GrizzlyBay: Retroactive SBOM with Automated Identification for Legacy Binaries

Yung Ryn Choe

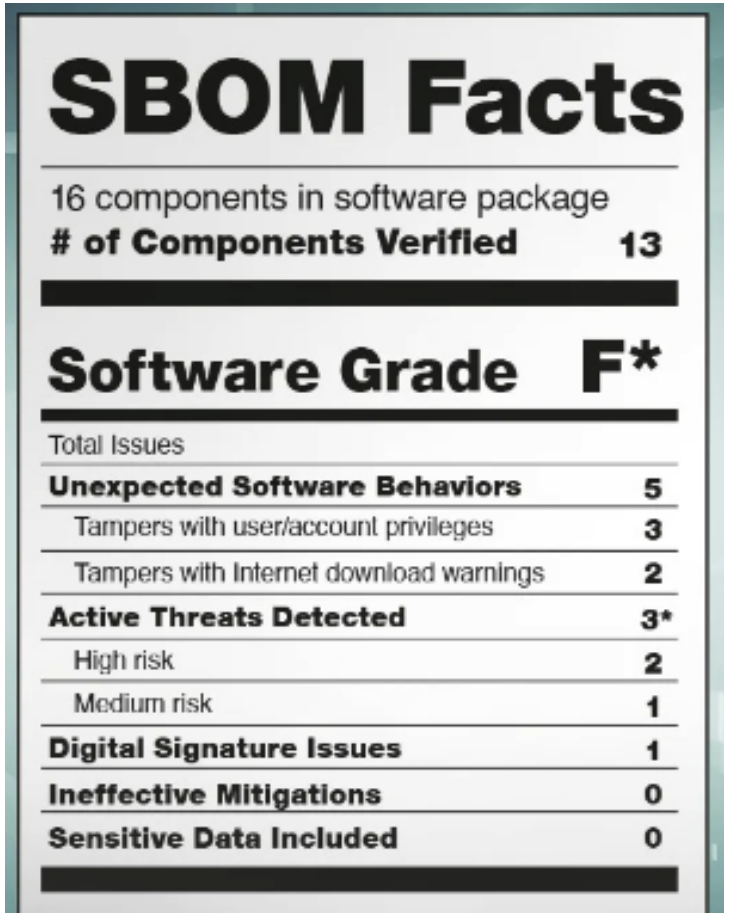
Junghwan Rhee, Fei Zuo,
Cody Tompkins, Jeehyun Oh

IEEE Military Communications Conference (MILCOM 2024)

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Motivation

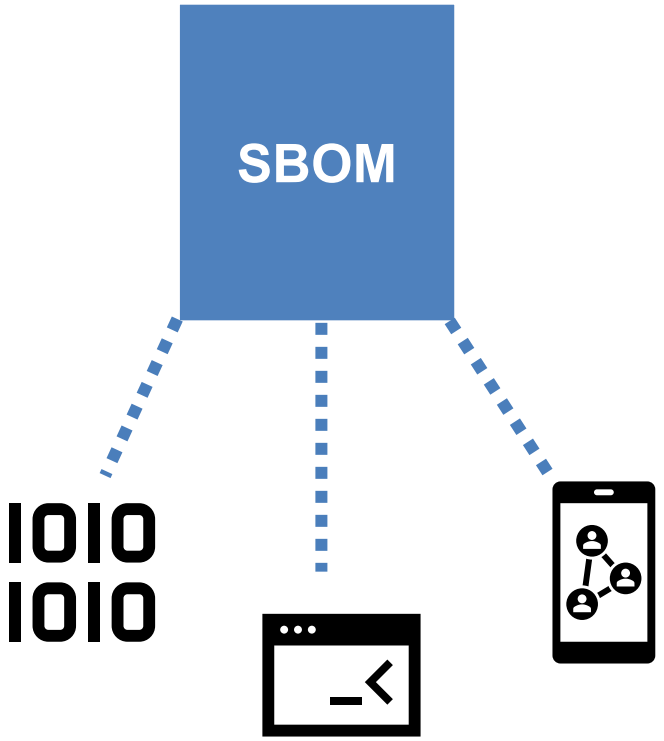
- Software Bill of Materials (SBOM) is a common practice to keep track of software information and make software supply chain transparent.
- SBOMs include multiple useful information such as developers, versions, and dependent software.
- SBOMs have been used as specifications as main goals.



