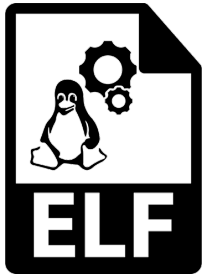
Code Genome: Fingerprinting Code to Build Trustworthy SBOMs

Doug Schales
Dhilung Kirat
Jiyong Jang
Ian Molloy
Ted Habeck
JR Rao

IBM

THE LINUX FOUNDATION




```
$ foo install bar
```

- Signed with a certificate.
- Lists dependencies.

- Do you trust it?

"You can't trust code that you did not totally create yourself."

—Ken Thompson

Reflections on Trusting Trust

To what extent should one trust a statement that a program is free of Trojan horses? Perhaps it is more important to trust the people who wrote the software.

KEN THOMPSON

INTRODUCTION

I thank the ACM for this award. I can't help but feel that I am receiving this honor for timing and serendipity as much as technical merit. UNIX[1] swept into popularity with an industry-wide change from central mainframes to autonomous minis. I suspect that Daniel Bobrow [1] would be here instead of me if he could not afford a PDP-10 and had had to "settle" for a PDP-11. Moreover, the current state of UNIX is the result of the labors of a large number of people.

There is an old adage, "Dance with the one that brought you," which means that I should talk about UNIX. I have not worked on mainstream UNIX in many years, yet I continue to get undeserved credit for the work of others. Therefore, I am not going to talk about UNIX, but I want to thank everyone who has contributed.

That brings me to Dennis Ritchie. Our collaboration has been a thing of beauty. In the ten years that we have worked together, I can recall only one case of miscoordination of work. On that occasion, I discovered that we both had written the same 20-line assembly language program. I compared the sources and was astounded to find that they matched character-for-character. The result of our work together has been far greater than the work that we each contributed.

I am a programmer. On my 1040 form, that is what I put down as my occupation. As a programmer, I write programs. I would like to present to you the cutest program I ever wrote. I will do this in three stages and try to bring it together at the end.

[1] UNIX is a trademark of AT&T Bell Laboratories.

STAGE I

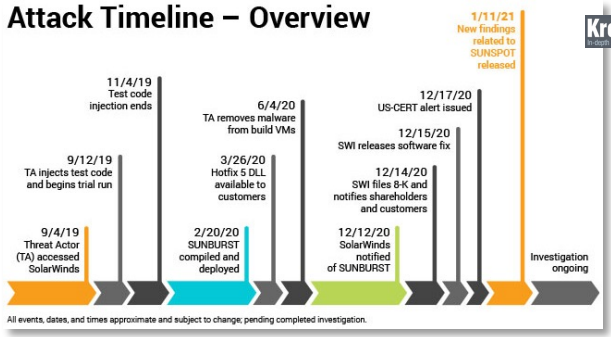
In college, before video games, we would amuse ourselves by posing programming exercises. One of the favorites was to write the shortest self-reproducing program. Since this is an exercise divorced from reality, the usual vehicle was FORTRAN. Actually, FORTRAN was the language of choice for the same reason that three-legged races are popular.

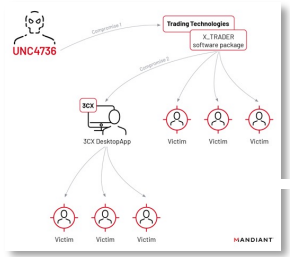
More precisely stated, the problem is to write a source program that, when compiled and executed, will produce as output an exact copy of its source. If you have never done this, I urge you to try it on your own. The discovery of how to do it is a revelation that far surpasses any benefit obtained by being told how to do it. The part about "shortest" was just an incentive to demonstrate skill and determine a winner.

Figure 1 shows a self-reproducing program in the C[3] programming language. (The purist will note that the program is not precisely a self-reproducing program, but will produce a self-reproducing program.) This entry is much too large to win a prize, but it demonstrates the technique and has two important properties that I need to complete my story: 1) This program can be easily written by another program. 2) This program can contain an arbitrary amount of excess baggage that will be reproduced along with the main algorithm. In the example, even the comment is reproduced.

Supply Chain Attacks

- SolarWinds (2019-2021) est. cost > $100B
  - Malicious code (backdoor) pushed out through updates
- Dependency confusion (Feb 2021)
  - Private vs public packages (npm, PyPi, RubyGems)
- Codecov (Apr 2021)
  - DevOps tool. Vulnerability in CI. Bash uploader modified
- Kaseya (Jul 2021) ransom $70M
  - IT solutions, including VSA (remote monitoring and management software) to deliver REvil ransomware
- Protestware (Mar 2022)
  - Popular NPM package wiped files in Russia and Belarus
- 3CX (Mar 2023)
  - Backdoor implanted into Windows and macOS due to secondary supply chain attack





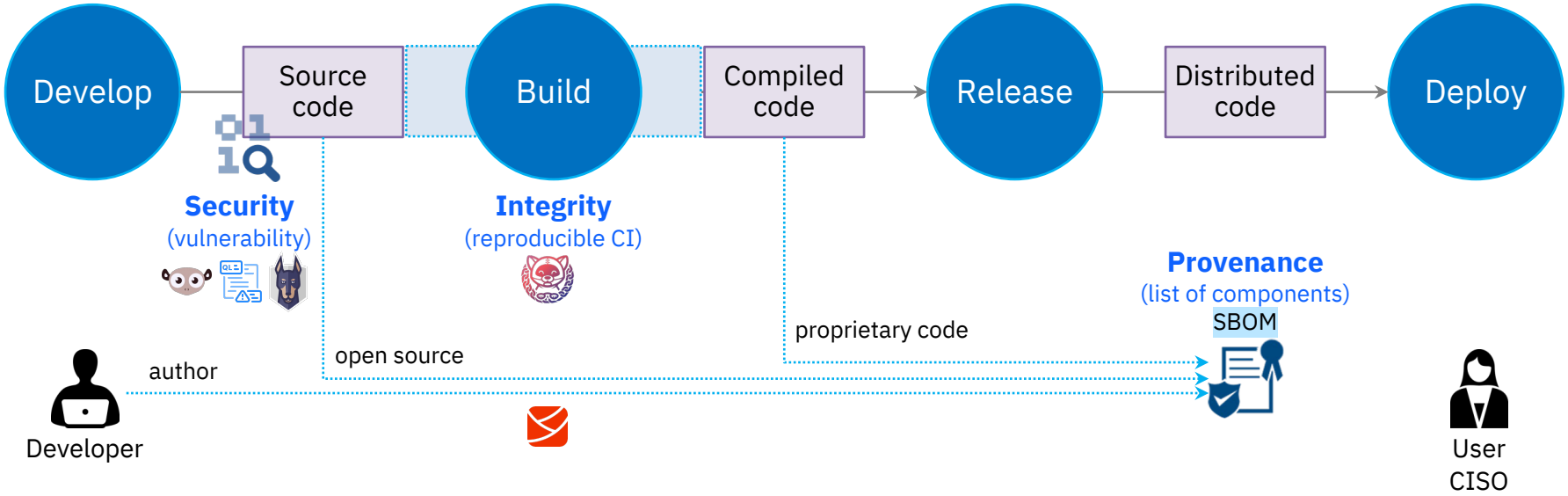
Codecov breach impacted 'hundreds' of customer networks: report

ZDNet

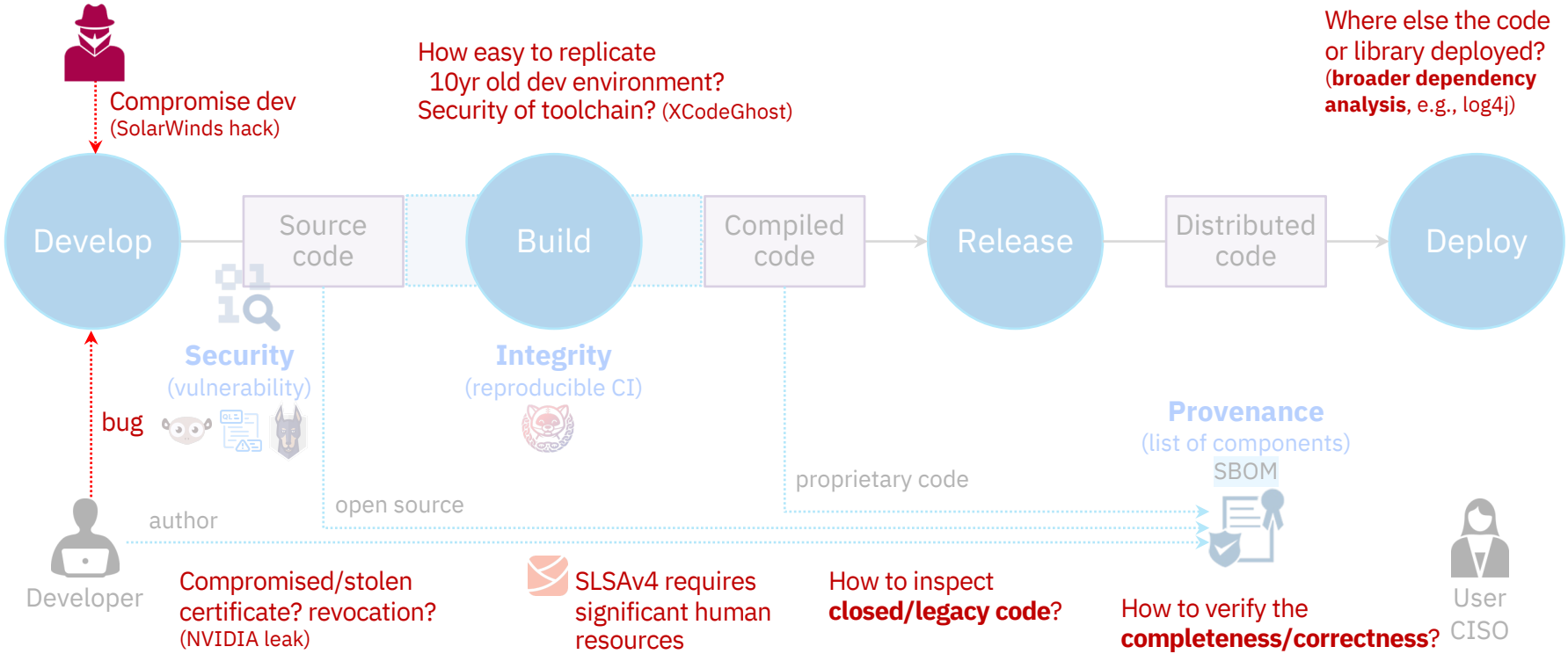
Updated: Reports suggest the initial hack may have led to a more extensive supply chain attack.

