Journal of Computing and Electronic Information Management
ISSN: 2413-1660 | Vol. 12, No. 1, 2024

Securing Supply Chains in Open Source Ecosystems: Methodologies for Determining Version Numbers of Components Without Package Management Files

Li Sun

QAX Technology Group Inc., Beijing, China
Li_Sun_1984@163.com

Abstract: In the case of supply chain security detection research, determining the component version number is a crucial task for the open source components of package-free management files. This paper aims to explore the new perspective of the determination of component version numbers based on various methods and to propose an effective method. First, by analyzing the source code of the component, you can try to determine the version number of the component by a specific mode, function, or variable in the code. This approach requires in-depth study and analysis of the source code to extract key code snippets that may contain version information. Second, the submission history of the component can be used to track the change of the version number. The modification content and update information for each version is obtained by viewing the submission records of the components in the version control system. Such an approach is relatively feasible for those components with a canonical versioning history. In addition, the metadata or metadata information of the component can be used to determine the version number. Some open-source components may contain version-related information in their code or documentation, such as release date, release instructions, version labels, etc. By parsing and extraction of these metadata, the version number of the components is obtained. In addition, the version number of the component can be obtained through communication with the community or the developer. Participate in the relevant open source community or contact component developers to consult them for information about the component version. This approach may require more time and resources, but is a viable option for those components that are difficult to determine the version number through other means. To sum up, the determination of the version number of open source components without package management files is an important link in supply chain security detection.

Keywords: Supply chain security detection; Non-package management files; Open source components; Component version number determination method.

1. Introduction

In today's digital age, supply chain security has become a major concern. Supply chain security refers to ensuring the security of all links in the entire supply chain network to prevent potential threats and risks. Among them, open source components play an important role in software development, providing developers with rich functions and resources. However, for security and reliability issues for open-source components, especially for those without package management files, version number determination becomes a challenging task. Establishing the version number of open source components is critical for supply chain security detection. Version numbers not only help developers understand the functions and features of components, but also access security updates and patches in a timely manner to reduce potential vulnerabilities and risks. However, determining the version numbers becomes more complex and difficult for open-source components without package-managed files. Therefore, this paper will explore the DNS security detection research based on threat intelligence and data statistical analysis to solve the problem of version number determination of open source components of package-free management files. By analyzing the source code of the components, viewing the submission history, exploiting the metadata of the components, and communicating with the community and developers, we will explore an effective way to determine the version numbers of these components. The goal of this study is to provide a feasible and effective method for version number determination to help developers to better manage and evaluate the security of open-source components. Through this research, we will provide a new perspective for the supply chain security detection, and contribute to protecting the security of the digital supply chain.

2. Open source ecosystem and supply chain security

2.1. Overview of open Source software Supply chain

With the rapid development of information technology, open source software has become an important part of the modern software supply chain. The open source software supply chain involves a number of links, from the initial code development, version control, to release, deployment and post-maintenance. Each link has an impact on the overall quality and safety of the software.

The open source software supply chain starts in the code development stage. At this stage, developers work together to write and modify software code through an open and transparent collaborative approach. This approach allows open source software to quickly iterate and optimize, while also attracting a large number of users and contributors.

Version control is a key link in the open-source software supply chain. With a version control system (such as Git), developers can track and manage the change history of the code to ensure code consistency and traceability. In addition, the version control system also supports code merging,

branch management and other functions, to improve the development efficiency.

Release and deployment is a necessary step to bring open source software to market. At this stage, the open source software is compiled, packaged, and tested, and is released to a public software warehouse or distribution warehouse. Users can download, install, and use the software from these warehouses. To ensure software security and stability, strict quality control and safety audits are required during the release and deployment phase.

Later maintenance is a continuous link in the open source software supply chain. At this stage, developers maintain and update the published software, fix vulnerabilities, add new features, etc. In addition, user feedback and problem tracking are also important parts of the maintenance process. Through close interaction with users, developers can understand their user needs and continuously optimize the software.

Security and stability are important considerations in the open source software supply chain. Since the source code of open source software is public, any potential security vulnerability can be exploited. Therefore, developers, users, and security experts need to work together to ensure the security of open-source software.

In short, the open-source software supply chain is a complex system involving multiple links. From code development to post-maintenance, each link has an impact on the final quality and safety of the software.

2.2. Security risks and threats in the supply chain

Security of the supply chain is becoming increasingly important in today's software development and deployment environments. As an important part of the supply chain, the impact of the security of open source software on the whole system cannot be ignored. However, due to the openness and sharing of open source software, it also faces various security risks and threats.

First, the security risks in the supply chain may originate from multiple aspects. Among them, code vulnerabilities are one of the most common security risks. Since the source code of open source software is public, any potential security vulnerability can be exploited by malicious attackers. In addition, open-source software dependencies can also pose security risks. If an open-source component has a security vulnerability, then other software using that component may also be affected.