* Image: Reversing labs

Motivation

- We would like to provide a new function of a **linkage** from specification to software binary and its variants.
- Binary-only software without source code makes it harder to understand its ingredients.
- There are so many software distributions across different operating systems and platforms that make their binary content varied. We would like to have a **similarity measurement** as a solution.



Contribution

- **Binary-only SBOM:** Our approach generates SBOM that can identify software and function names and versions **retroactively from native binary software** even when no information is available other than a binary blob.
- **Binary-similarity with platform-independent LSB:** We propose a novel **locality-sensitive binary signature** that can offer swift similarity comparison by combining platform-independent binary representations and locality-sensitive-hashing.
  - It enables the identification of program and function names and versions with similar signature matching and a ranking technique.
  - We evaluated our tool by identifying binary names, function names, and their accurate versions.

Design Goals and Our Solutions

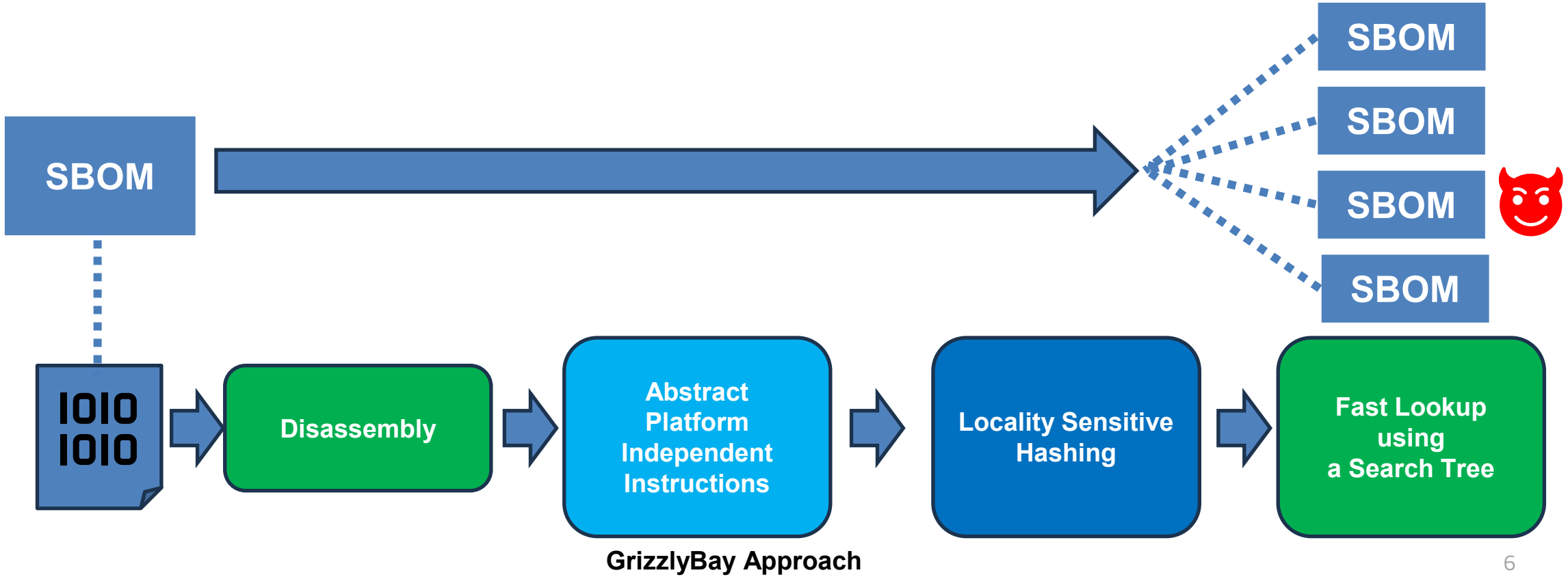
- **Design Goal 1**: Robust capability to handle CPU instructions in an abstract way so that it can deal with noises introduced by the software compilation process that may choose different instructions.
- Using **platform-independent PCode language** similar to RTL (register transfer language)
- Using a graph of basic blocks and P-code counts.