CISA-FBI Guidance for MSPs and their Customers Affected by the Kaseya VSA Supply-Chain Ransomware Attack

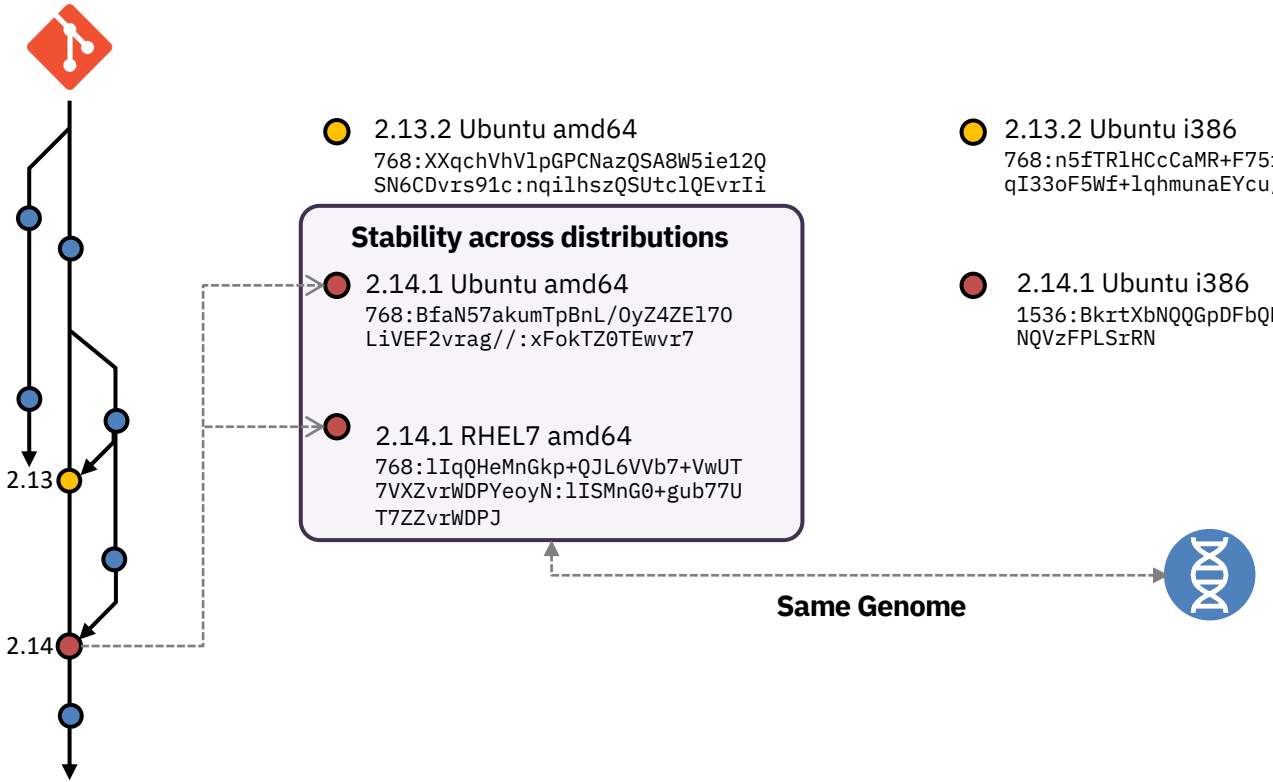
Supply Chain Security: Industry approach to protecting CI/CD pipelines



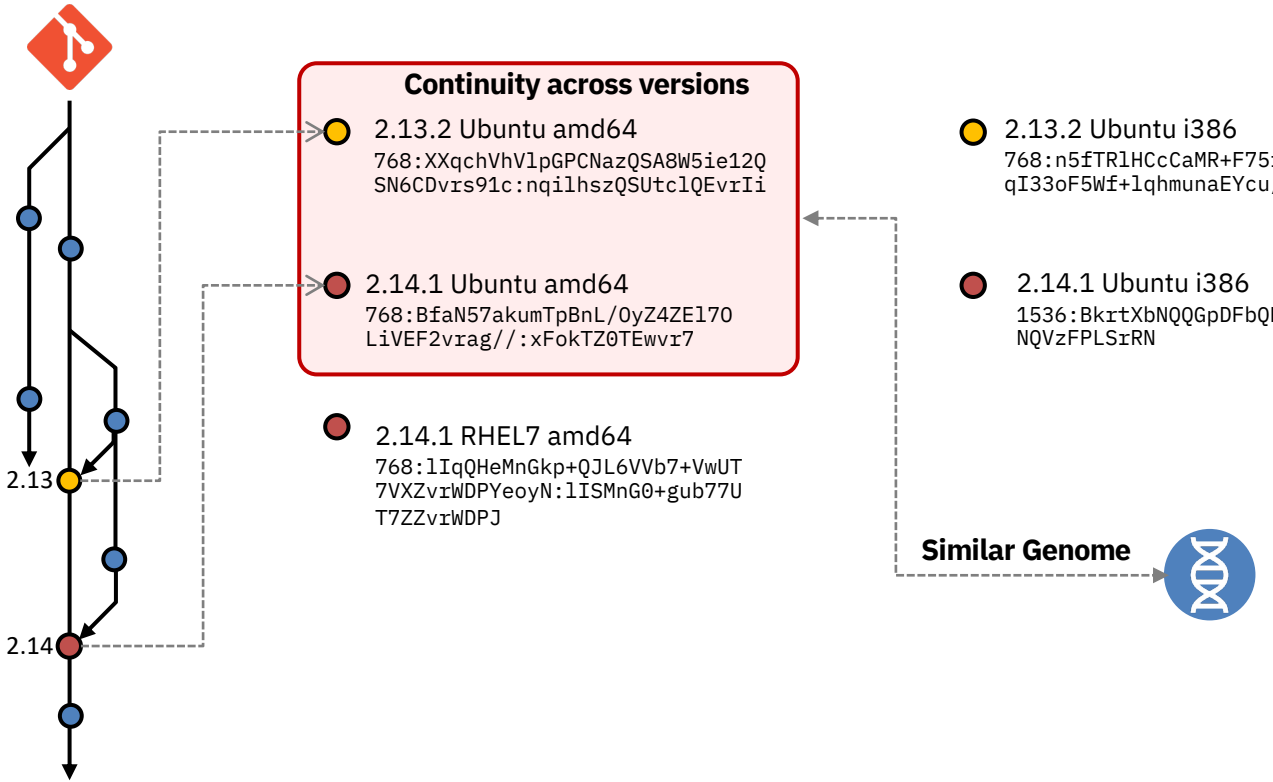
Supply Chain Security: Open security issues and residual risks



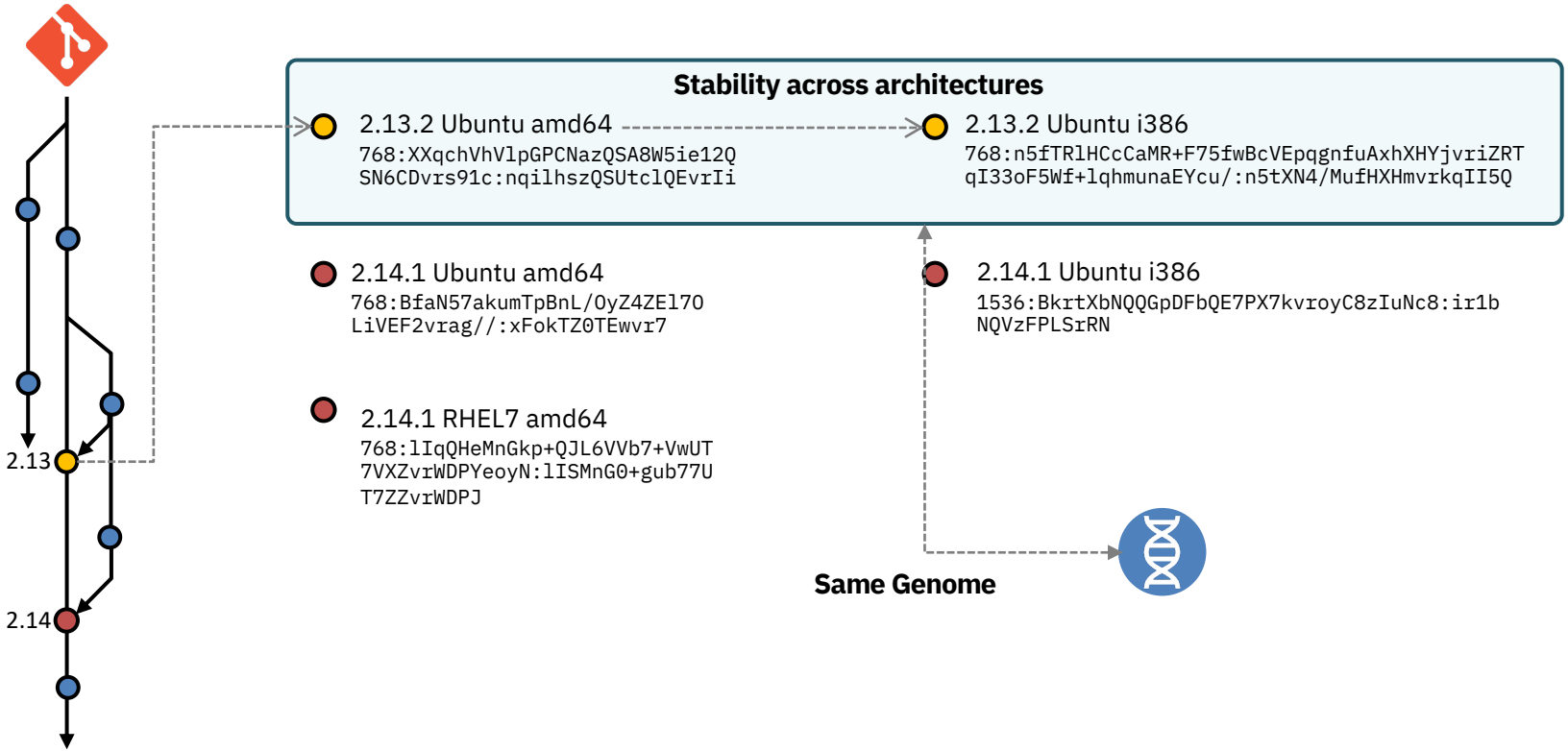
Software Fingerprints for Software Assurance



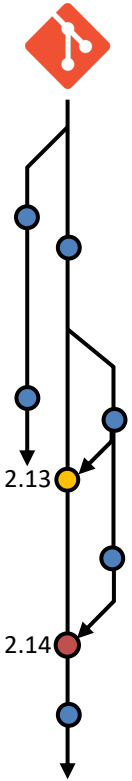
Software Fingerprints for Software Assurance



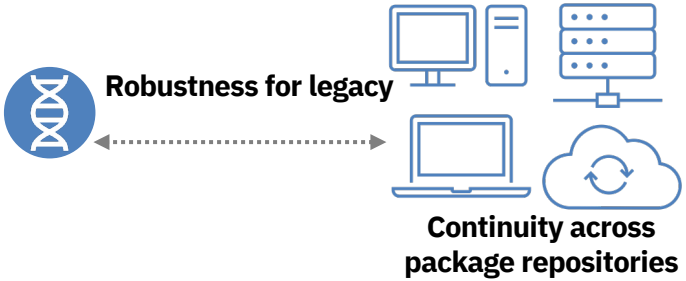
Software Fingerprints for Software Assurance