Second, the threats in the supply chain can also be diverse. Among them, code injection is a common threat. An attacker may inject malicious code into the open-source software to control or destroy the software's behavior. In addition, threats in the supply chain include malware, phishing attacks, etc. These threats could exploit vulnerabilities in the supply chain and have a serious impact on the entire system.

To deal with these security risks and threats, we need to take a series of measures to strengthen the security of the supply chain. First, enhanced code review and testing is necessary. A combination of automated tools and manual review, potential security vulnerabilities can be identified and fixed in time. Second, building safe dependencies is also the key. Developers and maintainers should regularly check and update dependencies to ensure the latest, safest components. In addition, strengthening safety training and awareness promotion are also necessary. Developers and users should be aware of common security risks and threats, and take precautions accordingly.

In short, there are many security risks and threats in the supply chain, and we need to take effective measures to strengthen security. By strengthening code review and testing, building security dependencies, and strengthening security training and awareness enhancement, we can reduce the impact of security risks and threats and ensure the security of the supply chain.

2.3. Current supply chain security solutions and their limitations

With the rapid development of information technology, the supply chain security problem pays more and more attention. To keep the supply chain safe, many solutions have emerged. However, these solutions also have some limitations. A detailed analysis of the current supply chain security solutions and their limitations is presented below.

First, security auditing of each node in the supply chain is a common solution. This scheme ensures that each node meets the safety standards by regularly checking and evaluating the safety of each node. However, such programmes are time consuming and resource intensive and may not detect security threats. In addition, for the supply chain of open source software, the difficulty and cost of security audit will further increase due to its openness and dynamics.

Second, it is also common practice to use security tools and frameworks to enhance supply chain security. For example, use automated tools such as code review tools and vulnerability scanning tools to detect potential security issues. These tools can improve development efficiency and safety, but there may also be false reports and underreports, which need to be verified in combination with manual review and testing.

In addition, the establishment of the security dependence relationship is also one of the important measures to ensure the security of the supply chain. By managing and controlling dependencies, you can ensure that the components used are safe and reliable. However, for the supply chain of open-source software, building secure dependencies is more complex. Due to the large number of open source components and frequent version updates, it is difficult to fully manage and control all dependencies.

In addition, strengthening safety training and awareness promotion is also one of the important measures to ensure supply chain security. By improving the security awareness and skills of developers and users, the security risks and the impact of threats can be reduced. However, the effectiveness of such solutions often depends on the individual's awareness and skill level, and it is difficult to fully cover all aspects of the supply chain.

To sum up, although the current supply chain security solutions have achieved some results, they still have some limitations. In order to ensure the security of the supply chain more comprehensively, it is necessary to combine with a variety of solutions, and strengthen technological innovation and international cooperation to jointly meet the challenges of supply chain security.

3. Determine the method of having no package management file component version number

3.1. Methods based on code-based static analysis

Code static analysis is a technique that detects potential problems in code without the need to actually execute code. In the field of supply chain security, the method based on code static analysis has become an important solution. The methods based on code static analysis are presented in detail below.

First, the method based on code static analysis finds potential security vulnerabilities and errors by scanning and analyzing the source code. The advantage of this approach is that it can detect problems during the code development stage, avoiding the maintenance costs and risks of finding problems at a later stage.

Common methods based on code static analysis include rule matching, semantic analysis, control flow analysis, etc. Rule matching is the simplest way to find potential problems by defining a set of rules to match specific patterns in the source code. Semantic analysis goes further by understanding and analyzing the semantics of the code to determine its correctness and security. Control flow analysis finds potential logic errors and security vulnerabilities by analyzing the control flow chart of the code.

Methods based on code-based static analysis have many advantages. First, it can detect problems during the code development phase and avoid increasing later maintenance costs. Secondly, static analysis with automated tools can quickly scan and analyze a large amount of code to improve the detection efficiency. In addition, static analysis can also find some problems that are difficult to find in dynamic analysis, such as some logic errors and data access problems.

However, methods based on static analysis of code have some limitations. First, false reporting and underreporting are common problems. Some static analysis tools may misreport some normal code as a potential problem, or miss some real problem. Second, for some complex code structure and algorithms, the accuracy and reliability of the static analysis may be affected. In addition, static analysis requires a certain amount of computational resources and time, which may bring some performance overhead for large-scale code bases.

To improve the accuracy and reliability of static analysis based on code, some improvement methods have been proposed. For example, using multiple static analysis methods for combined detection, combining the advantages of methods such as rule matching, semantic analysis and control flow analysis. In addition, data mining and pattern recognition of static analysis results using machine learning and artificial intelligence technologies can further improve the detection accuracy and efficiency.

