UDC 004.056

ENSURING INFORMATION SECURITY IN SOFTWARE SYSTEMS THROUGH PRELIMINARY STATIC CODE ANALYSIS

Elenhaupt V. V.
Scientific supervisor – Dr. Sci., Prof.. Antipov I. E.
Kharkov National University of Radio Electronics, CRETISS dep.
vitalii.elenhaupt@nure.ua

This work is about developing a static code analyzer for the Crystal programming language to identify and prevent potential vulnerabilities. Crystal is a young and promising programming language that offers modern tools for creating reliable software. In this work, we look at how static analysis can help identify and fix weaknesses in code that could lead to information leaks, encryption vulnerabilities, and other potential security issues.

Considering modern challenges, we note the constant growth of cyber attacks and the their forms diversity. In this context, special attention should be paid to the static code analysis use. Static analysis allows to identify vulnerabilities at early stages of development, ensuring the ability to eliminate them in a timely manner. This approach is an integral part of developing reliable and secure software products.

The Crystal language uses static typing, which ensures type safety and maintains a high level of development flexibility. This is achieved due to the fact that it does not require explicit specification of types.

Unfortunately, the number of tools for finding vulnerabilities for the Crystal language is still limited due to its newness. Traditional methods such as DAST, Penetration testing and bug bounty programs can be applied to applications developed on Crystal, but they have their limitations.

After analyzing the features of Crystal, we can conclude that the task comes down to building a static analyzer using the internal mechanisms of the Crystal compiler to create an AST tree. Fortunately, the Crystal programming language provides access to its internals, making it easy to build an AST tree for any program. This greatly simplifies our task.

Since there are many rules covering different aspects of security, in this work we focus on developing rules that can guarantee:

1. Secure processing and storage of information
2. Data encryption control.
3. Protection against information leakage during logging.
4. Secure data processing during data serialization and deserialization.

The feature of static analysis it is based on analyzing code without executing. Most programming languages, including Crystal, have tools for representing program code as an AST tree. An AST, or abstract syntax tree, is a tree structure that represents the syntactic code structure. Each node in the AST represents a separate program element, allowing to analyze its components in de-

tail. A tree node can be any programming language construct: a class, function, function call, variable, assignment, multiplication or addition operation, etc. To traverse the AST tree in the Crystal programming language, the Visitor design pattern is used. It is this that allows the compiler and other programs to execute different algorithms on the same data structure without conflicts.

Let's give an example of using static code analysis to identify potential vulnerabilities associated with data processing in software systems. Thus, when working with databases, there is a SQL Injection vulnerability. This is a type of attack, where a hacker can execute malicious SQL on an electronic database through user input into software systems.

For example, the value of the *user_id* variable can be inserted into a SQL without any processing or escaping. This means that the user can enter any SQL code and it will be executed by the database. This, in turn, can lead to information leakage, since if the hacker finds the necessary conditions, the generated query will select all users from the table in the electronic database. To identify this dangerous code using static analysis, you need to perform several operations:

1. Analyze the data flow in the application, tracing incoming data from the user to places where it could be used in a dangerous way, such as in SQL queries.

2. Find the use of methods that can perform SQL queries with data received from the user. In our case, this is the execute method.

3. Check that user input is processed before being used in SQL queries. If the data is inserted directly, then it is obvious that we have discovered a vulnerability. Using the AST syntax tree and the traversal method described above, we can easily find the corresponding tree nodes that could lead to a SQL Injection vulnerability.

Checks for other types of vulnerabilities are performed in a similar way.

The results of users survey of this software product are follows: common effectiveness is estimated at 7.00 (out of 10) points. At the same time, SQL Injection detection effectiveness is 8.20 points, the problems with encryption detection - 6.20 points, the leaks during logging detection 5.20, the serialization and deserialization problem detection 4.20 points. In general, the survey results indicate the high effectiveness of the software product.

References:

1.OWASP. (2021). OWASP Top Ten Project. Retrieved from https://owasp.org/www-project-top-ten

2. Chess, B., & West, J. (2007). Secure Programming with Static Analysis.

3. Zalewski, M. (2011). The Tangled Web: A Guide to Securing Modern Web Applications.