BINBOM
UserGuide

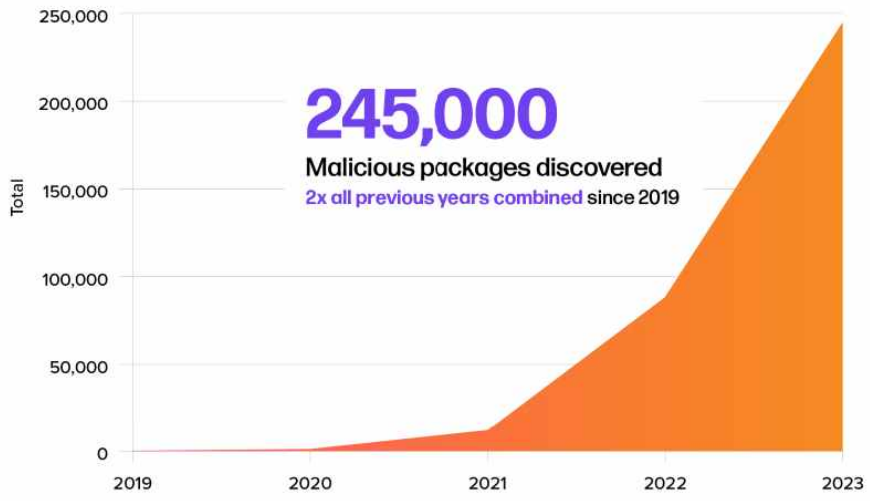
2023. 12. 14.
CSSA

Contents

1. Introduction and Overview of Binary SBOM

A. Introduction to SBOM

With software supply chains becoming increasingly complex and the reuse of open-source software (OSS) growing, attacks targeting the supply chain are also increasing. In response, many countries are actively discussing a range of countermeasures-including attack detection, proactive response, identification of affected components, confirmation of impact scope, and follow-up actions—to build safer supply-chain environments. One foundational approach to addressing these issues is recognizing the importance of the Software Bill of Materials (SBOM), which helps identify the components that make up software.



(Figure 1 in the original document: “Increase in supply chain attacks” (Sonatype, 2023)

B. Introduction to Binary SBOM

Conventional SBOM generation typically requires access to source code. However, in real industrial environments, organizations often use firmware or libraries they did not develop themselves and receive only in binary form (e.g., pre-built firmware images or compiled binary libraries). In such cases, generating an SBOM is constrained by the lack of source code. Therefore, there is demand for generating SBOMs directly from binary artifacts. This technical document focuses on methods for generating an SBOM for binaries.

C. Document Version History

Date Written	Version	Reason	Author
2023-12-14	V1.0	First edition	Kyeongseok Yang

2. Background

A. Limitations of Existing Binary Similarity Analysis Techniques Problems with binary code-clone (similarity) techniques

studied to date include:

1. Using approaches that reduce accuracy: relying on static features makes them vulnerable to variations caused by the operating system, architecture, compiler type and version, and compilation options.
2. Using slow approaches: dynamic-feature-based comparison can handle such variations, but overall speed becomes slow.

This document presents a similarity-analysis approach that addresses these issues. By applying the improved similarity analysis, embedded components can be detected quickly and accurately when generating a Binary SBOM.

B. STRANDS

A strand is a set of instructions required to compute a single variable. Because a strand represents an independent data flow, it can be used as a medium for extracting data-flow information within a basic block. This enables selective extraction of core data flows within a program. Compared to the basic-block-graph representations commonly used by similarity-analysis techniques, strands operate at a finer granularity and can partly mitigate problems arising from differences in compilation environments. In addition, strands can be extracted statically, enabling fast extraction. Accordingly, this document performs binary-code similarity detection at the strand level to better handle various changes, including compilation-environment differences.

3. Binary Similarity Analysis Technology

This section explains the design and implementation approach of the binary similarity analysis technology.