3.2. Methods based on dependency relationship analysis

In the software supply chain, the dependence between the components is one of the important factors affecting the security. The approach based on dependency analysis is an effective supply chain security solution to discover potential security risks and vulnerabilities by analyzing the security relationships between components.

Methods based on dependency analysis focus on dependencies between software components, including direct and indirect dependence. Direct dependencies are direct call or reference relationships between components, while indirect dependencies are dependencies passed through other components. By analyzing these dependencies, potential security risks and vulnerabilities can be identified.

Methods based on dependency analysis can be implemented by static or dynamic analysis. Static analysis is to analyze dependencies without executing code, finding potential problems through techniques such as code review, abstract grammar tree (AST) analysis, and data flow analysis. Dynamic analysis identifies security issues by monitoring interactions between components at runtime.

The method based on dependency analysis has the following advantages:

Comprehensiveness: It can comprehensively analyze the dependencies between software components to identify potential security risks and vulnerabilities.

High efficiency: Rapid detection of extensive code through dependency analysis with automated tools.

Flexibility: It can combine static analysis and dynamic analysis to choose the appropriate method according to the specific needs.

However, the method based on dependency relationship analysis also has some limitations:

Complexity: Dependencies between components can be very complex and require a deep understanding of the logical structure and behavior of the software.

Misreporting and underreporting: Due to the diversity and complexity of dependencies, misreporting and underreporting may occur.

Performance overhead: For large-scale code libraries, dependency analysis can bring some performance overhead.

In order to improve the accuracy and reliability of dependency-based analysis, future studies can further explore improved methods and technologies, such as combining machine learning and artificial intelligence technologies for data mining and pattern recognition, or using more advanced code analysis techniques to understand the logical structure and behavior between components.

3.3. Method to determine the version number based on GitHub full text data

In the open source software supply chain, the determination of component version number is crucial to ensure software quality and security. However, many open source components do not explicitly mark the version numbers when they are released on code hosting platforms such as GitHub, which brings inconvenience and potential security risks to users. To this end, methods for determining component version numbers based on GitHub full-text data are becoming increasingly important.

The method mainly utilizes the full-text search function provided by GitHub and the project management function of the open-source components. By entering critical version information, such as specific strings or numbers, in GitHub full-text searches, we can tentatively identify the relevant code repository. Next, further analysis of the submission history, labels, and release instructions of these code warehouses can help us more accurately determine the version number of the components.

Full text search: Enter possible version information, such as a specific string or number, in the GitHub full text search box. This can preliminary screen out the code warehouse

containing the relevant information.

Screening and positioning: Based on the search results, the warehouse related to the target components. This may require some domain knowledge or experience, and a knowledge of the target components.

Analyze submission history: view the submission history of the selected warehouse for submission records related to the version number. This may involve specific version labels, submission notes, or dates, etc.

Check the release instructions: Some open source projects will clearly mark the version number in the warehouse release instructions or documents. This information is critical for determining the component version number.

Verification and Confirmation: To verify the accuracy of the preliminary version number by comparing other reliable sources or official documents.

This method has some utility and effectiveness, especially for those open source components that do not explicitly annotate version numbers. However, this method does not guarantee 100% accuracy due to potentially incomplete or misleading information on GitHub. Therefore, when using this method, other sources of information and verification means should be combined to ensure the accuracy of the identified version number.

4. Experimental design and evaluation

4.1. Data set preparation and experimental environment setup

In machine learning and data analysis, data set preparation and experimental environment setup are crucial steps. To ensure the accuracy and repeatability of the experiments, we need to follow certain specifications and procedures.

Dataset preparation

Dataset preparation is the first step in the whole machine learning process, and its quality directly affects the training and prediction effect of subsequent models. Here are some key steps:

Data collection: Determine the required data sources based on the research question. This may include publicly available datasets, private data sets, or data collected through tools such as sensors.

Data cleaning: This step is critical because the raw data often contains noise, missing values, or outliers. Appropriate strategies are needed to address these issues, such as filling in missing values, removing outliers, or performing data transformation.

Data preprocessing: For machine learning algorithms to work better, the data may need to be standardized, normalized, or encoded. Feature selection is also an important part of preprocessing that facilitates the removal of irrelevant or redundant features.

Data segmentation: The data set is divided into training set, verification set and test set, usually according to the ratio of 70-15-15. This evaluates the performance of the model on unseen data.

Experimental environment construction

The experimental environment determines the efficiency of machine learning projects and the replicability of the results. Here are some of the key components:

Hardware configuration: Select the appropriate hardware, such as CPU, GPU, and memory, according to the project requirements. For large-scale datasets and complex models, high-performance hardware is necessary.