Design Goals and Our Solutions

- **Design Goal 2:** Concise representation of binary code with a fixed length to be part of an SBOM
  - Locality-sensitive hash (LSH) has a **concise fixed-length representation of a binary for efficient comparison and identification**.
  - Our design is agnostic to a choice of LSH. We use tlsh from TrendMicro for implementation of our prototype.
- **Design Goal 3**: Support for scalable and efficient comparisons for a large collection of binary information
  - **The locality-sensitive hash tree** provides scalable and efficient look-up.
  - TLSH tree provides O(logN) look-up time for a match and neighbors.

GrizzlyBay SBOM

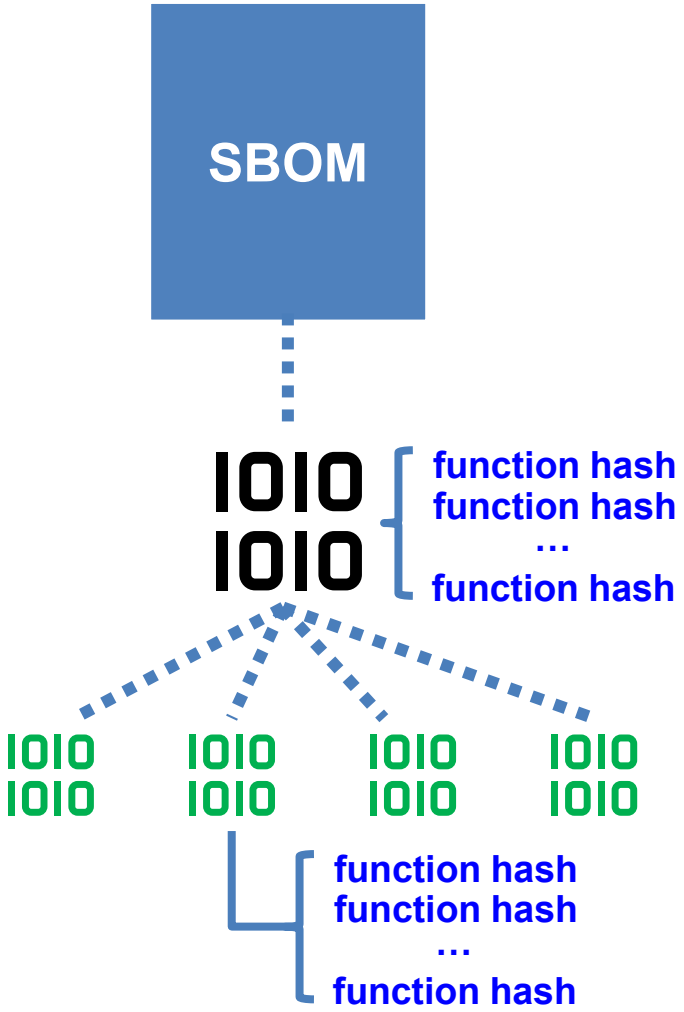
- **Binary-only analysis for SBOM** and vulnerability identification
- The key is to enable **scalable similarity match of binary code**.



