Poster: Towards SBOM-based Access Control for Transparent and Explicit Program Execution

Yuta Shimamoto*, Hiroyuki Uekawa†, Mitsuaki Akiyama† and Toshihiro Yamauchi*
* Okayama University, Okayama, Japan
Email: shimamoto@s.okayama-u.ac.jp yamauchi@okayama-u.ac.jp
†NTT Social Informatics Laboratories, Tokyo, Japan
Email: h.uekawa@ntt.com akiyama@ieee.org

***Abstract*—Although a Software Bill of Materials (SBOM) plays a key role in software transparency, inconsistencies in SBOM descriptions can undermine its value. To address this, we propose a novel approach to program access control, SBOM-AC, which leverages Mandatory Access Control (MAC) systems to ensure transparent and explicit program execution. In this study, we identify the challenges associated with implementing this approach and present preliminary investigation results to address these challenges.**

I. Introduction

A Software Bill of Materials (SBOM) is an inventory for explicitly enumerating and managing the various software components included in a system, making it easier to identify and address security and license issues. Its importance has been highlighted in the U.S. Executive Order [1] and the EU Cyber Resilience Act [2], drawing increasing attention.

However, since SBOMs are generated either manually or using tools, the content of the SBOM may contradict the actual system behavior. Below, we outline situations where contradictions may arise and the reasons behind them:

Unintentional contradiction: (1) Human errors and (2) Inaccuracy or incompleteness of SBOM-generation tools [3]

Intentional contradiction: To conceal malicious programs (1) Deleting specific SBOM entries manually and (2) Manipulating tool outputs [4].

Based on these observations, situations where an SBOM has inaccuracy and/or incompleteness can easily arise. As a result, software transparency may deteriorate, significantly reducing the effectiveness of software management practices based on SBOM, including security measures such as vulnerability mitigation.

To address the contradiction issue, it is essential to ensure consistency between the system's behavior and the contents of the SBOM. To achieve consistency and transparency, we propose SBOM-based Access Control (SBOM-AC), an approach that leverages Mandatory Access Control (MAC) mechanisms to execute only the programs explicitly enumerated in the SBOM selectively.

In this study, we clarify the challenges in implementing our approach and present preliminary investigation results to address these challenges.

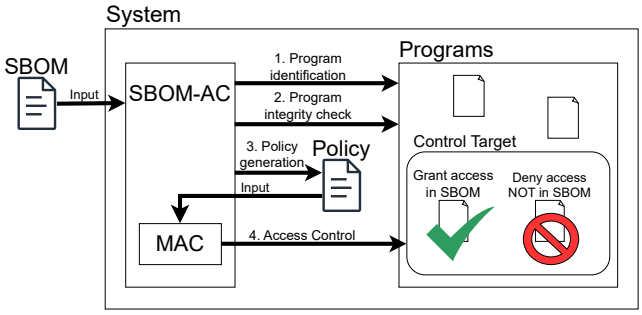


Fig. 1. Overview of our proposed approach

II. Methodology

SBOM-AC links the descriptions in the SBOM to the programs within the system and denies access that conflicts with the SBOM. As shown in Figure 1, SBOM-AC consists of the following four functions: Program identification, Program integrity check, Policy generation, and Access control.

Program identification refers to associating the programs listed in the SBOM with their corresponding executable files in the file system by referencing information such as file paths.

Program integrity check verifies the consistency between the existing programs and the SBOM (e.g., comparing hash values of programs), allowing only programs without inconsistencies to be executed.

Policy generation creates policies that deny/grant execution of any programs under the control target (specific directories where executables are located).

Access control enforces access control by loading the generated policies into the MAC system.

III. Challenges and Preliminary Study

A. Can existing tools generate SBOMs for SBOM-AC?

SBOM-AC has two requirements for SBOM: (1) it must cover a runtime environment or system image (e.g., a file system) rather than just code and repositories, and (2) it must include information (e.g., hash values and file paths) to identify distinct programs on a target system uniquely.

Tools automatically generating SBOMs have been developed to reduce human error and associated costs. However,

the scope of analysis and the output information varies significantly across tools. To investigate the extent to which off-the-shelf tools can be utilized to generate SBOMs meeting SBOM-AC's requirements, we conducted a literature review.

We analyzed the following tools mentioned in at least five of 35 recent studies: Syft, Trivy, Cdxgen, OSS Review Toolkit (ORT), Microsoft SBOM Tool, and Tern.