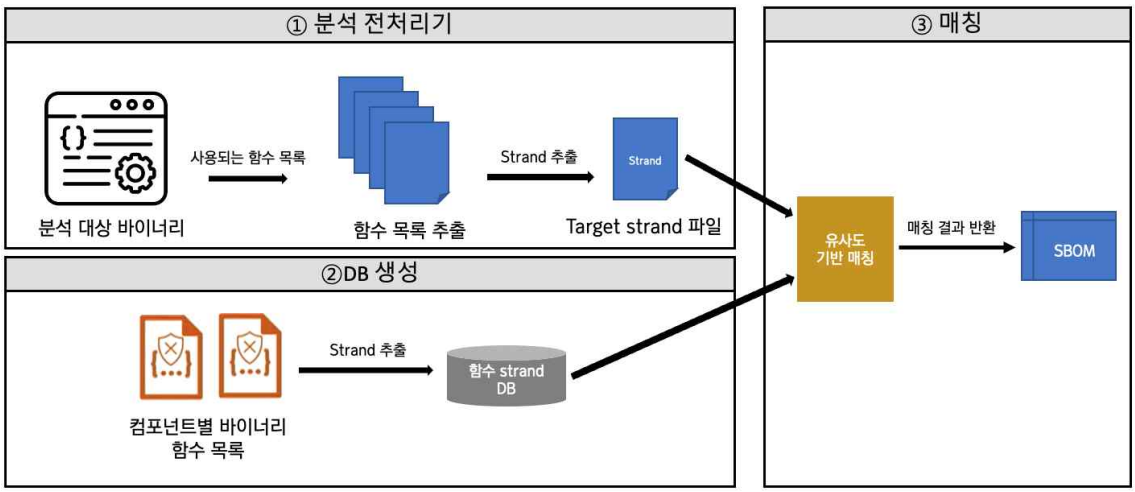


Figure3. in the original document shows the end-to-end overview

① Analysis Pre-processing

Analysis Pre-processor In the overall system, the analysis pre-processing stage proceeds as follows:

1) The program receives the target binary to analyze as input.
2) It extracts all functions that exist in the binary.
3) For each function, it converts to an intermediate representation (IR) and extracts the set of strands.

Finally, the extracted strands are converted into JSON and stored as a file-based database. As shown in the example in the original document (Figure 6), the function name, offset within the binary, size, and strand information are stored.

Figure 4 Example Output of DB

② DB Generation

DB Pre-processor (DB Generation) This describes the process for building a strand database used for mapping:

1) The program receives, as input, a set of binaries from which to build the DB.

2) It extracts all functions in the binary set.
3) For each function, it converts to IR and extracts the set of strands.

The strand database is also stored as JSON files. To remove duplication across binaries in the DB, overlapping functions across multiple binaries are saved into a single file. Multiple binaries are described together in the JSON file, reducing space complexity.

③ Matching

The strand set of the input binary is compared against the vulnerability (or reference) database constructed as above to detect code clones. The process is as follows:

1) Receive the file path of the input binary's strand DB as input.
2) Apply a function-length filter to improve performance.
3) Perform binary similarity checks via strand-level similarity analysis.

D. Binary SBOM Generator Screen

The SBOM generator is integrated into the IoTcube tool. As shown in the original document (Figure 5), the user first selects the Binbom build tool under the Testing menu.

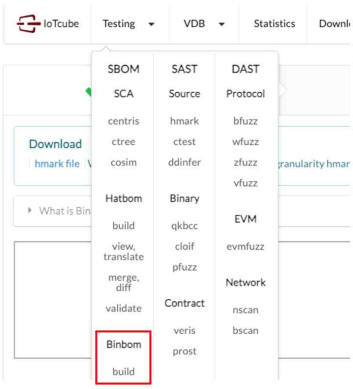


Figure 5 Tool Screen

E. How to Use the Binary SBOM (binbom)

The user inputs the binary extracted in strand form via drag-and-drop on the screen.

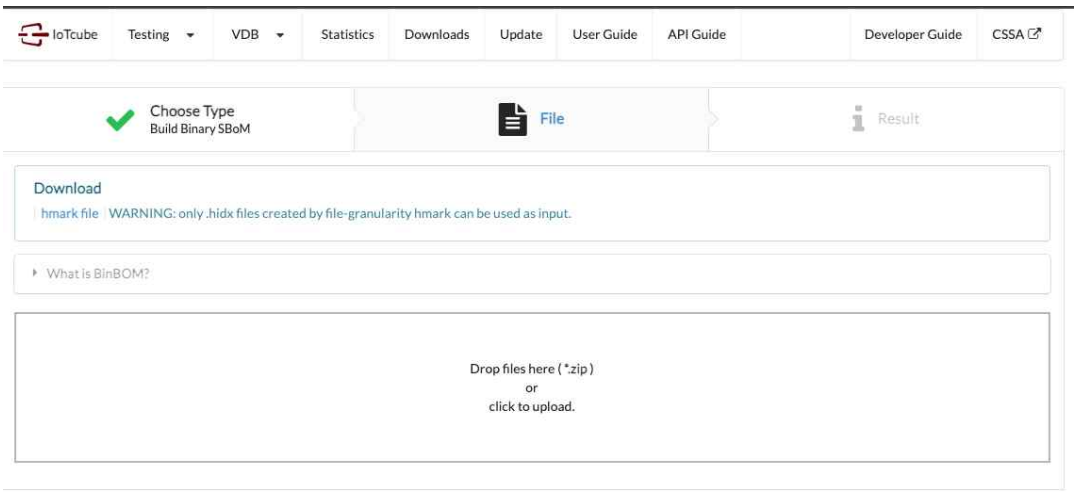


Figure 6 Input Binary Entry Screen

As a result, the SBOM output for the input binary is generated, as shown in the original document (Figures 6 and 7).

Figure 7 SBOM Generation Screen

4. Future Improvements

At present, the tool is a prototype for Binary SBOM technology. By adding more data to the DB and taking additional measures regarding how inputs are generated, it should be possible to build a Binary SBOM tool with improved accuracy.

5. Support

A. Inquiries

If you have questions about this technical document, please contact the following information.

B. Contact Information

Phone: 010-5913-8015
E-mail: ks8171235@korea.ac.kr