GrizzlyBay Approach

GrizzlyBay SBOM

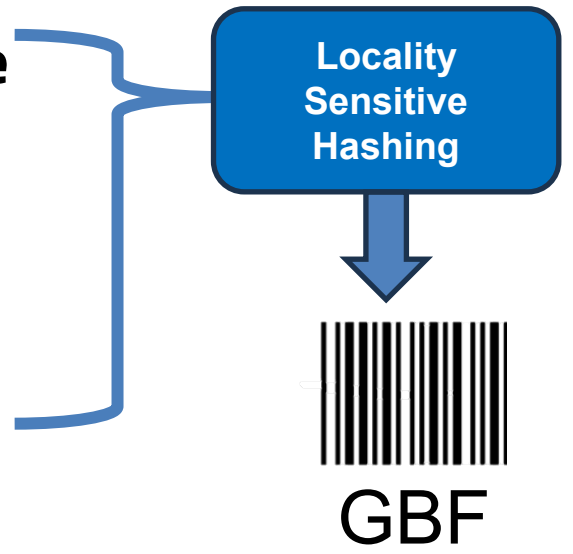
- New representation of **the main binary**:
  - GrizzlyBay supports a new hash with a similarity matching function for the main software binary (GB Binary Level Hash).
- New representations for **dependent components**:
  - For all dependent libraries, new hashes with a similarity matching function are generated (GB Binary Level Hash).
- New representations for **functions**:
  - This is a **finer-grained representation of functions** that our SBOM uniquely offers. This is **a desired function to identify vulnerabilities** because most vulnerabilities and patches are associated with a particular function.
  - **For each function inside the binary, a new hash with a similarity-matching function is provided** (GB Function Level Hash).



GrizzlyBay Locality Sensitive Hashes

- **GrizzlyBay Locality Sensitive Function Level Hash (GBF)** is made by applying a locality sensitive hash function on the list of following information.

  - **The opcode list of PCode and the count of each opcode**
  - **The count of basic blocks**
  - **The number of basic block outgoing edges**
  - **A list of named callee functions**
  - **The count of callee functions**



GrizzlyBay Locality Sensitive Hashes

- **GrizzlyBay Locality Sensitive Program Level Hash (GBP)** is created using a sorted list of function-level hashes.
- Specifically, a locality-sensitive hash for a file is produced by applying the locality-sensitive hashing algorithm on the concatenated sorted function hashes.
- The function level hashes with similar content will be in a similar location inside the whole binary hash.

Implementation

- We implemented a prototype system, GrizzlyBay, by using multiple open-source components.
- Ghidra version11 is utilized as a disassembly engine to generate control-flow-graphs, call graphs, basic blocks, pcode for each instruction, imported symbols, and dependent libraries.
- The disassembly process, a function-level hash query, a file-level hash query, and a CVE look-up functions are implemented as micro-services which are integrated into one system.
- We wrote around 3,500 lines of Python code for the entire implementation.