Software Fingerprints for Software Assurance



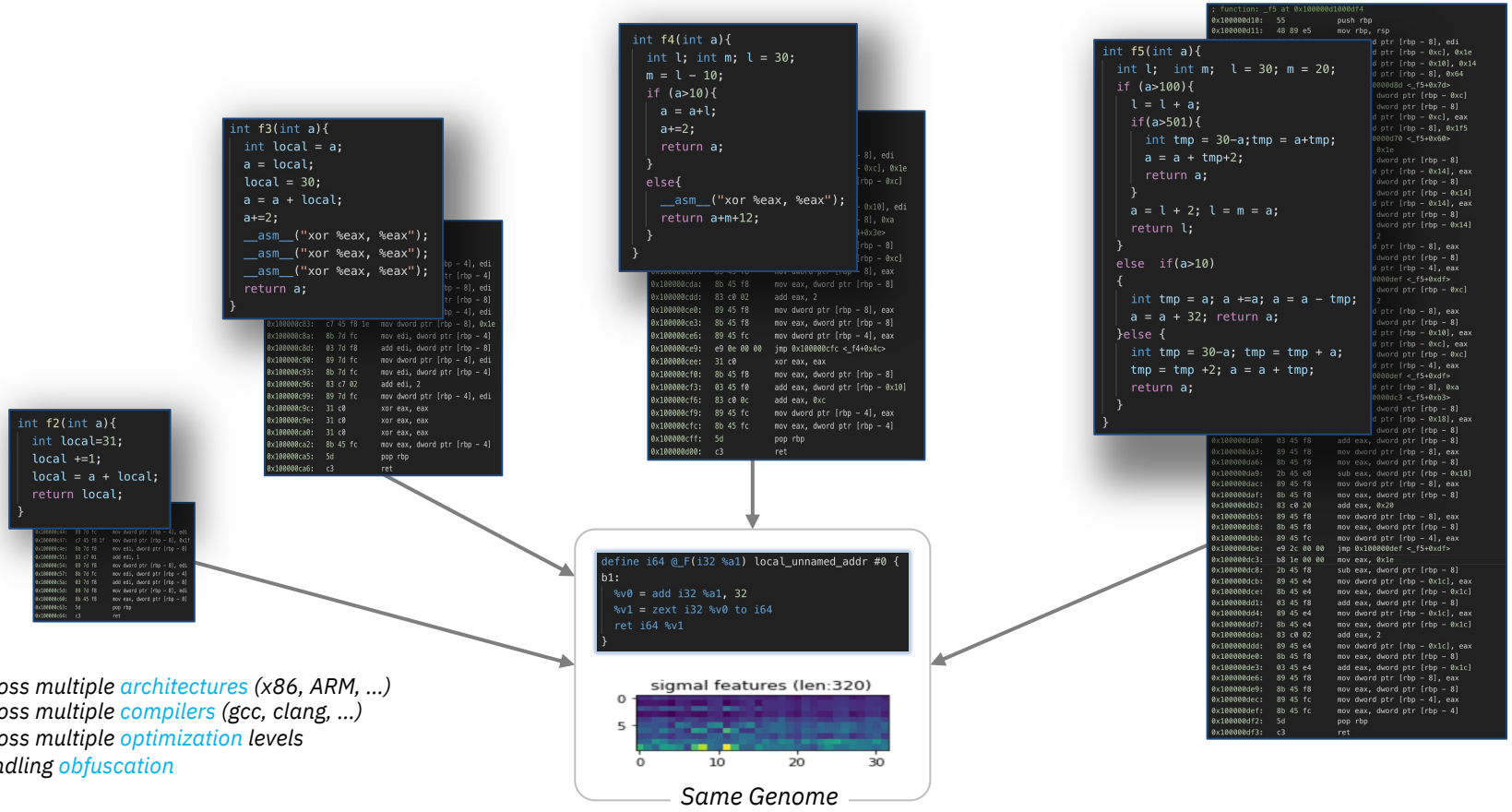
- 2.13.2 Ubuntu amd64
  `768:XXqchVhVlpGPCNazQSA8W5ie12Q SN6CDvrs91c:nqilhszQSUtclQEvrIi`
- 2.13.2 Ubuntu i386
  `768:n5fTRlHCcCaMR+F75fwBcVEpqgnfuAxhXHYjvriZRT qI33oF5Wf+lqhmunaEYcu/:n5tXN4/MufHXHmvrkqII5Q`
- 2.14.1 Ubuntu amd64
  `768:BfaN57akumTpBnL/OyZ4ZEl7O LiVEF2vrag//:xFokTZ0TEwvr7`
- 2.14.1 Ubuntu i386
  `1536:BkrtXbNQQGpDFbQE7PX7kvroyC8zIuNc8:ir1b NQVzFPLSrRN`
- 2.14.1 RHEL7 amd64
  `768:lIqQHeMnGkp+QJL6VVb7+VwUT 7VXZvrWDPYeoyN:lISMnG0+gub77U T7ZZvrWDPJ`



Timeline

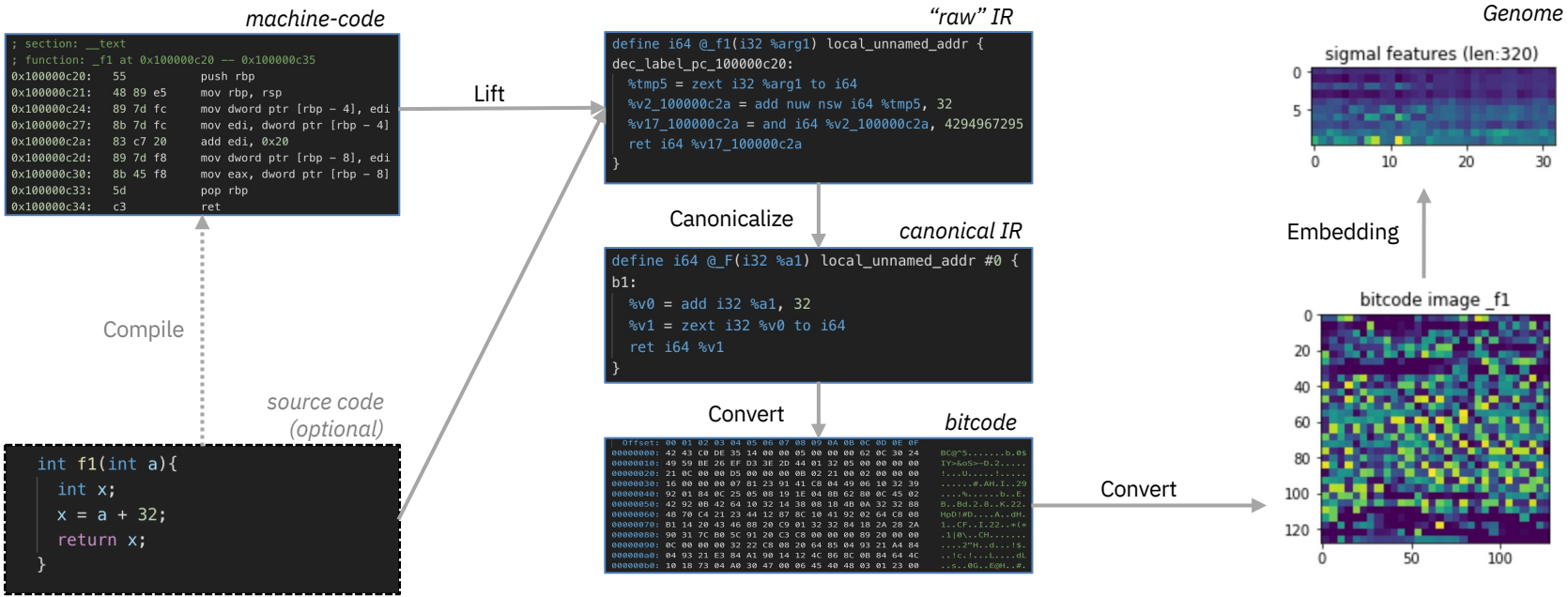
2011	BitShred: Feature Hashing Malware for Scalable Triage and Semantic Analysis
2012	ReDeBug: Finding Unpatched Code Clones in Entire OS Distributions
2013	Sigmal: A static signal processing-based malware triage Towards automatic software lineage inference
2015	Malgene: Automatic extraction of malware analysis evasion signature Experimental Study of Fuzzy Hashing in Malware Clustering Analysis
2017	Building a better vulnerability scanner for Docker
2020	Consolidating structured and unstructured security and threat intelligence with knowledge graphs
2021	Software intelligence as-a-service Find log4j buried on legacy systems
2022	Code Genome: Linux Foundation Member's Summit
2023 *(today)*	**Code** Genome: Open Source Summit North America

Code Genome: Semantically meaningful fingerprint



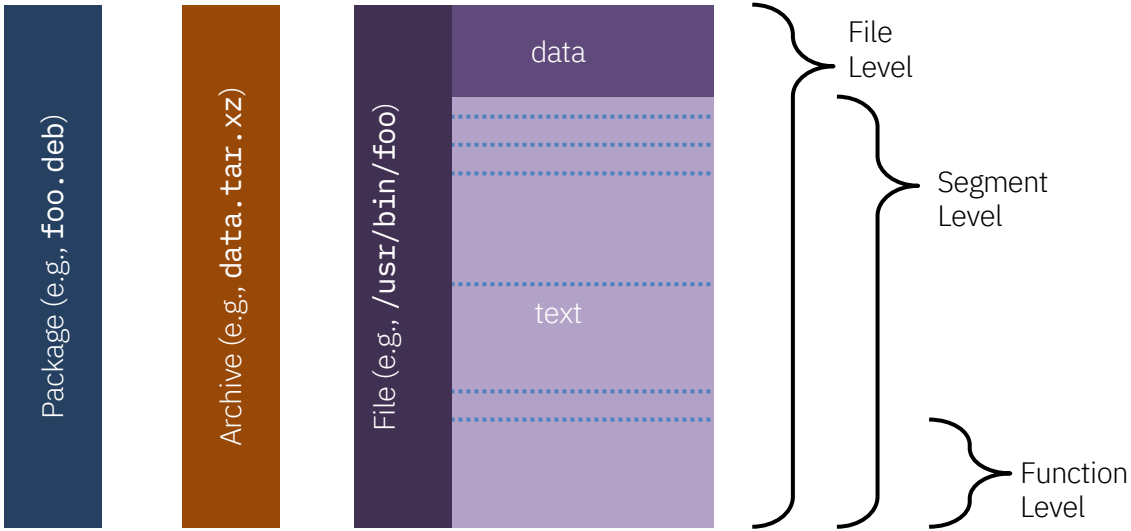
- *Across multiple architectures (x86, ARM, ...)*
- *Across multiple compilers (gcc, clang, ...)*
- *Across multiple optimization levels*
- *Handling obfuscation*

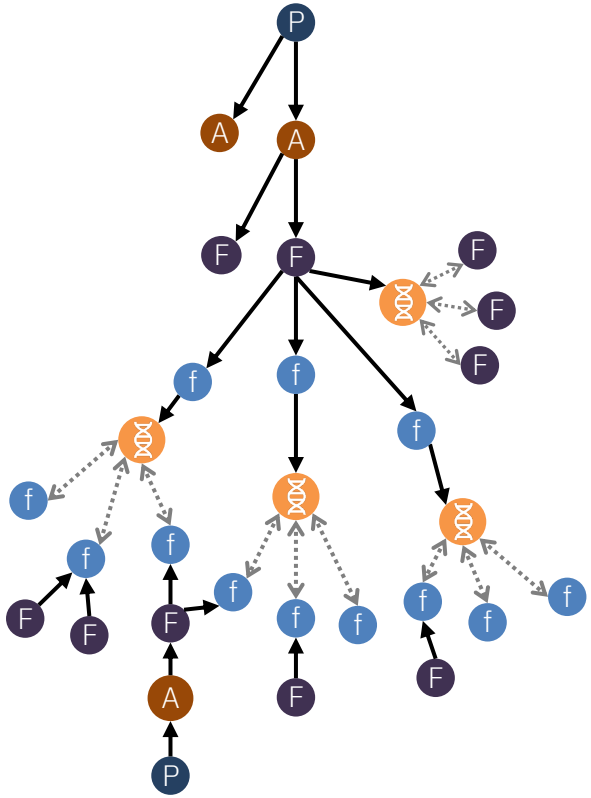
Key Idea: Code Genome construction



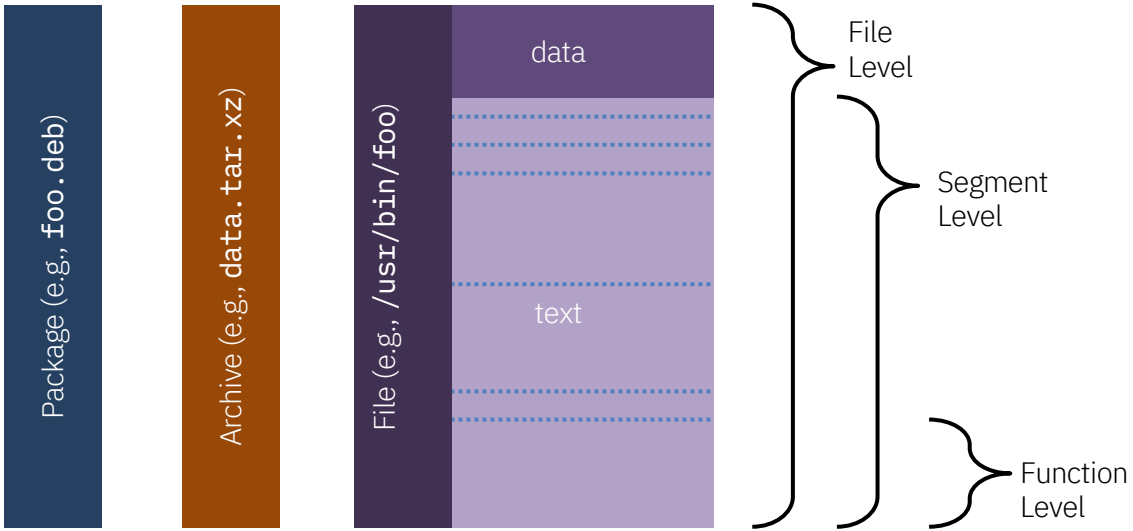
Genome can be constructed from closed-source/legacy code where source code is not easily available.