Scope for SBOM: These tools support analysis of runtime environments or system images (as envisioned by SBOM-AC), including file systems, VM images, Docker containers, Open Container Initiative (OCI), Singularity Image Format (SIF), and Kubernetes. All tools, except for ORT, support at least one of these runtime environments. Therefore, using SBOMs generated by these tools in SBOM-AC is feasible. While the range of supported forms for analysis is currently limited and varies across tools, future developments are expected to broaden the range of supported forms.

Necessary information: Among the tools, only Syft generates SBOMs containing the necessary information for program identification and integrity check (i.e., specific file paths and hash values). We hope that more tools will incorporate this type of information into their SBOM outputs.

B. Can existing MAC systems control programs based on SBOM-generated policies?

SBOM-AC is designed to determine programs permitted for execution based on an SBOM and generate access control policies accordingly. Our aim in this study is to utilize MAC systems to enforce control over program execution. Specifically, a program is granted execution only if it is correctly listed in the SBOM; otherwise, its execution is denied. Additionally, it is desirable for the scope of programs controlled on the system (referred to as the "control target") to be flexibly configurable, depending on the type of system. This flexibility addresses two concerns: (1) The potential overhead of controlling every program on the system and (2) Debates over which programs should be included in the SBOM—for instance, how to handle programs that are standard components of Linux distributions (i.e., OS-bundled programs).

We investigated whether off-the-shelf MAC systems can define policies to control both the execution of programs (allow/deny) and the scope of controlled targets. To evaluate this, we tested AppArmor, a widely used implementation of MAC systems.

Using a simple file system and SBOM, we created the policy illustrated in Figure 2. In this setup, the `/usr` directory on the system was designated as the control target, and `s1-prog1` and `s2-prog3` were permitted for execution. The policy ensures that programs listed in the SBOM are allowed to execute, while those not listed are denied.

Key rules implemented in the AppArmor policy:

1) Any processes are monitored. All operations are denied by default.
2) All operations except execution are allowed.
3) Execution of programs listed in the SBOM is allowed.

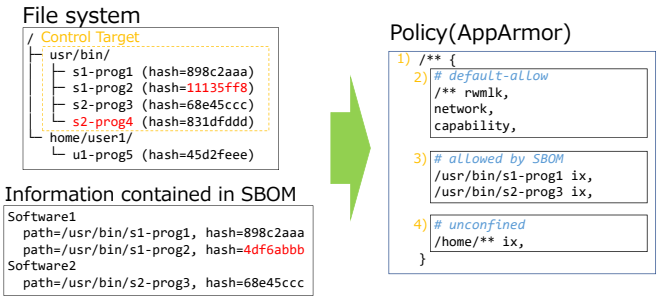


Fig. 2. Example of SBOM-generated policy for AppArmor

4) Programs outside the defined control target scope are not denied execution.

Rules 1, 2, and 4 collectively define the scope of the control target. By default, access within the control target is denied unless explicitly allowed. Rule 3 ensures that only the programs listed in the SBOM are granted execution.

These rules ensure that only the desired programs listed in the SBOM are executable while maintaining flexibility in defining the control target scope. As a result, AppArmor is shown to support the creation of policies that meet the basic requirements of SBOM-AC.

IV. Coclusion and Future Work

We proposed SBOM-AC, an approach to detect and control inconsistencies between an SBOM and a system, and presented the challenges and preliminary investigation results. In future work, we plan to measure the overhead of SBOM-AC and explore the optimal scope of control targets based on system types. Additionally, we aim to develop flexible policy generation and control mechanisms to accommodate changes in programs (e.g., rewrites) during system operation.

References

[1] White House, "Improving the Nation's Cybersecurity." [Online]. Available: https://www.federalregister.gov/documents/2021/05/17/2021-10460/improving-the-nations-cybersecurity

[2] EU, "The Cyber Resilience Act." [Online]. Available: https://www.cyberresilienceact.eu/

[3] N. Kawaguchi, C. Hart, and H. Uchimura, "Understanding the effectiveness of sbom generation tools for manually installed packages in docker containers," in *Journal of Internet Services and Information Security*, vol. 14, August 2024, pp. 191–212.

[4] S. Yu, W. Song, X. Hu, and H. Yin, "On the correctness of metadata-based sbom generation: A differential analysis approach," in *2024 54th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2024, pp. 29–36.

Towards SBOM-based Access Control for Transparent and Explicit Program Execution

Yuta Shimamoto † Hiroyuki Uekawa ‡ Mitsuaki Akiyama ‡ Toshihiro Yamauchi †
†Okayama University ‡NTT Social Informatics Laboratories

1. Introduction

What is SBOM
- A Software Bill of Materials (SBOM) is an inventory for explicitly enumerating and managing the various software components included in a system.
- SBOM makes it easier to identify and address security and license issues.
- SBOM importance has been highlighted in the U.S. Executive Order and the EU Cyber Resilience Act.