Evaluation

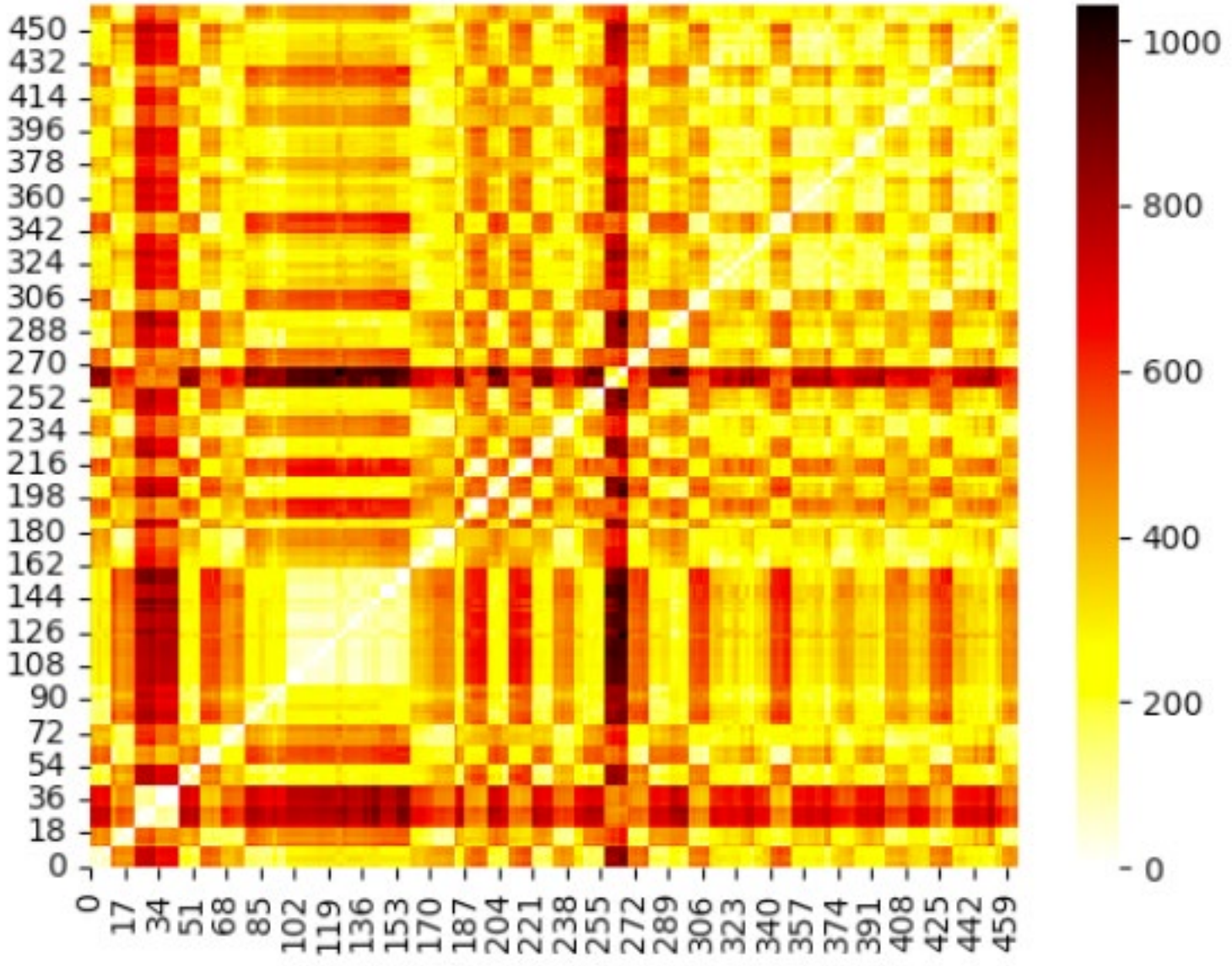
- 5G core network software, Open5GS2, over 11 different versions:
  - 2.1.3, 2.1.4, 2.2.1, 2.3.2, 2.3.6, 2.4.0, 2.4.9, 2.6.1, 2.6.2, 2.6.3, and 2.6.4.
- Each version has
  - 13~16 main executables
  - 23~30 libraries.
- In this benchmark set, there are a total of 465 files and 131,696 functions included.

Matching Effectiveness – Program Hash

- GBP Performance Summary:
- We define the number of program binaries matched at the top X rank as N_X and the total number of program binaries as T.
- The Top X hash matching rate formula R_X is as $R_X = N_X / T$
- All 465 binary files found themselves as the rank 1 using the GBP. 100% of the top 1 hash matching rate (R_1=1) is achieved.

Matching Effectiveness – Binary Similarity using GrizzlyBay SBOM Program Hash (GBP)

- We made the combinations of all binaries and calculated their distance score.
- A light color shows high similarity with a low distance and a dark color indicates low similarity (high distance).
- Each binary matched itself with the zero distance. This matching is shown as a white diagonal line across the picture.