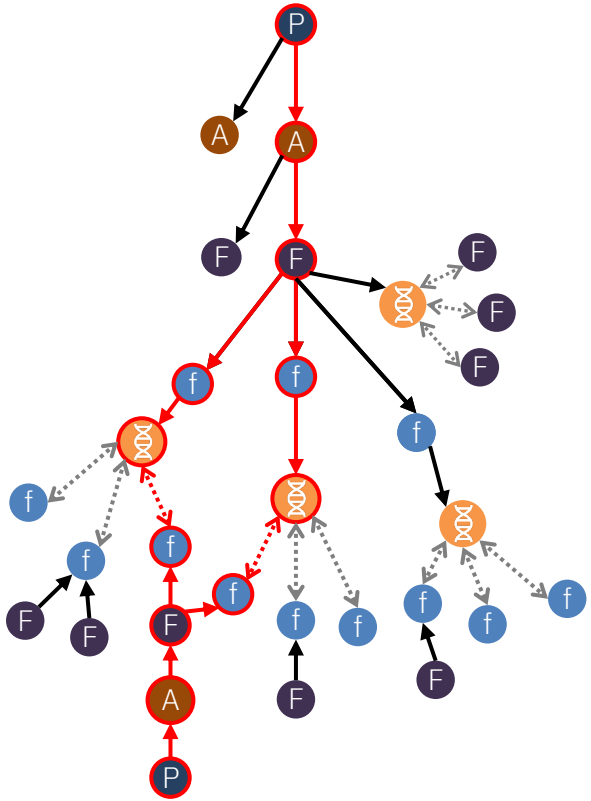
Granularity



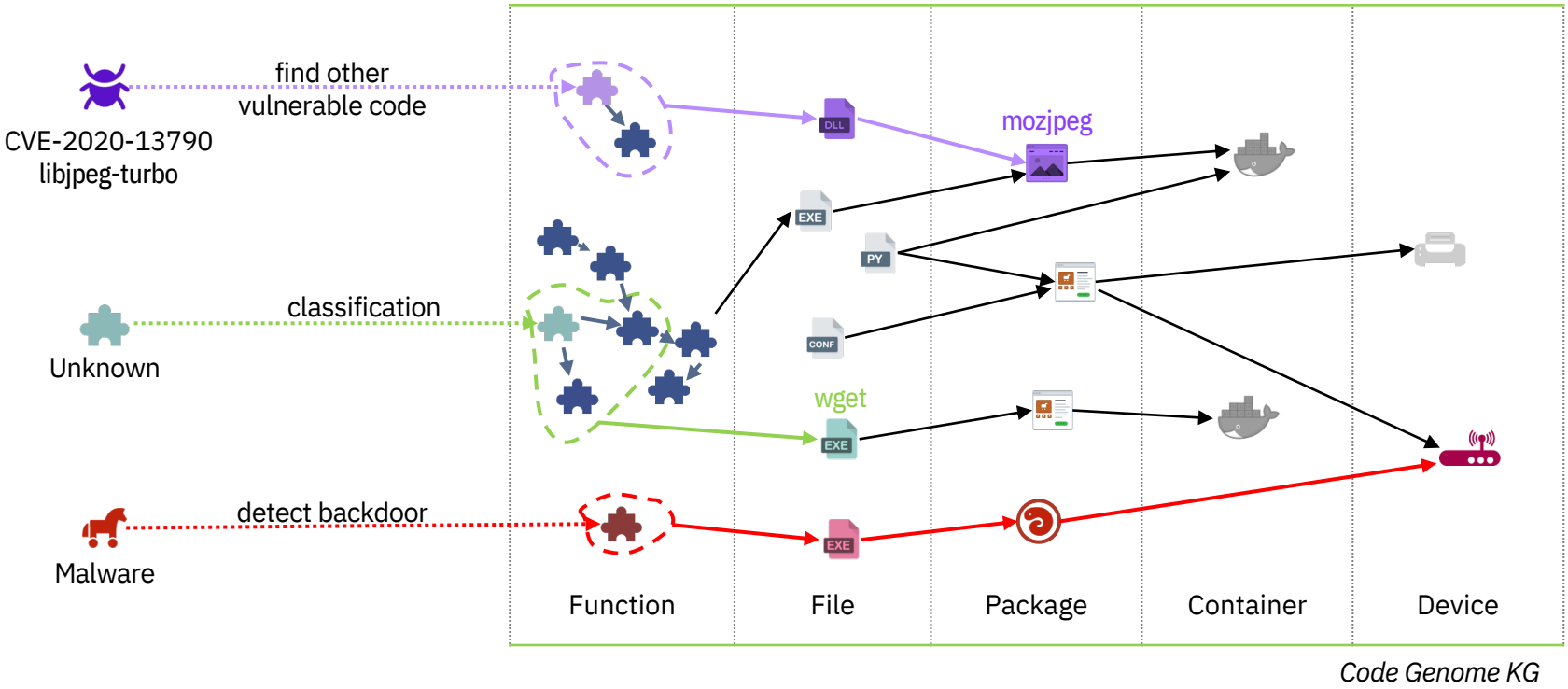


Granularity



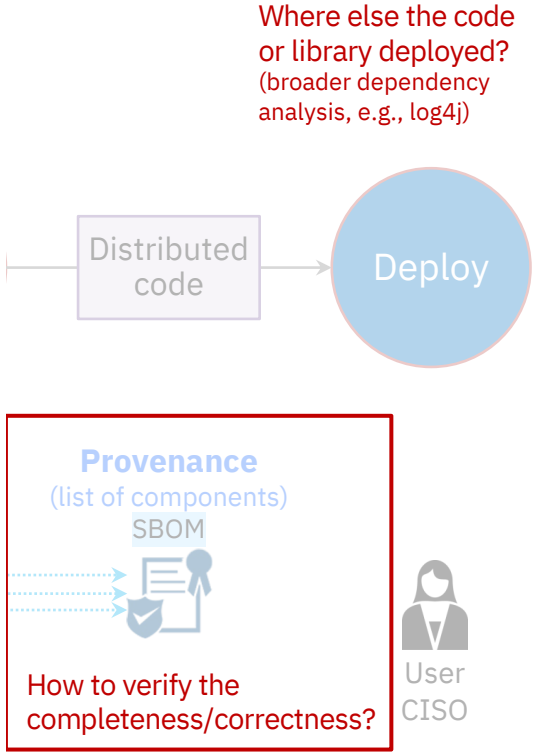


Knowledge Graph: Code Genome and Use Cases



Use Case: SBOM Verification

- Problem
  - Each vendor creates SBOM of their own software including open-source and closed-source components.
  - How can we verify its *correctness* (containing incorrect library mistakenly/maliciously) and *completeness* (missing library)?
- Value
  - Given software, we can verify (generate) SBOM
  - Support closed-source and legacy software without requiring source code access
  - Help developers generate correct SBOM
  - Vet software before integrating/deploying into a product



Trust but Verify SBOM: Metadata vs. Code

"Unfortunately, some images – such as the official node image on Docker Hub – incorrectly report the version of OpenSSL that's used by the Node.js runtime."

https://www.chainguard.dev/unchained/mitigating-critical-openssl-vulnerability-with-chainguard

$ sbom generation tools

Dockerfile > ...

```
1   FROM ubuntu:focal
2
3   RUN apt-get update
4   RUN apt-get install -y wget
5
6   RUN apt-get update
7
```

```
"bom-ref": "pkg:wget@1.20.3-1ubuntu2",
"type": "library",
"name": "wget",
"version": "1.20.3-1ubuntu2",
"licenses": [
  {
    "license": {
      "name": "UNKNOWN"
    }
  }
],
```

Dockerfile > ...

```
1   FROM ubuntu:focal
2
3   RUN apt-get update
4   RUN apt-get install -y wget
5
6   RUN mv /var/lib/dpkg/status /var/lib/dpkg/status.bak
7   RUN touch /var/lib/dpkg/status
8
9   RUN apt-get update
10
```

delete dpkg DB

```
sbom/docker ❯ grep wget sbom.dpkg.json
sbom/docker ❯
```

```
/usr/local/bin$ go version -m kube-proxy | head
kube-proxy: go1.19.7
path k8s.io/kubernetes/cmd/kube-proxy
mod k8s.io/kubernetes (devel)
dep github.com/NYTimes/gziphandler v1.1.1
dep github.com/beorn7/perks v1.0.1
dep github.com/blang/semver/v4 v4.0.0
dep github.com/cenkalti/backoff/v4 v4.1.3
dep github.com/cespare/xxhash/v2 v2.1.2
```

Modify strings in the binary

```
/usr/local/bin$ go version -m kube-proxy | head
kube-proxy: go1.19.7
path k8s.io/kubernetes/cmd/kube-proxy
mod k8s.io/kubernetes (devel)
dep github.com/hackers/gziphandler v1.1.1
dep github.com/beorn7/perks v1.0.1
dep github.com/blang/semver/v4 v4.0.0
dep github.com/cenkalti/backoff/v4 v4.1.3
dep github.com/cespare/xxhash/v2 v2.1.2
```

