Datasheet: Assemblage Total Binaries (Snapshot March 2024)

ASSEMBLAGE TEAM

1 Introduction

To address the lack of available benign binaries, along with ground truth compilation and configuration data, we present Assemblage. Assemblage is both a richly-diverse corpus of Windows PE and Linux ELF binaries, and also the distributed system that generates the corpus. Assemblage continuously crawls GitHub, diversifies repositories, and builds as many binary artifacts as it can. The design of Assemblage enables adding new workers, builders, and post-processors to add new toolchains, analyses, and diversification mechanisms. Here we describe the largest to-date dataset we have built using Assemblage, which includes publicly-available source repositories. We distribute a subset of the binaries which correspond to permissively-licensed repositories (as to not distribute unlicensed code); however, Assemblage does enable crawling and building unlicensed code, and is designed to enable distributing "recipes," which can reproduce a binary corpus with high fidelity.

2 Dataset Generation

The main source for the dataset lies on GitHub and package managers. The binaries are built from 4 million C++ repositories queried from Jan 01 2010 to Nov 28 2023, while the Windows worker will try build the repositories with Solution files, and the Linux worker will build the repositories with Makefile.

To diversify the binaries from same copy of source code, parser for configuration files of Visual Studio and Makefile are implemented to modify the compiler flags during compilation and building. The actual compiling and building are implemented by calling MSBuild on Windows and make on Linux.

During the compilation and building, Make and MSBuild are called on either makefile or solution file, each triggered its own compiler tool chain. By inspecting the binary generated and the compiler tool chain's process exit code, this copy of binary is decided whether successfully built, if so the parser for pdb files utilizing DIA2dump [2] would read from pdb files and retrive the mapping from source code to the address and bytes within binaries.

3 SQLite Details

A comprehensive capture of binary and function information stored during building and retrieved from pdb files is provided as a SQLite database, where each tables stores specific part of the information and can be queried to generate

Author's address: Assemblage Team.

a subset of features for faster access. The overview of the schema is illustrated in Figure 1, and each table's details is listed as below,

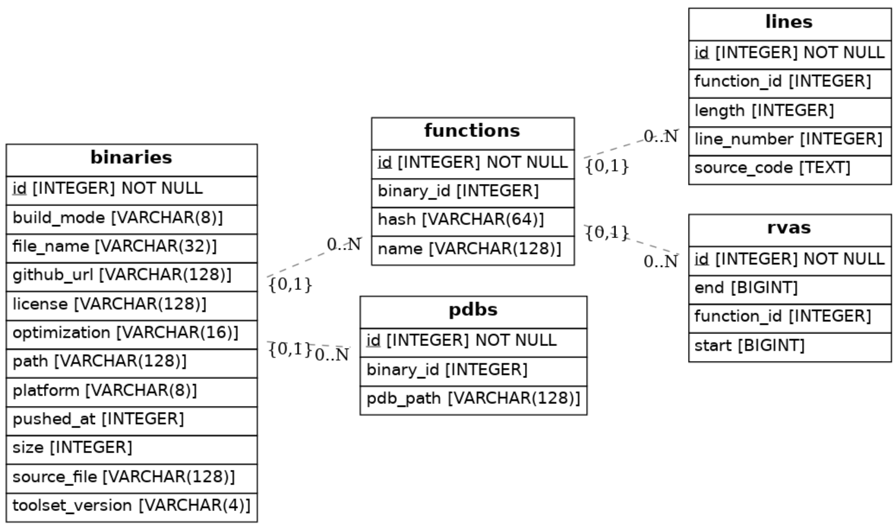

Fig. 1. Schema of SQLite database for dataset

- Binaries table provides the basic binary information, the compiler version and optimization level, the source code URL, and size of each binary
- Functions table provides complete information about each function, such as source code and the hash of its bytes
- RVAs table provides the relative virtual address for each function, which can be queried together with functions in case that one function spans in multiple chunks within the binary.
- Lines table provides the mapping from one line of source code to the RVA address function
- PDBs table indicates the pdb file for each binary

While the SQLite database provides a comprehensive capture of all information about the dataset, it is quite common that only a subset of it is necessary for specific tasks, so shaping data into certain format would increased the data querying speed. Hence, SOME example of querying SQLite database are provided, and with certain script such as sqlite3 and pandas module in Python, these information can be stored into csv for faster access.

4 Dataset Statistics

By the source code hosting platform, license, and build platform, the dataset generated by Assemblage can be divided into several categories, and an overview of the datasets statistics can be found at in Table 1.