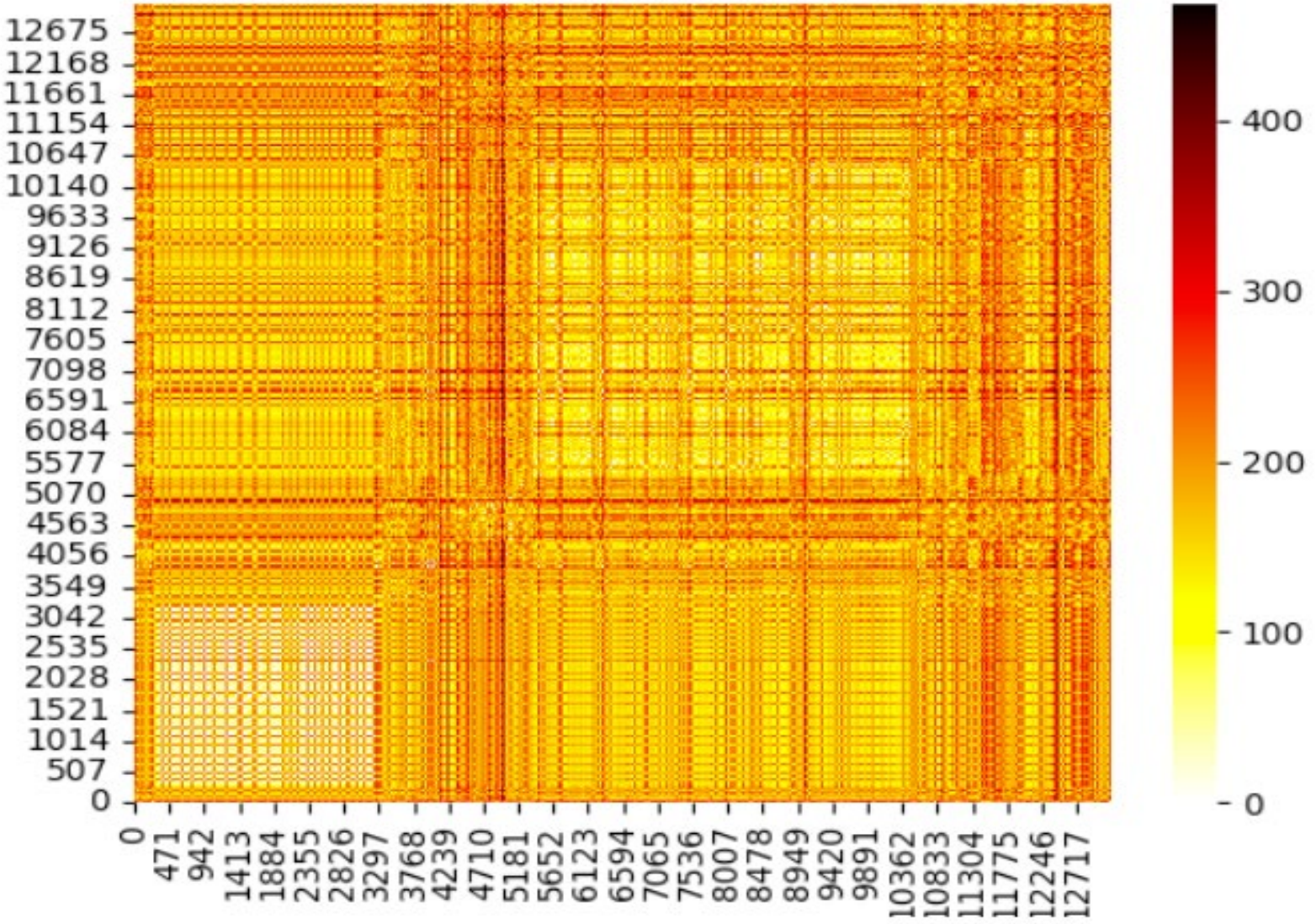
Matching Effectiveness – Function Hash

- A function has significantly less information than a program binary. Therefore, it poses a greater challenge in its evaluation.
- We compared each function from each program binary with every other function in all program binaries.
- GBF Performance Summary: All 131,696 functions found themselves as rank 1 using the GBF.
- In other words, 100% of the top 1 hash matching rate (R_1=1) is achieved. This result shows our technique's high accuracy with a fine-grained resolution so that it can match the identical function with no confusion (100% accuracy).

Matching Effectiveness – Function Hash

- We performed the entire computation, but it couldn't be plotted due to its size of 17 billion comparison caused 262 GB of the data in the text format. Instead, Figure 2 illustrates our 1/10 sampled comparison result.
- A lighter color shows a high similarity.
- We found the LSH difference score is generally lower (more similar) than the score difference from the binary-level signature comparisons.
- This is expected because a binary-level signature is composed of the aggregation of all function-level data in the binary.