CISA: Types of SBOM

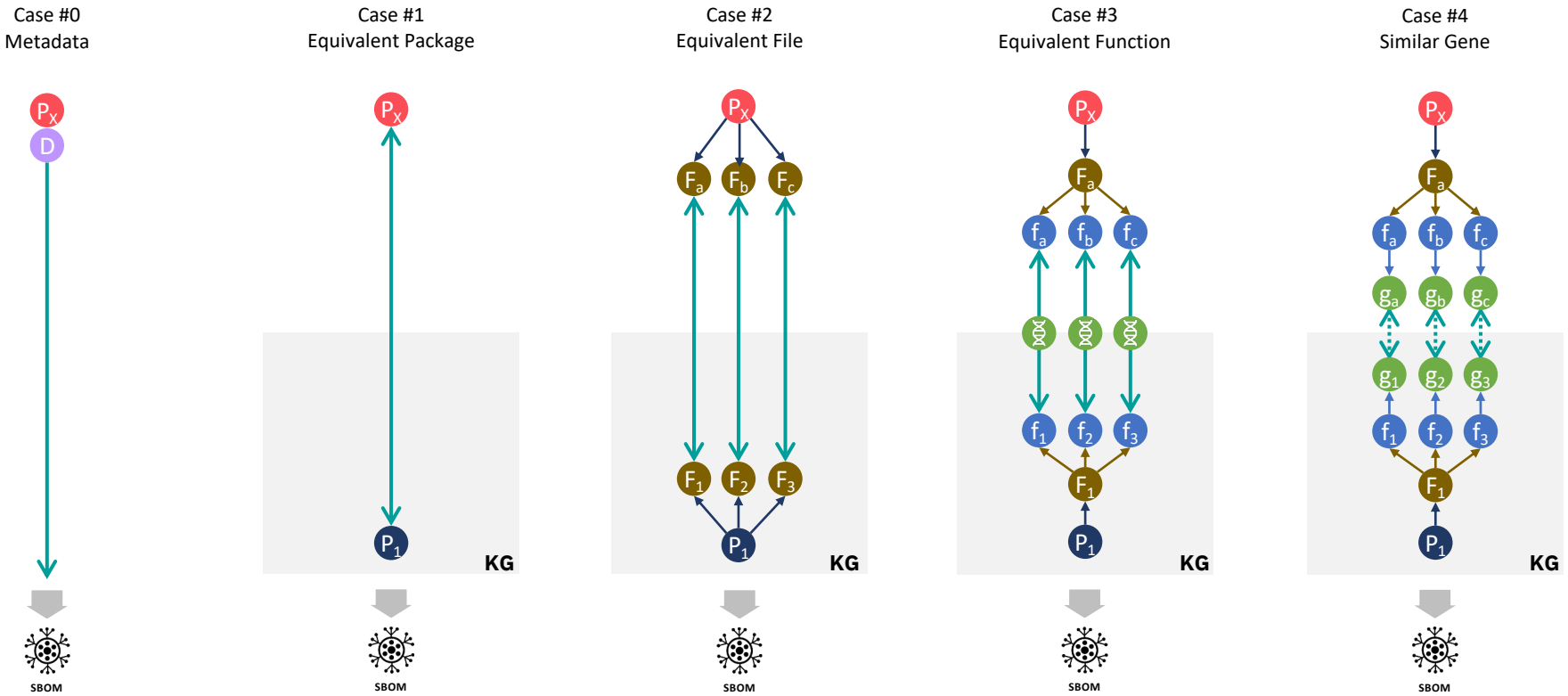
Table 1: SBOM Type Definition and Composition

SBOM Type	Definition	Data Description
Design	SBOM of intended, planned software project or product with included components (some of which may not yet exist) for a new software artifact.	Typically derived from a design specification, RFP, or initial concept.
Source	SBOM created directly from the development environment, source files, and included dependencies used to build an product artifact.	Typically generated from software composition analysis (SCA) tooling, with manual clarifications.
Build	SBOM generated as part of the process of building the software to create a releasable artifact (e.g., executable or package) from data such as source files, dependencies, built components, build process ephemeral data, and other SBOMs.	Typically generated as part of a build process. May consist of integrated intermediate Build and Source SBOMs for a final release artifact SBOM.
Analyzed	SBOM generated through analysis of artifacts (e.g., executables, packages, containers, and virtual machine images) after its build. Such analysis generally requires a variety of heuristics. In some contexts, this may also be referred to as a “3rd party” SBOM.	Typically generated through analysis of artifacts by 3rd party tooling.
Deployed	SBOM provides an inventory of software that is present on a system. This may be an assembly of other SBOMs that combines analysis of configuration options, and examination of execution behavior in a (potentially simulated) deployment environment.	Typically generated by recording the SBOMs and configuration information of artifacts that have been installed on systems.

Benefits

Analyzed	- Provides visibility without an active development environment, such as legacy firmware artifacts. - Does not need access to the build process. - Can help verify SBOM data from other sources. - May find hidden dependencies missed by other SBOM type creation tools.
Deployed	- Highlights software components installed on a system, including other configurations and system components used to run an application.

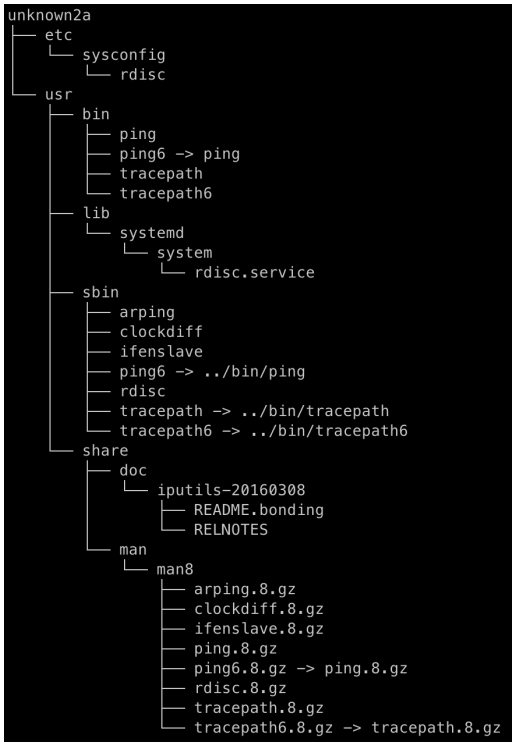
SBOM for Binaries: Beyond metadata for generation and verification