Software installation and configuration: select the appropriate operating system and install the necessary software package. For example, the Anaconda or Miniconda of Python is a popular scientific computing environment containing a large number of libraries.

Environmental isolation: Use virtual machines or Docker containers to isolate different experimental environments, ensuring that each project has consistent dependencies and configuration.

Version control: Use Git or other version control systems to track code changes for backtracking and collaboration.

Code and Data Management: To manage code and large data sets with appropriate storage solutions. Cloud storage, network file systems, or distributed storage systems are all viable options.

Experimental configuration and logging: Set the appropriate parameter range and hyperparameter adjustment strategy, and record all experimental configurations and results at the same time for subsequent analysis and comparison.

4.2. Evaluation indicators and methods

Evaluation indicators:

Accuracy (Accuracy): In the classification task, the accuracy is the proportion of samples that are correctly predicted by the model.

Accuracy (Precision): The proportion of samples correctly predicted to be positive at a particular threshold.

Recall (Recall): the proportion of all positive samples correctly predicted to be positive.

F1 score: harmonic mean of precision and recall for comprehensive assessment of model performance.

AUC-ROC (Area Under the Curve-Receiver Operating Characteristic): The area under the ROC curve that measures the ability of the model to distinguish between positive and negative samples.

MSE (Mean Squared Error): measures the average of the mean square error between the predicted value and the true value in the regression task.

RMSE (Root Mean Squared Error): the square root of the MSE that provides an estimate closer to the actual value.

method:

Cross-validation (Cross-Validation): divide the data set into k parts, use k-1 part for training each time, remaining 1 part for testing, repeat k times.

Leave method (Holdout): Part of the data set is used for testing and the rest is used for training.

Self-help method (Bootstrap): Training and test sets are constructed using sampling with replay.

5. Conclusion

Through this study, we explore the component version number determination method for the open-source components of packet-free management files. In supply chain security testing, it is critical to determine the version number of components to obtain timely access security updates and patches to reduce potential vulnerabilities and risks. We propose several effective methods to determine the version number of the open-source components of package-free management files. First, by analyzing the source code of the component, we can find specific patterns, functions or variables in which to infer the version number of the component. This requires in-depth study and analysis of the source code to extract key code snippets that may contain

version information. We can use the submission history of the component to track the version number changes. The modification content and update information for each version is obtained by viewing the submission records of the components in the version control system. This approach is relatively feasible for components with a canonical versioning history. We can use the metadata or metadata of the component to determine the version number. Some open-source components may contain version-related information in their code or documentation, such as release date, release instructions, version labels, etc. Through the parsing and extraction of these metadata, the version number of the components is obtained. By communicating with the community or developers, we can obtain the version number of the components. Participate in the relevant open source community or contact component developers to consult them for information about the component version. Although this approach may require more time and resources, it is a feasible option for those components that are difficult to determine the version number through other means. We propose multiple methods to determine the version number of the open-source components of package-free management files.

References

[1] Zhang Xiaofei, Peng Hua, Chen Beibei. "A method of open source package-free management file based on code variation analysis." Journal of Software, 2021,32 (6): 1667-1682.

[2] Li Wenyan, Liu Fang, Zhao Feng, et al. "Analysis method of open source components of package-free management files based on vulnerability mining." Software Engineering and Application, 2020,49 (10): 86-91.

[3] Guo Ming, Zhang Lin. "Review of version tracking and determination technologies for open source components of package-free management files." Computer Application Research, 2019,36 (11): 3218-3225.

[4] Wang Yafei, Zhou Yunfei, Zhang Yu. "A method of open-source component identification based on code static analysis." Computer Engineering and Design, 2018,39 (12): 3142-3147.

[5] Li Ting, Hu Dongsheng, Zhang Baoqing, et al. "Study on the determination method of open source components of package-free management files." Computer Science and Exploration, 2017,11 (11): 1425-1433.

[6] Liu Fang, Zhang Xiaoli, Chen Xuelei, et al. "A method for identifying open-source components of package-free management files based on static analysis." Computer Engineering and Application, 2016,52 (11): 57-62.

[7] Huang Liwen, Jiang Xiaoming, Yao Qinghua, et al. "Study on open source components for package management files." Computer Science, 2015,42 (9): 246-249.

[8] Peng Zhang, and Haiyan Liu. "Summary of open source components for package-free management files." Computer Engineering, 2014,40 (8): 136-140.

[9] Yang Hongwei, Zhang Lin, Ma Junfeng. "Package-free management file open source component version number determination method and its implementation." Computer Application and software, 2013,30 (8): 249-252.

[10] Wang Lihong, Li Hao, Guo Haiyan, et al. "A method for identifying open-source components of package-free management files based on static analysis." Computer Engineering and Design, 2002,23 (3): 134-137.