Fig. 2. Examples of SQL query

```
-- Count functions of binaries size more than 100KB
SELECT COUNT(*) FROM functions
WHERE binary_id IN (SELECT id FROM binaries WHERE size>100);

-- Select binary information and RVA by function id:
SELECT f.id, f.name, r.start,
b.id, b.toolset_version, b.optimization, b.github_url
FROM functions
WHERE functions.id=some_id
JOIN rvas r ON r.function_id=f.id
JOIN binaries b ON b.id=f.binary_id;

-- Dump all function name, rva address and binary id:
SELECT f.name, f.binary_id, r.start
FROM functions f JOIN rvas r ON f.id==r.function_id;

-- Dump ascending function name and rva starts for binary some_id
SELECT f.name,  r.start
FROM rvas r
JOIN functions f ON r.function_id = f.id
JOIN binaries ON f.binary_id = binaries.id
WHERE binaries.id = some_id
ORDER BY r.start ASC;
```

Table 1. Datasets statistics

Source	Platform	License	Total	Repősotories	Functions	Functions (w/ source code)
GitHub	Windows	Mixed	890k	172k	298M	20M
		Licensed	62k	12k	38M	3M
	Linux	Mixed	428k	48k	316M	N/A
		Licensed	211k	13k	186M	N/A
vcpkg	Windows	Licensed	29k	1k	48M	N/A

In terms of the functions, there exists 160M unique function bytes exists within 298M functions, which can be utilized in various tasks, such as function boundary identification, similarity identification [1, 3, 4]. It is also possible to query each function's address and extract them with PEFile Python module with query shown in Figure 2.

References

[1] Andrea Marcelli, Mariano Graziano, Xabier Ugarte-Pedrero, Yanick Fratantonio, Mohamad Mansouri, and Davide Balzarotti. 2022. How machine learning is solving the binary function similarity problem. In *31st USENIX Security Symposium (USENIX Security 22)*. 2099–2116.

[2] Microsoft. 2022. Microsoft Visual studio Dia2dump Sample. https://learn.microsoft.com/en-us/visualstudio/debugger/debug-interface-access/dia2dump-sample?view=vs-2022

[3] Kexin Pei, Jonas Guan, David Williams King, Junfeng Yang, and Suman Jana. 2021. XDA: Accurate, Robust Disassembly with Transfer Learning. In *Proceedings of the 2021 Network and Distributed System Security Symposium (NDSS)*.

[4] Hao Wang, Wenjie Qu, Gilad Katz, Wenyu Zhu, Zeyu Gao, Han Qiu, Jianwei Zhuge, and Chao Zhang. 2022. jTrans: jump-aware transformer for binary code similarity detection. *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis* (2022). https://api.semanticscholar.org/CorpusID:249062999

A Appendix

Table 2. Configuration distribution of licensed datasets.

OS	Source	Compiler	Opt.	Count
Windows PE	GitHub	MSVC-v140	O1	5286
			O2	12961
			Ox	7395
		MSVC-v141	O1	538
			O2	4017
			Od	4379
			Ox	818
		MSVC-v142	O1	6156
			O2	4260
			Od	4799
		MSVC-v143	O1	5543
			Od	3470
			Ox	3247
	vcpkg	MSVC-v120	O1	952
			O2	945
			Od	956
			Ox	926
		MSVC-v142	O1	2160
			O2	2142
			Od	2187
			Ox	2145
		MSVC-v143	O1	3081
			O2	3078
			Od	3074
			Ox	3083
Linux ELF	GitHub	GCC-11.4.0	Od	25855
			O1	25039
			O3	24081
			Oz	10609
		Clang-14.0.0	Od	28489
			O1	30542
			O2	32809
			O3	34239

Table 3. Configuration distribution on three datasets.

OS	Source	Compiler	Opt.	Count
Windows PE	GitHub	MSVC-v140	O1	60126
			O2	165866
			Ox	100831
		MSVC-v141	O1	6113
			O2	78203
			Od	86841
			Ox	7470
		MSVC-v142	O1	87497
			O2	50055
			Od	72613
		MSVC-v143	O1	80243
			Od	57011
			Ox	37302
	vcpkg	MSVC-v120	O1	952
			O2	945
			Od	956
			Ox	926
		MSVC-v142	O1	2160
			O2	2142
			Od	2187
			Ox	2145
		MSVC-v143	O1	3081
			O2	3078
			Od	3074
			Ox	3083
Linux ELF	GitHub	GCC-11.4.0	Od	50338
			O1	49860
			O3	50995
			Oz	22030
		Clang-14.0.0	Od	50338
			O1	63188
			O2	65934
			O3	69082