Demo

Demo 1: SBOM generation for an unknown `rpm` package

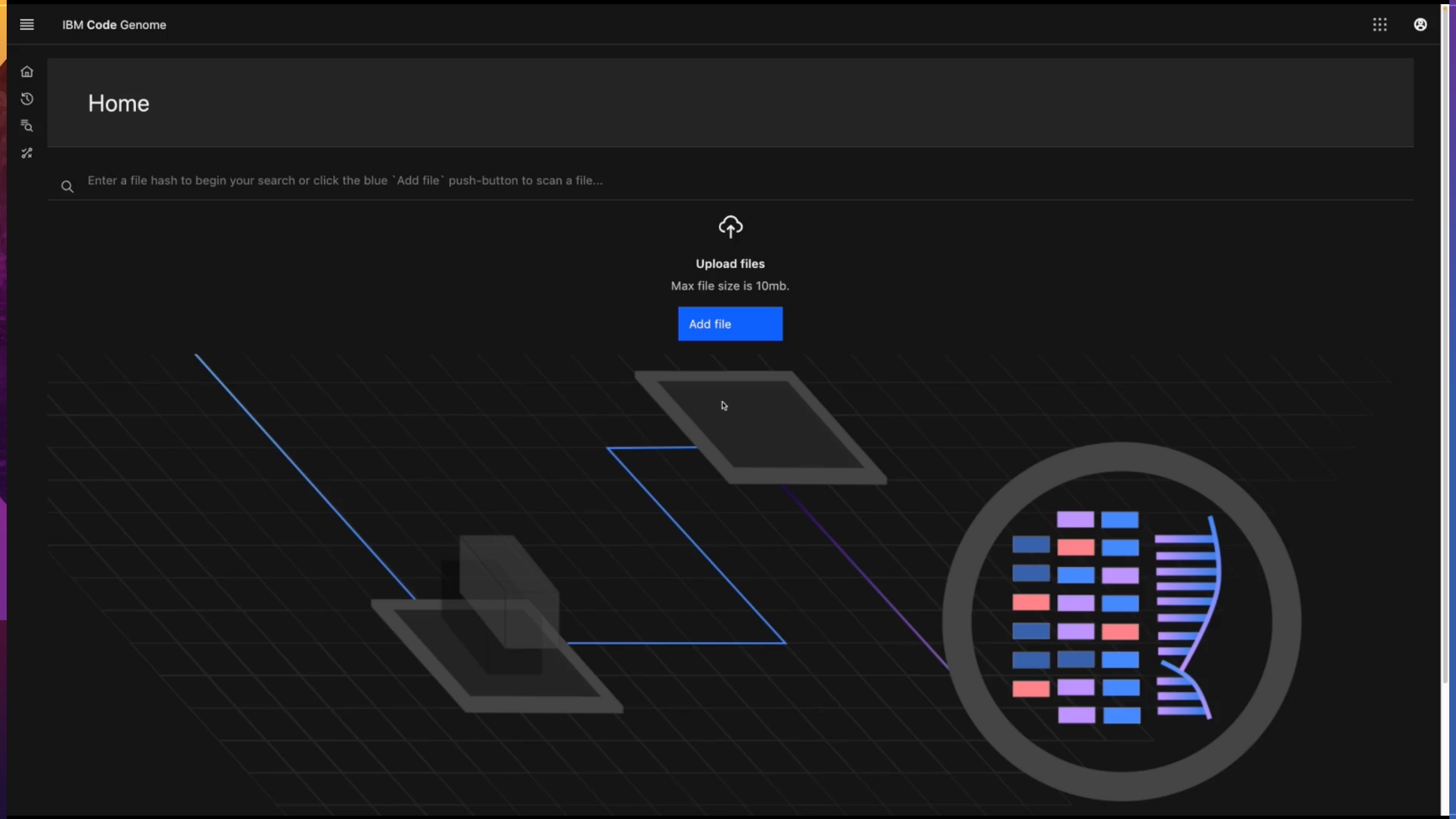
Custom `rpm` package

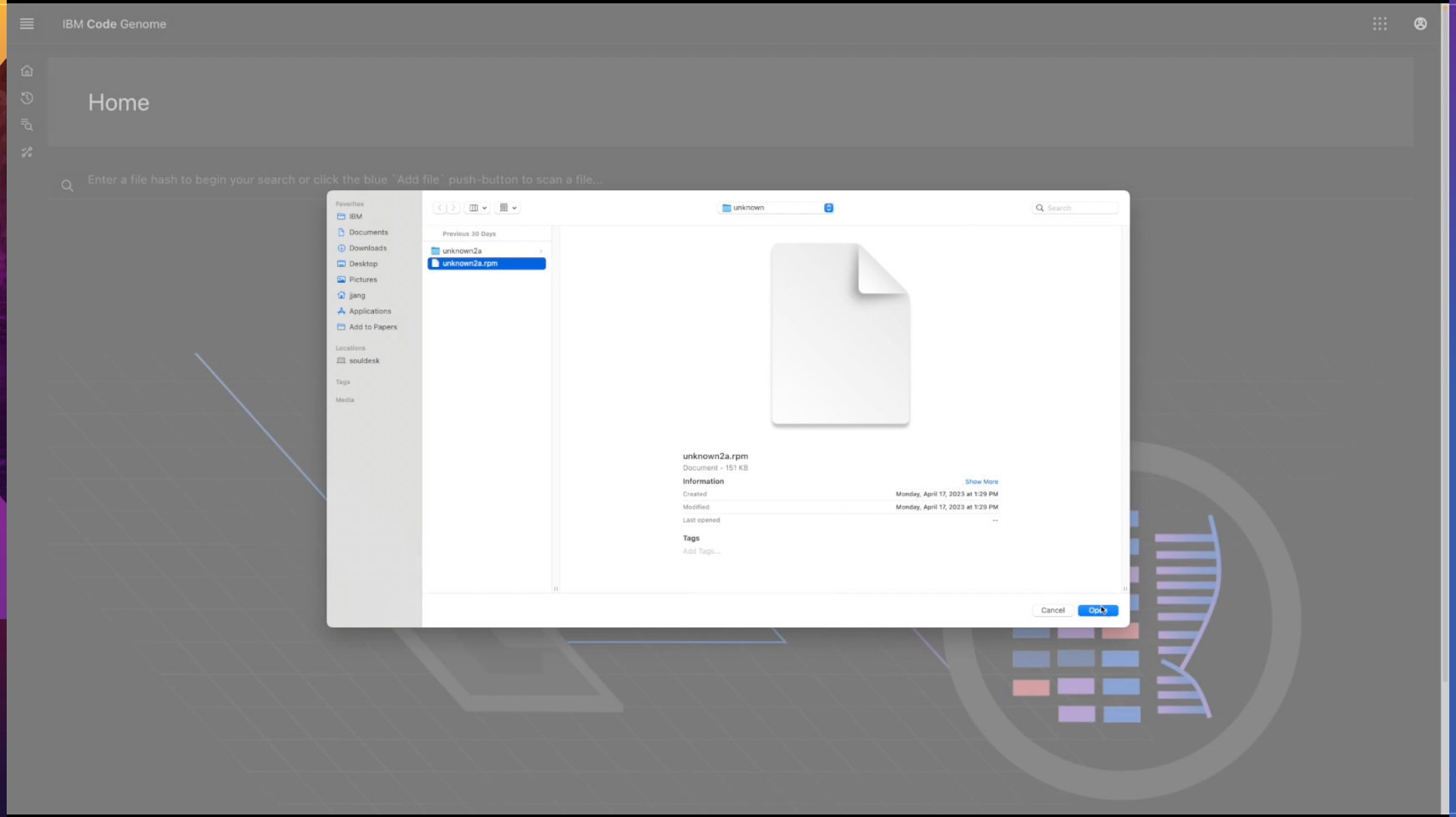


SBOM generated by Code Genome

Component	Version	License
arping	20160308	BSD and GPLv2+
clockdiff	20160308	BSD and GPLv2+
ifenslave	20160308	BSD and GPLv2+
iputils	20160308	BSD and GPLv2+
ping	20160308	BSD and GPLv2+
rdisc	20160308	BSD and GPLv2+
tracepath	20160308	BSD and GPLv2+
tracepath6	20160308	BSD and GPLv2+

Integrating with other SBOM analysis platforms


IBM Code Genome
Home
Enter a file hash to begin your search or click the blue `Add file` push-button to scan a file...
Upload files
Max file size is 10mb.
Add file


IBM Code Genome
Home
Enter a file hash to begin your search or click the blue `Add file` push-button to scan a file...
Favorites
IBM
Documents
Downloads
Desktop
Pictures
jjang
Applications
Add to Papers
Locations
souldesk
Tags
Media
unknown
Search
Previous 30 Days
unknown2a
unknown2a.rpm
unknown2a.rpm
Document – 151 KB
Information
Show More
Created
Monday, April 17, 2023 at 1:29 PM
Modified
Monday, April 17, 2023 at 1:29 PM
Last opened
--
Tags
Add Tags...
Cancel
Open

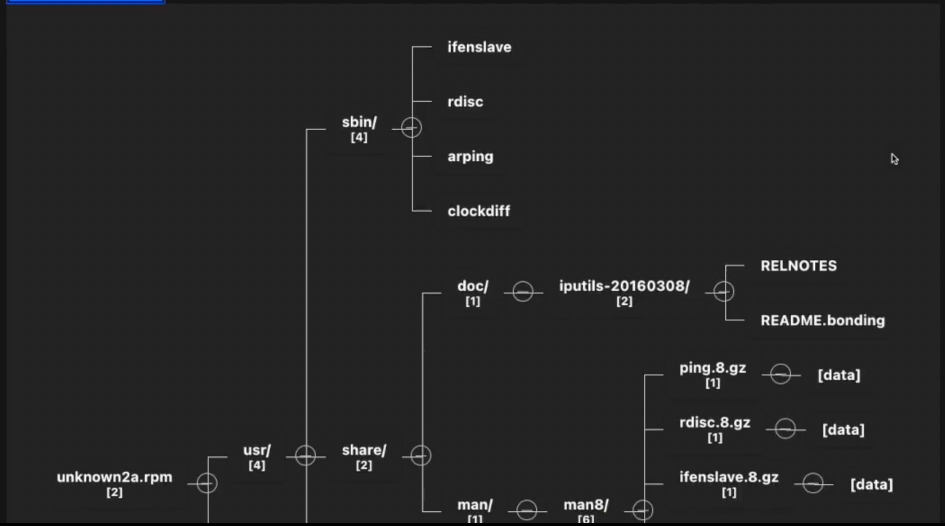
IBM Code Genome

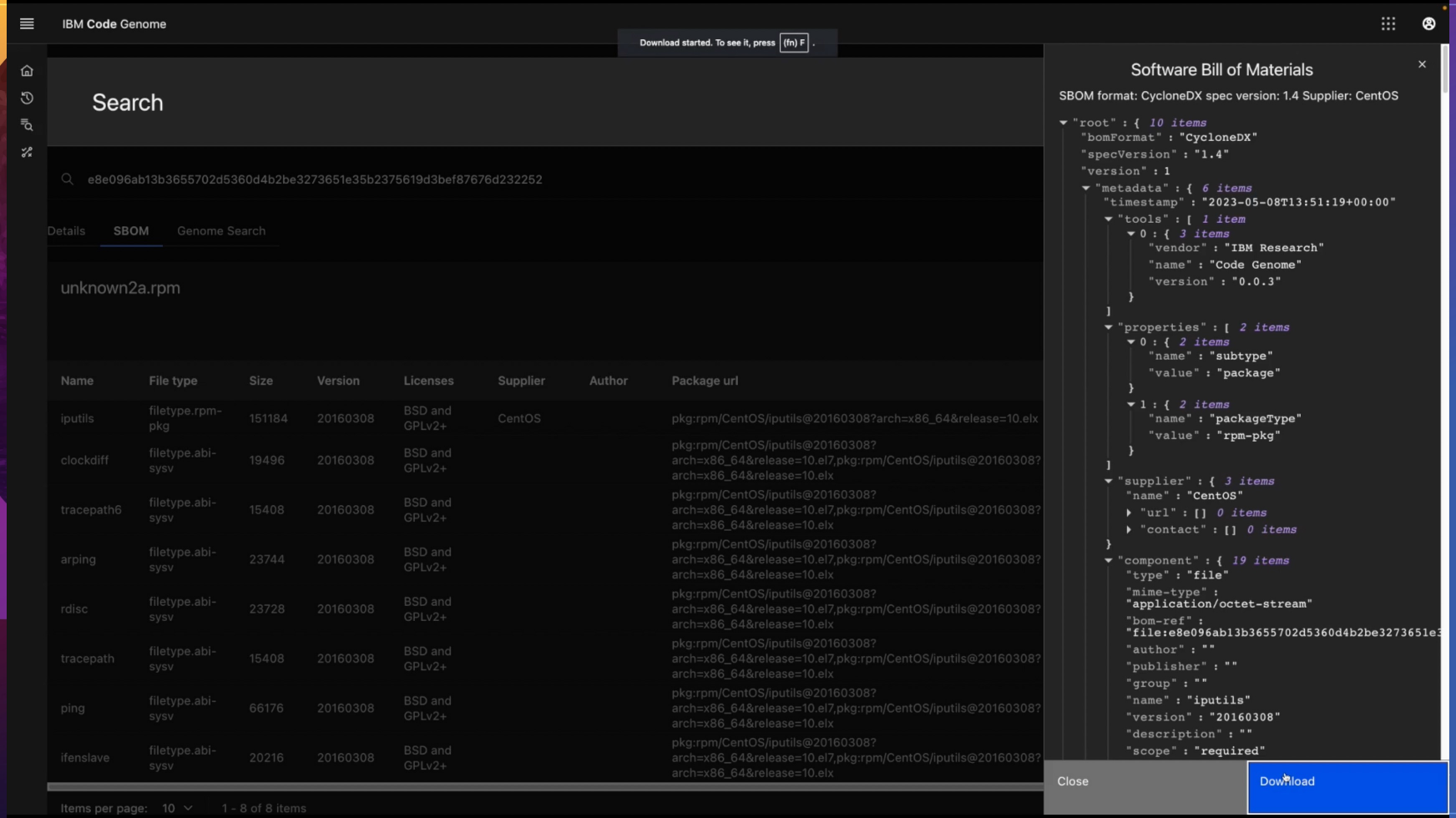
Details | SBOM | Genome Search

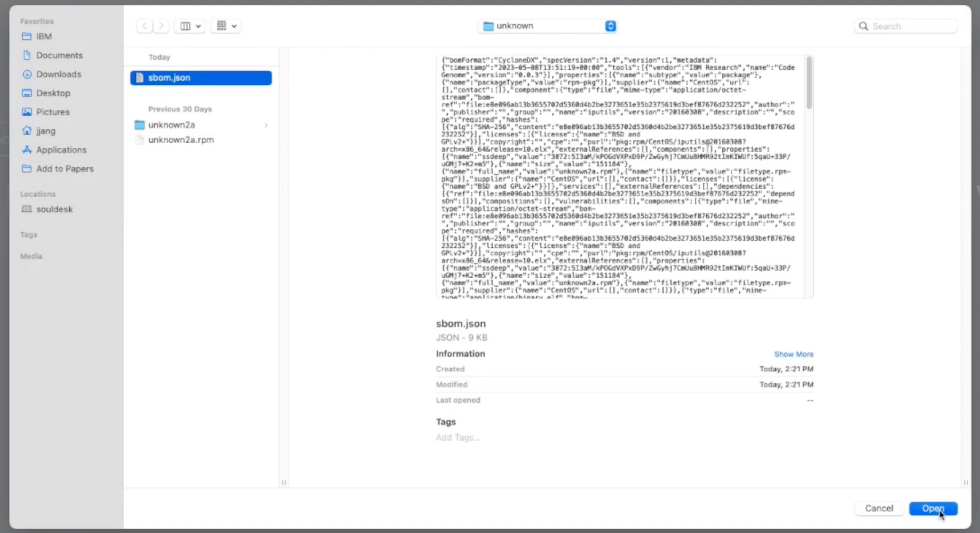
File Name	unknown2a.rpm
File Hash	e8e096ab13b3655702d5360d4b2be3273651e35b2375619d3bef87676d232252
File Type	rpm-pkg
File size	151184

