values = [10,0]
6     for i in range(len(values) - 1):
7         print(f"division {values[i]} by {values[i+1]}")
8         outcome = divide_numbers(values[i],values[i+1])
9         print(f"Result: {outcome}")
10
11     if __name__ == "__main__":
12         main()
```

Fig. 9. Error scenario `DIVISION_BY_ZERO.EX`.

```
1 def item(item, item_list=[]):
2     item_list.append(item)
3     return item_list
4
5
6 print(item('apple'))
7 print(item('banana'))
```

Fig.10. Error scenario *MUTABLE_DEFAULT_ARGUMENT*.

One of the examples of the *BAD_COPY_PASTE* error is shown in Fig. 13. In this example, the calculate difference function should calculate the difference between variables a and b, but due to code copying, an error was made that may cause the program to work incorrectly.

The *CATCH.NO_BODY* error scenario is shown in Fig. 14. In this example, the exception is not handled in the except block, which may cause the error to be ignored and the program to continue execution without handling the error. Fig. 15 shows the *INVARIANT_RESULT* error scenario. In this example, the value of result1 does not depend on the variables x, y, z, because multiplication by 0 will always result in 0. The value of result2 will always be x + z because y - y is 0, so this part of the expression can be omitted without changing the result. Similarly with the value of result3, it will always be equal to x + y because z - z is 0.

```
1 def main(x, y):
2     return f"{y}, {x}!"
3
4 print(main("x", "y"))
```

Fig.11. Error scenario *WRONG_ARGUMENTS_ORDER*.

```
1 x = 10
2
3 if x > 5:
4     y = x * 2
5 else:
6     y = x * 2
```

Fig. 12. Error scenario *SIMILAR_BRANCHES*.

```
1 def calculate_sum(a, b):
2     return a + b
3
4
5 def calculate_difference(a, b):
6     return a + b
```

Fig.13. Error scenario *BAD_COPY_PASTE*.

```
1 try:
2     x = 1 / 0
3 except ZeroDivisionError:
4
5     pass
```

Fig.14. Error scenario *CATCH.NO_BODY*.


```
1 def calculate(x, y, z):
2
3
4     result1 = x * 0 + y * 0 + z * 0
5
6     result2 = x + y - y + z
7
8     result3 = x + y + z - z
```

Fig.15. Error scenario *INVARIANT_RESULT*.

5. Conclusion

As it was shown in the article, static analysis tools help detect errors at the initial stages of application development. Different static analyzers use different algorithms for detecting vulnerabilities in application source code. Svace static analyzer allows you to analyze program code in different languages. We also analyzed the project in Python, Pandas 2.2.1. In the process of analysis, we found errors of different classes. Classification of errors by their level of influence on the program code, their full description and examples of scenarios when these errors can occur in the program code were carried out. The results of the research may be useful for improving the quality and reliability of the product.

References

- [1]. Chernov D. Code analysis: problems, solutions, perspectives. 2022. URL: https://www.tadviser.ru/index.php/Статья:Анализ_кода:_проблемы,_решения,_перспективы#:~:text=У метода есть два недостатка,программы%2C который не всегда доступен, accessed 20.03.2024.
- [2]. Borodin A.E., Belevantsev A.A. Static analyzer Svace as a collection of analyzers of different levels of complexity. Proceedings of ISP RAS, vol. 27, issue 6, 2015, pp. 111-134. (in Russian). DOI: 10.15514/ISPRAS-2015-27(6)-8.
- [3]. Zabolova-Zotova A.V., Orlova Y.A. MODELING OF LEXICAL ANALYSIS OF TECHNICAL TASK TEXT. URL: <https://www.elibrary.ru/item.asp?id=9506673>, accessed 21.03.2024.
- [4]. Mavchun E.V. Comparison of algorithms of tabular bottom-up syntactic analysis. URL: https://oops.math.spbu.ru/SE/dip_loma/2015/s/544-Mavchun-report.pdf, accessed 24.03.2024.
- [5]. A. P. Syzranov, A. I. Nenakhov, A. Y. Bolotov, A. J. Osipov. Development of criteria for evaluation of means of automation of certification testing on the level of control of the absence of undeclared capabilities. URL: <https://elibrary.ru/item.asp?id=47856981>, accessed 24.03.2024.
- [6]. Fadeev S.G. TECHNOLOGY OF STATIC ANALYSIS OF SOURCE CODE. URL: <https://apni.ru/media/Sbornik-6-3.pdf#page=131>, accessed 20.03.2024.
- [7]. Kuchin I.Yu. Review of existing methods of program code analysis. URL: <https://cyberleninka.ru/article/n/obzor-suschestvuyuschih-metodov-analiza-programmnogo-koda/viewe>, accessed 21.03.2024.
- [8]. Svace static analyzer. URL: <https://www.ispras.ru/technologies/svace/>, accessed 24.03.2024.
- [9]. Svace components. URL: <https://svace.pages.ispras.ru/svace-website/2023/10/04/analysis-types.html>, accessed 24.03.2024.
- [10]. Afanasyev V.O., Borodin A.E., Vikhlyantsev K.I., Belevantsev A.A. Static analysis based on generalized abstract syntactic tree. Proceedings of ISP RAS, vol. 35, issue 6, 2023, pp. 103-120. (in Russian). DOI: 10.15514/ISPRAS-2023-35(6)-6.
- [11]. V.V. Bykova, G.E. Glukhov, A.N. Sharypov, P.E. Chernikov, S.V. Koval, A.Yu. Konkov. PROBLEMS OF VULNERABILITY OF INFORMATION SYSTEMS OF AVIATION INDUSTRY ENTERPRISES: ANALYSIS AND CLASSIFICATION OF ERRORS. URL: <https://mlgvs.ru/files/iac/art-niiga-2019-27.pdf>, accessed 01.04.2024.

Информация об авторах / Information about authors

Мария Анатольевна ЛАПИНА – кандидат физико-математических наук, доцент, доцент кафедры информационной безопасности автоматизированных систем Северо-Кавказского федерального университета. Сфера научных интересов: цифровые технологии, киберфизические системы, анализ данных, управление информационной безопасностью, доверенный искусственный интеллект, криптография, анализ программного кода.

Maria Anatolyevna LAPINA – Cand. Sci. (Phys.-Math.), Associate Professor, Associate Professor of the Department of Information Security of Automated Systems of the North Caucasus Federal University. Research interests: digital technologies, cyber-physical systems, data analysis, information security management, trusted artificial intelligence, cryptography, program code analysis.

Максим Иванович ХОДАКОВ – студент кафедры информационной безопасности автоматизированных систем Северо-Кавказского федерального университета. Сфера научных интересов: программирование, цифровые технологии, анализ программного кода.

Maxim Ivanovich KHODAKOV – student of the Department of Information Security of Automated Systems of the North Caucasus Federal University. Research interests: programming, digital technologies, program code analysis.

Софья Кирилловна ГРОБОВА – студентка кафедры инфокоммуникаций Северо-Кавказского федерального университета. Сфера научных интересов: программирование, цифровые технологии, анализ программного кода.

Sofya Kirillovna GROBOVA – student of the Department of Infocommunications of the North Caucasus Federal University. Her research interests: programming, digital technologies, program code analysis.