Binary Similarity using GrizzlyBay SBOM Function Hash (GBF).



Reliability Comparison with Ghidra's BSIM

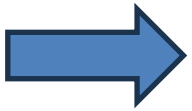
- Ghidra could not generate BSim signatures for multiple software binaries due to errors raised in the decompilation process (converting instructions into source code).
- 8% of total functions (37% of files) in the benchmark set caused BSim to be unable to handle function signatures.
- Our approach is not affected because it does not require the decompilation function (disassembly only) having higher reliability.

Hash Look-up Performance

- Function-level signature matching provides a powerful capability to identify a function. But a matching system will need to take care of **a huge number of signature matchings**.
- This is a concern in a traditional relational database because it performs exact matchings with each individual signature in the database.
- However, our approach is based on the **hash search function works like a binary tree**. It can find a correct match with **a significantly smaller number of comparisons**.
- We measured a signature lookup performance. Our signature database has 465 file signatures and 131,696 function signatures.
- On average out of hundreds of random lookups, **a file signature lookup took 4 ms and a function information lookup with tracking close signatures took 10 ms**. This experiment was conducted in a KVM with a 2394.374 MHz CPU.

Case Study of Vulnerability Identification

- **CVE-2021-41794:**
  - **ogs_fqdn_parse** function in the binary libogscore.so.
  - Signature: the version 2.1.4.
  - Matched versions: 2.1.3, 2.1.4, 2.2.1, 2.3.2 were found as the identical matches.
  - Those functions are indeed the same code sharing the vulnerability.
- **CVE-2021-44081:**
  - **ogs_nas_5gs_suci_from_mobile_identity** function of the **libogsnas-5gs.so** binary.
  - Signature: the version 2.1.4
  - Matched versions: 2.1.3, 2.1.4, 2.2.1 were detected.
  - They are the same functions with the common vulnerability.
- **CVE-2021-44108:**
  - **amf_namf_comm_handle_n1_n2_message_transfer** function of the **open5gs-amfd**.
  - Signature: the version 2.1.4
  - Matched versions: 2.1.4, 2.2.1 are matched as the exact function.
  - Both versions are confirmed to be vulnerable.



Accurate vulnerability identification for real cases

Conclusion

- We propose a new **Software Bill of Materials (SBOM) for legacy binary-only software.**
- It enables **similarity search, identification of binary programs and individual functions, and corresponding vulnerability search** for transparent software supply chains and usages.
- As the key technique, we introduced a novel binary hash which is defined by combining **abstract machine instruction statistics, basic block, and control block properties with locality-sensitive hashing**.
- GrizzlyBay accurately identify correct software binaries and functions with their versions with the top 1 rank enabling CVE vulnerability searches with high accuracy.
- Our search database based on locality sensitive hashing performs **prompt queries over several hundred thousand signatures under 10 ms** with our unoptimized Python-based implementation.

Thank you!