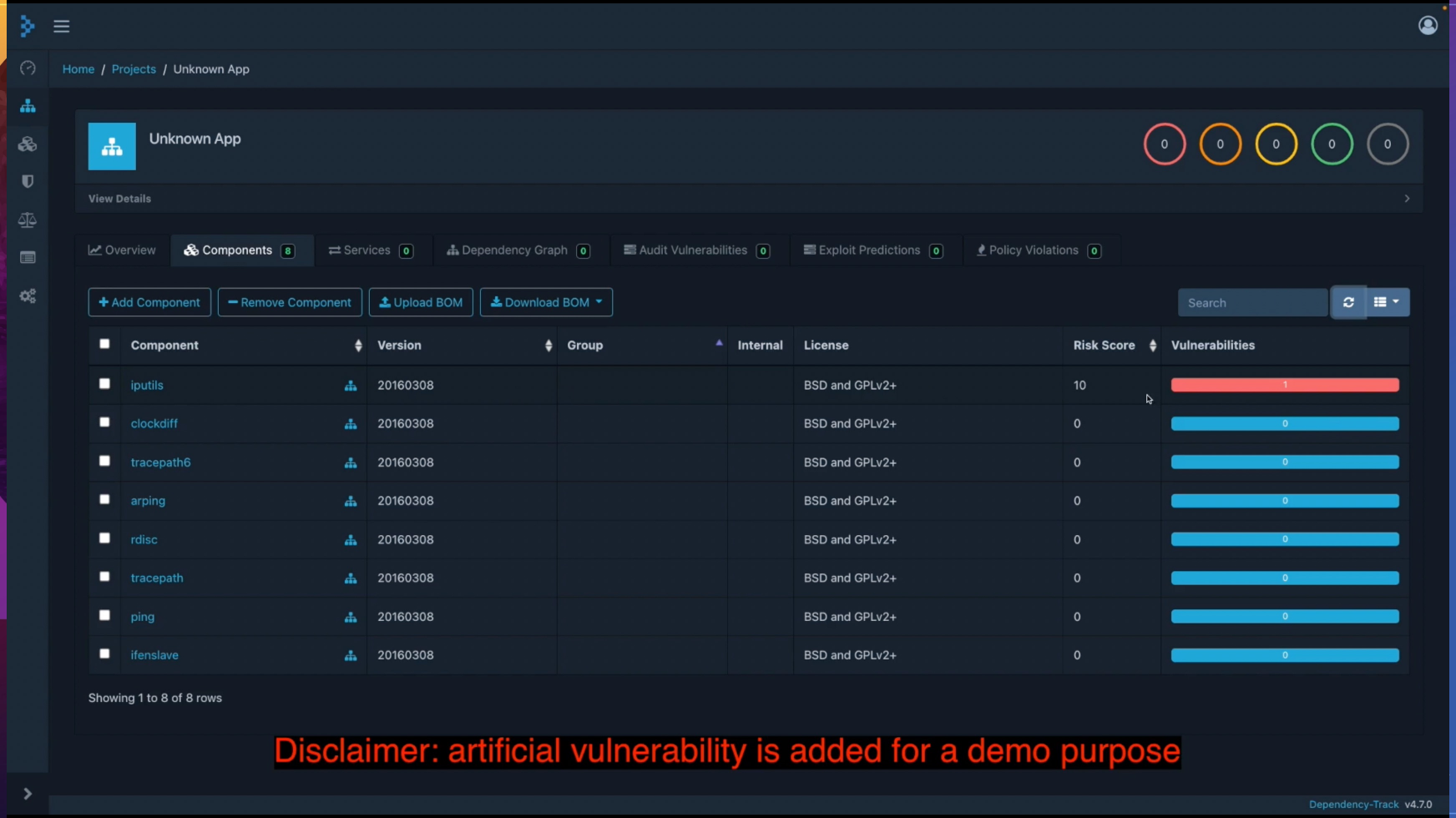
File content

Collapse All

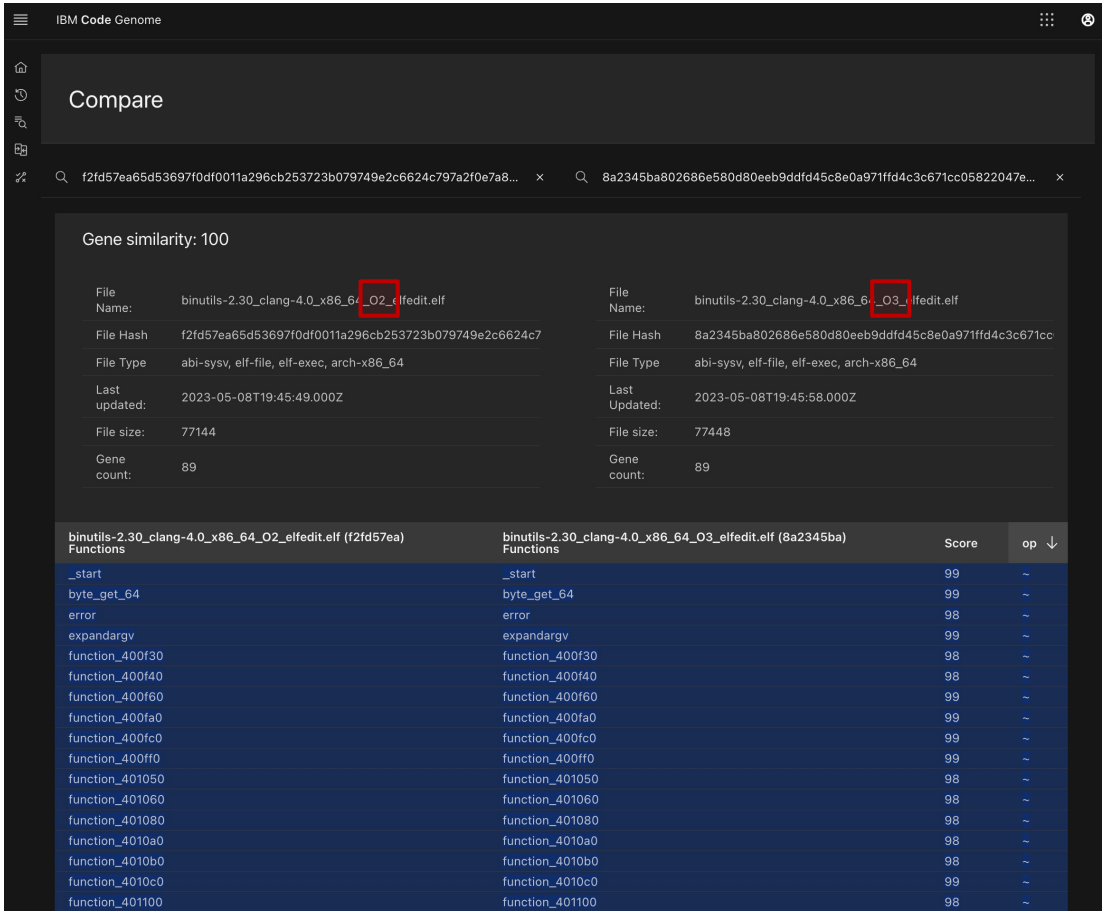



IBM Code Genome
Download started. To see it, press (fn) F .
Search
e8e096ab13b3655702d5360d4b2be3273651e35b2375619d3bef87676d232252
Details
SBOM
Genome Search
unknown2a.rpm
Name
File type
Size
Version
Licenses
Supplier
Author
Package url
iputils
filetype.rpm-pkg
151184
20160308
BSD and GPLv2+
CentOS
pkg:rpm/CentOS/iputils@20160308?arch=x86_64&release=10.elx
clockdiff
filetype.abi-sysv
19496
20160308
BSD and GPLv2+
pkg:rpm/CentOS/iputils@20160308?arch=x86_64&release=10.el7,pkg:rpm/CentOS/iputils@20160308?arch=x86_64&release=10.elx
tracepath6
filetype.abi-sysv
15408
20160308
BSD and GPLv2+
pkg:rpm/CentOS/iputils@20160308?arch=x86_64&release=10.el7,pkg:rpm/CentOS/iputils@20160308?arch=x86_64&release=10.elx
arping
filetype.abi-sysv
23744
20160308
BSD and GPLv2+
pkg:rpm/CentOS/iputils@20160308?arch=x86_64&release=10.el7,pkg:rpm/CentOS/iputils@20160308?arch=x86_64&release=10.elx
rdisc
filetype.abi-sysv
23728
20160308
BSD and GPLv2+
pkg:rpm/CentOS/iputils@20160308?arch=x86_64&release=10.el7,pkg:rpm/CentOS/iputils@20160308?arch=x86_64&release=10.elx
tracepath
filetype.abi-sysv
15408
20160308
BSD and GPLv2+
pkg:rpm/CentOS/iputils@20160308?arch=x86_64&release=10.el7,pkg:rpm/CentOS/iputils@20160308?arch=x86_64&release=10.elx
ping
filetype.abi-sysv
66176
20160308
BSD and GPLv2+
pkg:rpm/CentOS/iputils@20160308?arch=x86_64&release=10.el7,pkg:rpm/CentOS/iputils@20160308?arch=x86_64&release=10.elx
ifenslave
filetype.abi-sysv
20216
20160308
BSD and GPLv2+
pkg:rpm/CentOS/iputils@20160308?arch=x86_64&release=10.el7,pkg:rpm/CentOS/iputils@20160308?arch=x86_64&release=10.elx
Items per page: 10
1 - 8 of 8 items
Software Bill of Materials
SBOM format: CycloneDX spec version: 1.4 Supplier: CentOS
"root" : { 10 items
"bomFormat" : "CycloneDX"
"specVersion" : "1.4"
"version" : 1
"metadata" : { 6 items
"timestamp" : "2023-05-08T13:51:19+00:00"
"tools" : [1 item
0 : { 3 items
"vendor" : "IBM Research"
"name" : "Code Genome"
"version" : "0.0.3"
}
]
"properties" : [2 items
0 : { 2 items
"name" : "subtype"
"value" : "package"
}
1 : { 2 items
"name" : "packageType"
"value" : "rpm-pkg"
}
]
"supplier" : { 3 items
"name" : "CentOS"
"url" : [] 0 items
"contact" : [] 0 items
}
"component" : { 19 items
"type" : "file"
"mime-type" : "application/octet-stream"
"bom-ref" : "file:e8e096ab13b3655702d5360d4b2be3273651e3
"author" : ""
"publisher" : ""
"group" : ""
"name" : "iputils"
"version" : "20160308"
"description" : ""
"scope" : "required"
Close
Download


Upload BOM
No file chosen
Browse
Cancel
Reset
Upload
Favorites
IBM
Documents
Downloads
Desktop
Pictures
jiang
Applications
Add to Papers
Locations
souldesk
Tags
Media
unknown
Search
Today
sbom.json
Previous 30 Days
unknown2a
unknown2a.rpm
sbom.json
JSON - 9 KB
Information
Show More
Created
Today, 2:21 PM
Modified
Today, 2:21 PM
Last opened
--
Tags
Add Tags...
Cancel
Open

