# Program Analysis Report

### Introduction:

There is a large suite of Program Analyzer (PA) programs all that are intended for studying the correctness and soundness of programs. Program consist of multiple levels of source files, many of which can consist of different file extensions such as C++, Java, or Python. In this project, we explore the application of three different program analysis tools onto a potentially vulnerable application and study the accuracy rate between different tools.

### Problem:

Different tools provide different ranges of implementation and different spheres of completeness study. Modern-day program analyzers have different offerings for studying source code as well as detailing the results. This study will observe the different presentations, installation processes, and end-results provided by the tools.

### Contribution:

This project will study the following parts:

* Test allegations, features, and testing capabilities from program analyzers
* Learn more about how to use and apply different PA tools

### Experiment Setup:

This project will focus on program analysis tools aimed specifically for Windows, open source, and free ready-to-use. To test these programs, a popular application with open source code and a large repository of files to scan through will be utilized as the benchmarking application. In this paper, the 3D modeling software Blender was used to run the analysis tools due to its use of C++ and Python files.

Two versions of Blender was utilized, one unmodified from the GitHub repository and a second version with bugs introduced. These introduced bugs were:

* One C++ and C file altered to contain two memory leaky functions and a null pointer
* One Python script with unused and unreferenced values

The system tested was on a Windows 10 Home edition laptop, with 16GB of RAM, and with a 3.4Ghz CPU.

Three PA tools will be utilized for checking the application. These will be:

SonarQube: <https://docs.sonarqube.org/latest/analysis/scan/sonarscanner/>

This is a PA tool that is used for in-depth study of multiple file extensions. It can be ran on Windows and provides a detailed report for errors and “code smells” for code found to have errors. This program is proprietary but utilizes a client-side server to provide a GUI for the user to read through the errors found. This tool can analyze C++ and Python scripts, however for this project, we can only analyze python scripts on the freemium version of SonarQube. <https://www.sonarqube.org/>

**DevSkim:** [**https://github.com/microsoft/DevSkim/wiki/Download-and-Install**](https://github.com/microsoft/DevSkim/wiki/Download-and-Install)

DevSkim is a framework that is used as an IDE extension that is used by developers for real-time code examination and for locating vulnerabilities as core features are being implemented. This tool can be used for static analysis for full-project checking. The use of this tool is limited to utilizing the C++ analysis.

**CppCheck:** [**http://cppcheck.sourceforge.net/**](http://cppcheck.sourceforge.net/)

This is a tool used for general program analysis and deep exploration for program errors and faults. It can be used to automatically locate most program faults from a varierity of programming languages such as C++ and python. For this project, we can perform a full analysis of the project and generate a detailed report of errors and faults for both of the application languages.

**Preliminary Blender installation**: <https://wiki.blender.org/wiki/Building_Blender/Windows>

Since the GitHub repository is a recurrently updated repository, bugs are expected to be include in the repository. As such, this is a great candidate for testing for potential errors, bug smells, and faults. One guarantee of this strategy is that there will be false positives to be expected. Blender includes an instructive guide for installing the github repository at the following link:

### Results:

**SonarQube Results:**

SonarQube provides a suite that show step by step evolution of the code through different code repository pushes. Overall, it’s an easy tool to understand, however it is limited by the freemium version to only analyze python files. Developers are notified if the code is vulnerable to exploits through python scripts and provided line-by-line readout of the exploit as well as a quick-fix option. The following table shows a breakdown of the errors:

Figure 1 Report after modifying the code

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Bugs | Critical Issues | Code Smells | Vulnerabilities |
| Unmodified | 1 | 0 | 17 | 0 |
| Modified | **3** | **4** | **19** | **0** |

**DevSkim Results:**

DevSkim was built as a framework to be installed on top of another IDE or framework and provide real-time static analysis of the code as it’s being written. In this project, it was chosen to be ran after the code was completed and can check C++ code. This program focuses on detecting deprecated C++ styling and avoiding exploits through SSL/TLS. As a result, a large amount of the printout detected by DevSkim has been information regarding these two concerns. It did not detect bugs despite deliberate introduction to multiple files. The following table breaks down the data provided by DevSkim’s PA tool.

|  |  |  |
| --- | --- | --- |
|  | Critical Bugs | Important Bugs |
| Unmodified | 4 | 599 |
| Modified | 4 | 599 |

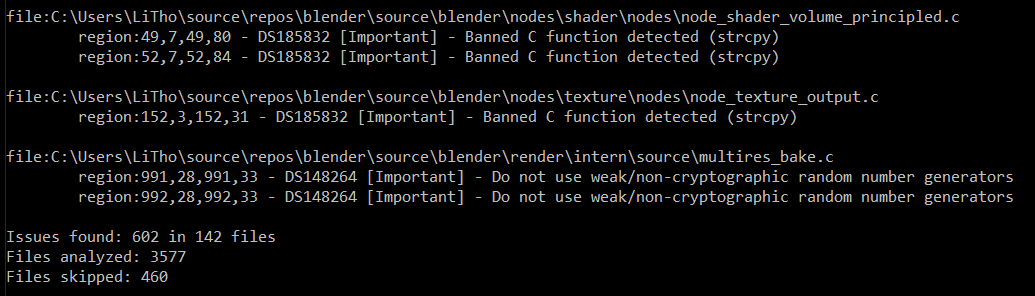


Figure DevSkim Results after execution

**CppCheck Results:**

This was the slowest program analyzer of the three examples shown in this paper. This one was the easiest PA to install and setup, however actual code analysis was more complicated to perform. CppCheck had issues with the type of computer it was installed at first and had additional issues with configuration/performance issues – crashing after 50% of the code base was analyzed through the GUI provided. A work-around was settled by using the terminal version of CppCheck and manually checking the output. Manually adding bugs did not change the bug count nor did it manage to locate the bugs. Below is the breakdown of the errors/warnings found by the tool of C/C++ files with results redacted to make space for the more common errors (Syntax, null pointers, etc.):

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Bug Type** | Variable Scope | Unknown Macro | Memory Leak | Invalid Pointer Cast | Null Pointer Redundant Check | Redundant  Initialization | Syntax Error |
| Unmodified | 2676 | 7 | 0 | 99 | 144 | 23 | 7 |
| Modified | 2676 | 7 | **1** | 99 | 144 | **25** | 7 |

### 

Figure 3 Code used to Execute CppCheck from the Terminal

### **Conclusion:**

Different tools solve different needs as seen by the results above. SonarQube provides long-term reports for developers to understand the potential vulnerabilities and issues that might be introduced. DevSkim is a very over-view type of tool which only explores code styling and potential exploits occurring in C code. CppCheck provides in-depth