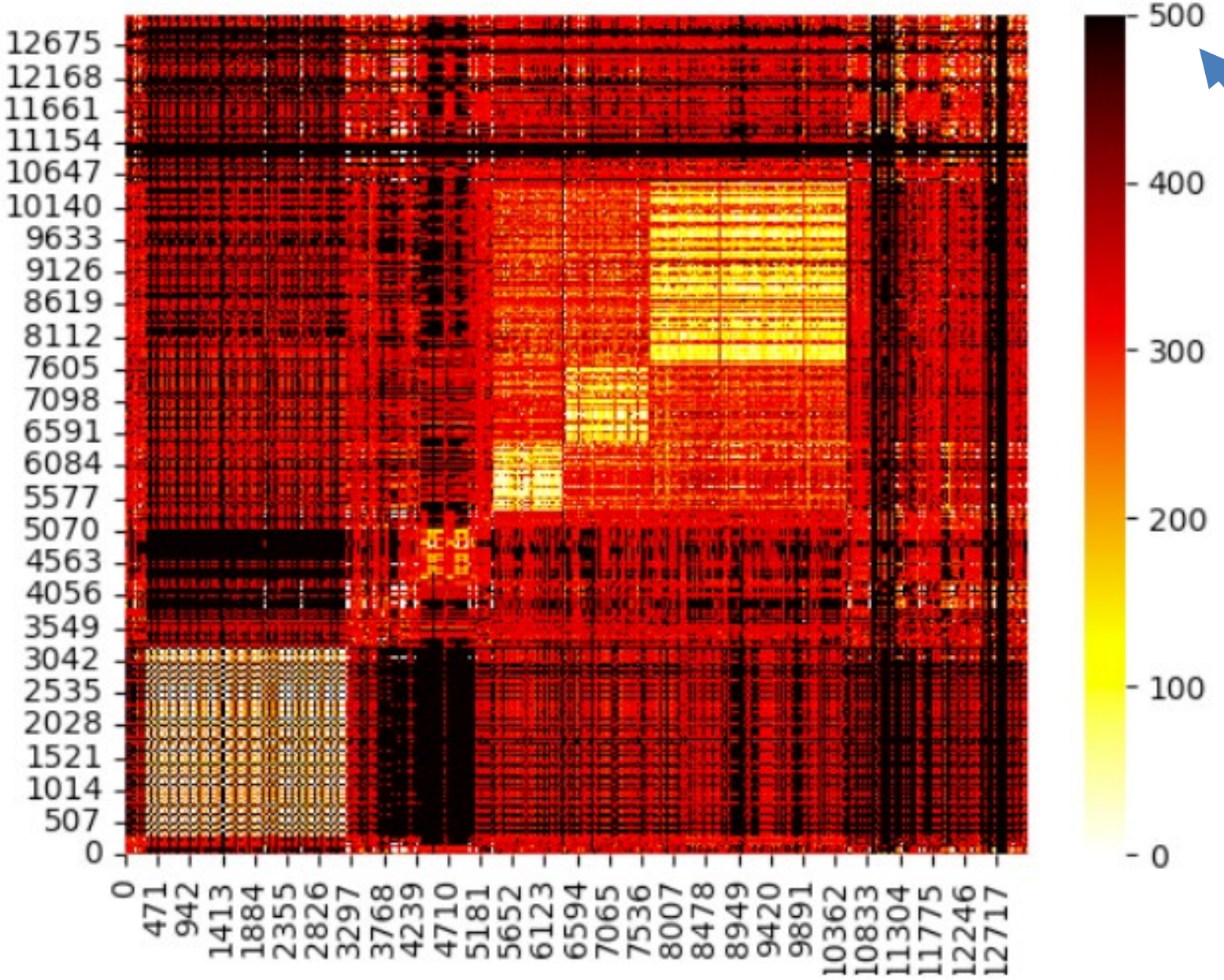
Q&A

Related Work

- SBOM
  - Currently, SPDX [5] and CycloneDX [4] are two representative SBOM standards proposed by The Linux Foundation and OWASP, respectively.
  - They also developed corresponding SBOM generation tools. Moreover, prior work [6], [7], [19] empirically investigated the current state of SBOM practice and discussed existing challenges.

Related Work

- Machine learning approaches
  - Applying the exhilarating advances in deep learning for automated vulnerability detection has attracted widespread in- terest, for instance, VulDeePecker [20] and SySeVR [21]. Nonetheless, recent study found that the performance of these learning-based methods drops by more than 50% in a real- world scenario [22].
- Strict binary check
  - In addition, OWASP Dependency-Check [23] and Intel's CVE-bin-tool [24] are two commonly employed static analysis tools that provide an objective evaluation of the vulnerabilities. The former is a software composition analysis tool that can scan applications and their dependent libraries to discover publicly disclosed vulnerabilities. The CVE-bin- tool is used for scanning known components in various formats, including several SBOM formats. However, both tools are not able to find any result from the binaries investigated in this work.



500: Missing comparison due to errors