# Statically Detecting Likely Buffer Overflow Vulnerabilities

David Larochelle and David Evans
USENIX'01
IEEE Software Jan/Feb 2002

Adam Polyak
30.03.2014

The paper presented an interesting approach for detecting possible buffer overflows vulnerabilities in a program by statically analyzing its source code. The tool presented in the article, Splint, leverages programmer's annotations (semantic comments) for the static analysis. To make the analysis lightweight and scalable, Splint makes some compromises. As a result, Splint is unsound and incomplete. Nonetheless, Splint still produces useful results.

As said, Splint uses annotations, which are added to the source code as regular C comments, for example /*@notnull@*/. When Splint analyzes the annotated source code, it generates **constraints** and function **preconditions** and **postconditions** from C statements about buffer usage. The conditions are used to make assumptions about function parameters and constrain function return value.

Splint analysis is done for each function separately by traversing the function code. It issues a warning when constraints are violated or a function is called without satisfying its **preconditions**. The check proceeds by assuming the **postconditions** are true. This way each function body is done separately. Complex structures such as loops and conditions are analyzed using heurists. Using intraprocedural analysis and heuristics enables Splint to be lightweight and scalable.

Splint is neither sound nor complete but evaluation performed on it suggests that it is useful. Although using Splint requires adding annotations – it is not an unreasonable effort, especially for security-sensitive programs. In addition, annotations improve the program documentation and maintainability.

In my opinion, Splint presented a great practical approach which represents the 80\20 principle; it is not a perfect solution but it is relatively simple, and shows good results. Although it is not accurate, Splint may be still used for generating feedback for developers.

In the presentation, I showed how Splint may be used as part of **Continuous Integration**. CI is practice in software development which allows fast feedback for developers. Static analysis can be used to ensure coding standards and avoid common bugs (such as buffer overflows).

During the discussion after the presentation the following points came up:

- Regarding Splint:
  - Splint is an open-source project, which was last updated in 2008. This is another disadvantage although not directly related to the article
  - For existing large code bases, Splint can be hard to use as it requires adding annotations to the source code. This requires a great amount of effort from the programmer.
  - Splint is useful to enforce consistency between documentation and code, as annotations are a form of documentation.
- Regarding general static analysis:
  - The notion of adding advance static analysis (like Splint) to compilers was suggested. Several challenges exists before that could be done:
    - Static analysis is an undecidable problem - there is no sound solution
    - Currently, there isn't any scalable solution with good enough results (Splint for example is efficient but is unsound and incomplete)
  - To overcome problems of unsounds like in Splint, many static analysis tools employ "bug ranking" – this way filtering critical bugs from all possible bugs found. In doing so, tools increase their credibility.
  - Static analysis is useful for dynamic languages, as an example: eval command in JavaScript imposes a severe security issue. Using static analysis can help to detect such vulnerabilities.