Inaccuracy and incompleteness of SBOM
- The content of the SBOM may contradict the actual system behavior.
- Software transparency may deteriorate, significantly reducing the effectiveness of software management practices based on SBOM.

Contradictions

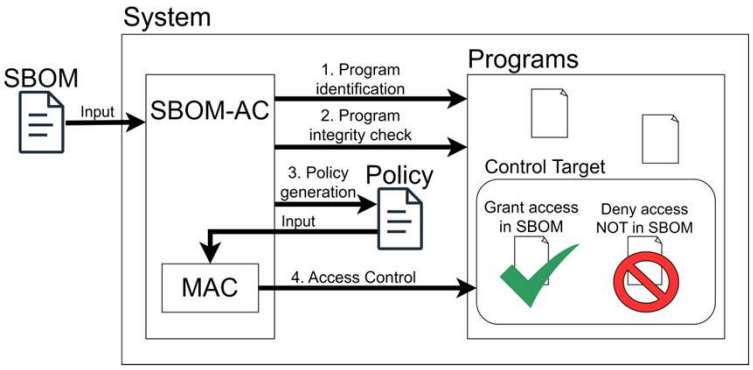
Intentional
- Concealing malicious programs by
  - ✓ Deleting specific SBOM entries manually
  - ✓ Manipulating tool outputs

Unintentional
- Human errors
- Inaccuracy or incompleteness of SBOM generation tools

SBOM-based Access Control (SBOM-AC)
- Executes only the programs explicitly enumerated in the SBOM
- Leverages Mandatory Access Control (MAC)

2. Methodology

SBOM-AC links the descriptions in the SBOM to the programs and denies access that conflicts with the SBOM



1. Program identification
- Associates programs listed in the SBOM with executable files in the file system (e.g., file paths)

2. Program integrity check
- Verifies the consistency between the existing programs and the SBOM (e.g., comparing hash values of programs)
- Allows only programs without inconsistencies to be executed

3. Policy generation
- Creates policies to deny/grant execution of any programs under the control target(Specific directories where executables are located)

4. Access Control
- Enforces access control by loading the generated policies into the MAC system

3-A. Can existing tools generate SBOMs for SBOM-AC?

Requirements for SBOM
- Cover a runtime environment or system image (e.g., a file system) rather than just code and repositories
- Include information (e.g., hash values and file paths) to identify distinct programs on a target system

Runtime environment
- VM images
- Docker containers
- Open Container Initiative
- Singularity Image Format
- Kubernetes

Popular SBOM generation tool
- Syft
- Trivy
- Cdxgen
- Tern
- OSS Review Toolkit (ORT)
- Microsoft SBOM Tool

Feasibility of using popular tools SBOMs

Scope for SBOM
- All tools except for ORT, support at least one of the runtime environments.
- Range of supported forms for analysis is currently limited and varies across tools.

Necessary information
- Only Syft's SBOMs contain the necessary information for program identification and integrity check.

3-B. Can existing MAC systems control programs based on SBOM-generated policies?

Required features for SBOM-AC
- Program must be granted execution only if it is correctly listed in the SBOM
- Desirable to be flexibly configurable the scope of programs controlled on the system (referred to as the "control target") depending on the type of system

Example of SBOM-generated policy for AppArmor

1) Any processes are monitored. All operations are denied by default.
2) All operations except execution are allowed.
3) Execution of programs listed in the SBOM is allowed.
4) Programs outside the defined control target scope are not denied execution.

File system

```
/ Control Target
├─ usr/bin/
│  ├─ s1-prog1 (hash=898c2aaa)
│  ├─ s1-prog2 (hash=11135ff8)
│  ├─ s2-prog3 (hash=68e45ccc)
│  └─ s2-prog4 (hash=831dfddd)
└─ home/user1/
   └─ u1-prog5 (hash=45d2feee)
```

Information contained in SBOM

```
Software1
  path=/usr/bin/s1-prog1, hash=898c2aaa
  path=/usr/bin/s1-prog2, hash=4df6abbb
Software2
  path=/usr/bin/s2-prog3, hash=68e45ccc
```

Policy(AppArmor)

```
1) /** {
2)   # default-allow
     /** rwmlk,
     network,
     capability,
3)   # allowed by SBOM
     /usr/bin/s1-prog1 ix,
     /usr/bin/s2-prog3 ix,
4)   # unconfined
     /home/** ix,
   }
```

4. Future Work

- Measure the overhead of SBOM-AC and explore the optimal scope of control targets based on system types.
- Develop flexible policy generation and control mechanisms to accommodate changes in programs (e.g., rewrites) during system operation