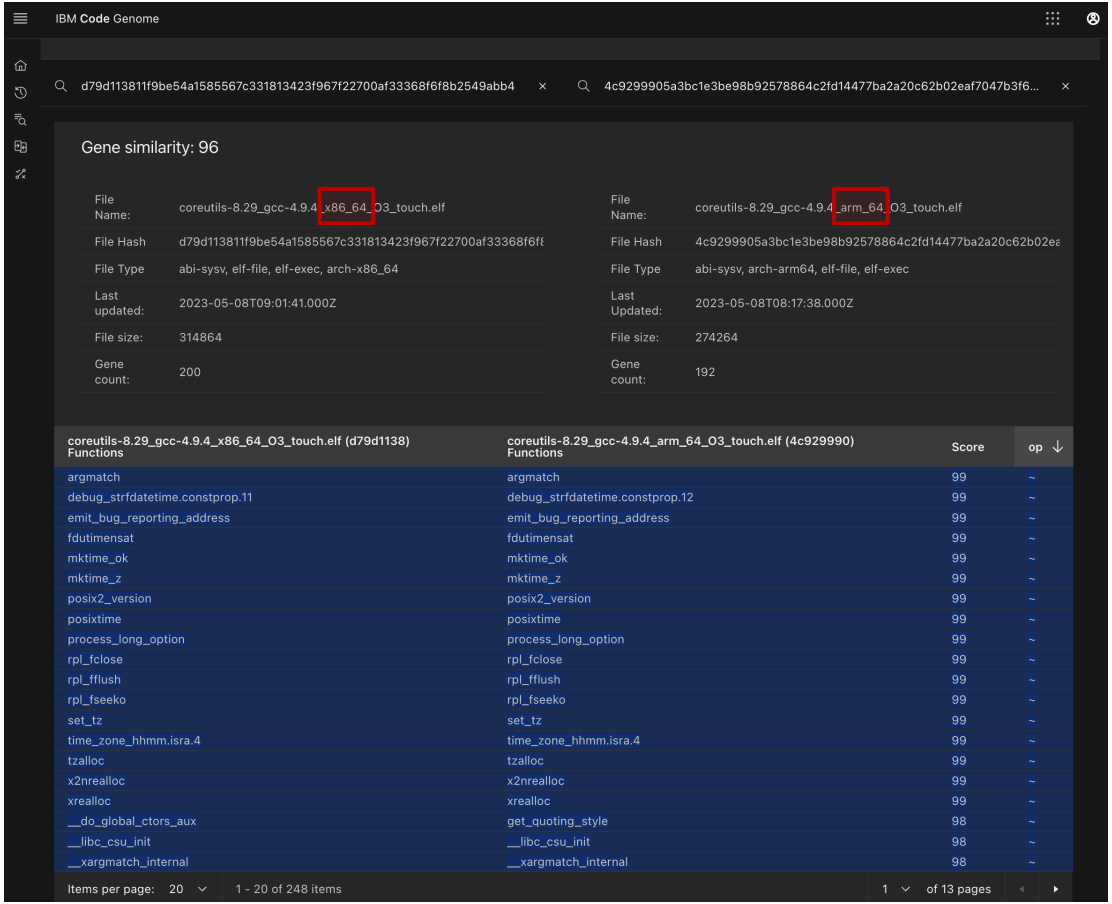

Home / Projects / Unknown App
Unknown App
0
0
0
0
0
View Details
Overview
Components 8
Services 0
Dependency Graph 0
Audit Vulnerabilities 0
Exploit Predictions 0
Policy Violations 0
+ Add Component
− Remove Component
Upload BOM
Download BOM
Search
Component
Version
Group
Internal
License
Risk Score
Vulnerabilities
iputils 20160308 BSD and GPLv2+ 10 1
clockdiff 20160308 BSD and GPLv2+ 0 0
tracepath6 20160308 BSD and GPLv2+ 0 0
arping 20160308 BSD and GPLv2+ 0 0
rdisc 20160308 BSD and GPLv2+ 0 0
tracepath 20160308 BSD and GPLv2+ 0 0
ping 20160308 BSD and GPLv2+ 0 0
ifenslave 20160308 BSD and GPLv2+ 0 0
Showing 1 to 8 of 8 rows
Disclaimer: artificial vulnerability is added for a demo purpose
Dependency-Track v4.7.0

Demo 2: Reproducible build



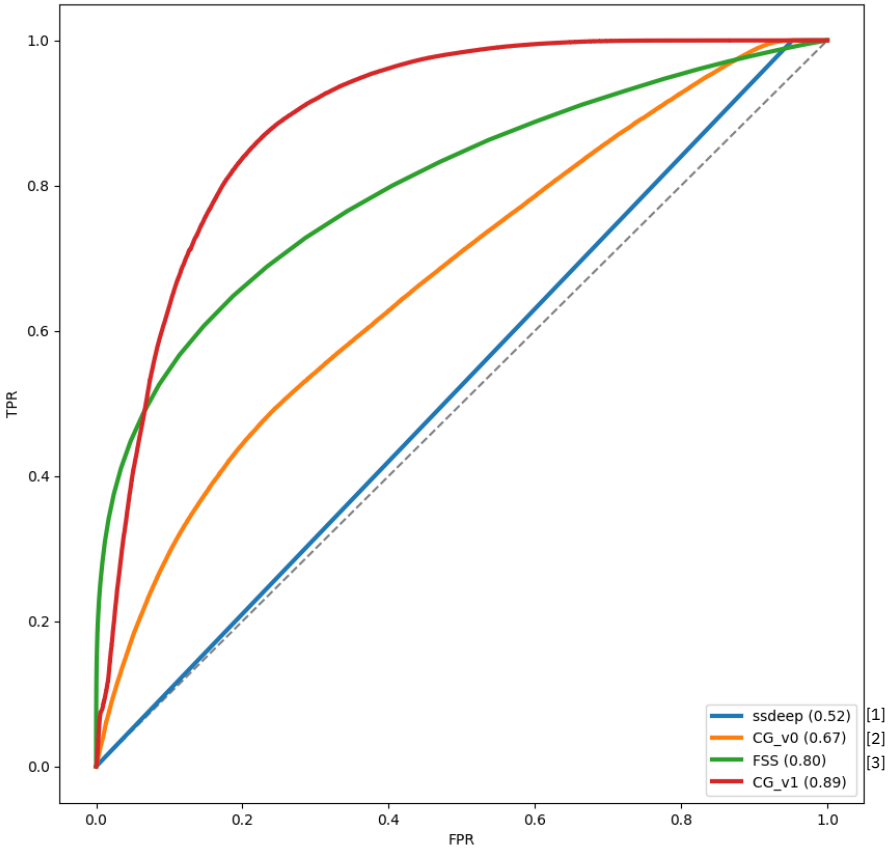
- Reproducible build is to ensure same source code results in the same binary for verification.
- Same source code compiled in different environments and setups generates the same gene.

Demo 2: Reproducible build



- Reproducible build is to ensure same source code results in the same binary for verification.
- Same source code compiled in different environments and setups generates the same gene.

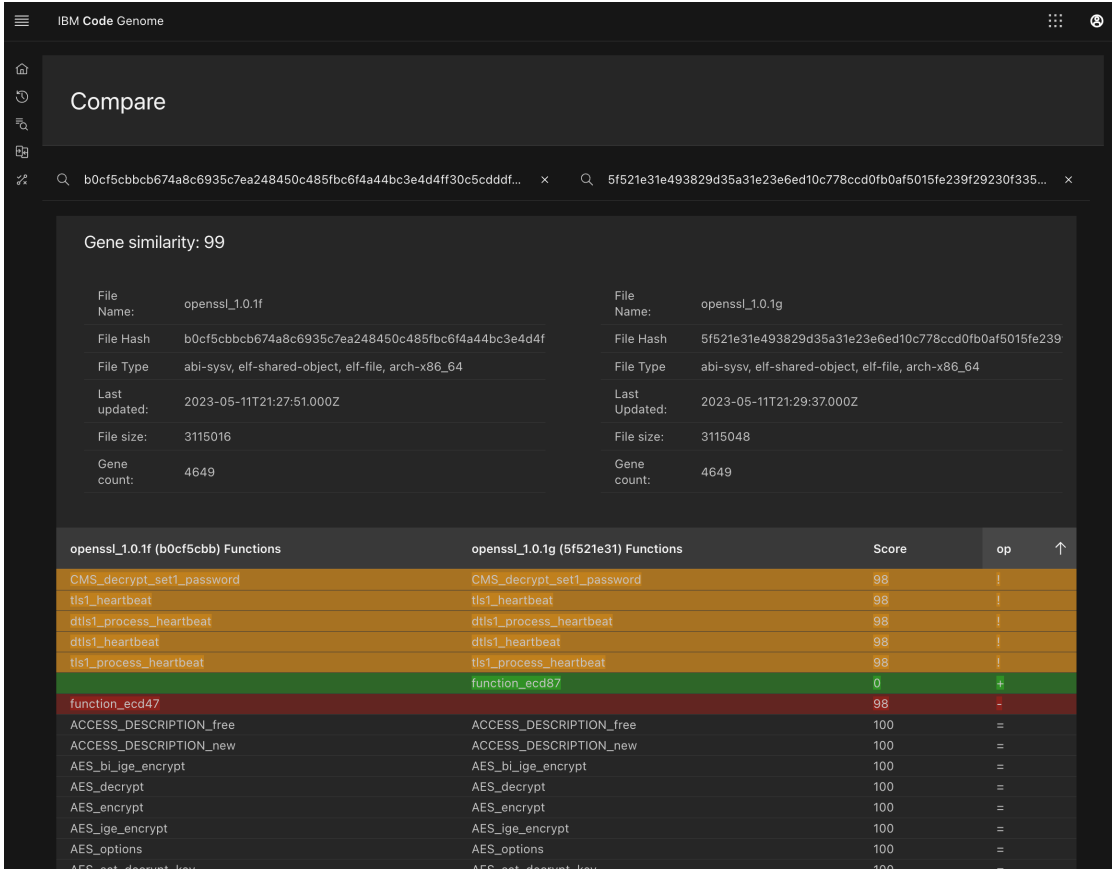
Gene Quality Evaluation



- coreutils
- 105 programs compiled in 180 combinations [4]
  - 5 arch: `x86, x86_64, arm, aarch64, mips`
  - 9 compilers: `clang` (4 versions), `gcc` (5 versions)
  - 4 compilation optimizations: `O0, O1, O2, O3`
- 250k positive and 250k negative pairs

[1] ssdeep. *https://github.com/ssdeep-project/ssdeep*
[2] SigMal: A Static Signal Processing Based Malware Triage, *ACSAC 2013*
[3] FunctionSimSearch. *https://github.com/googleprojectzero/functionsimsearch*
[4] BinKit. *https://github.com/SoftSec-KAIST/BinKit*

Demo 3: Binary Diff

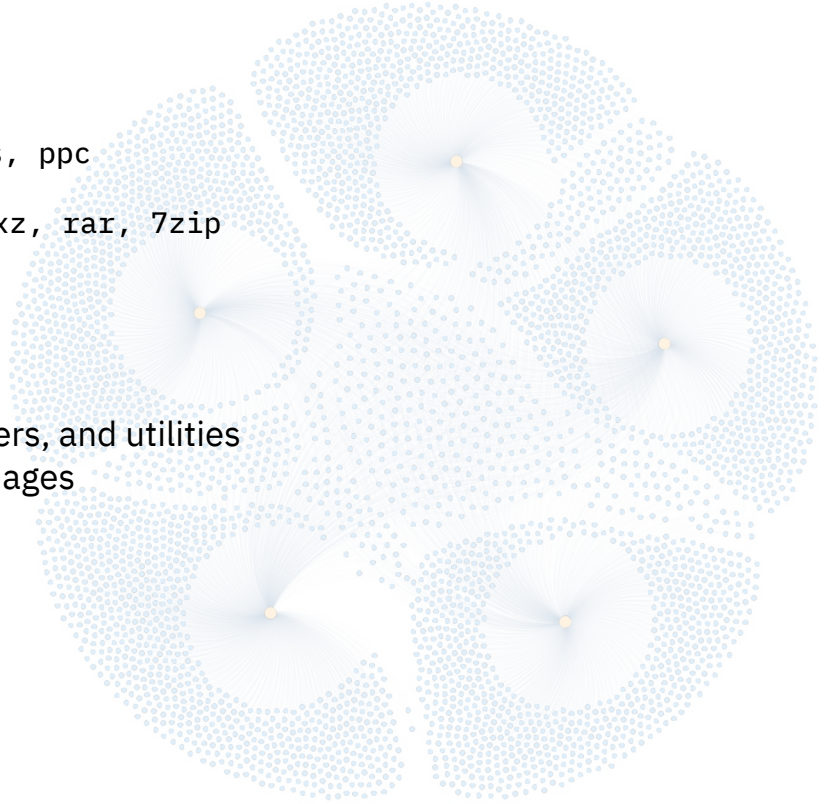


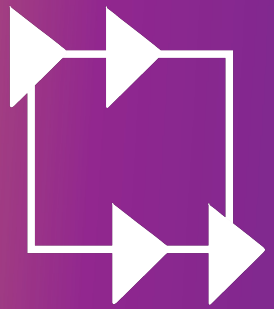
- Heartbleed vulnerability
- Highlight the differences between two binaries at the function gene level
- Examine the changes between two versions of projects

Function differences were due to the vulnerability patch and the source line numbers in the error message.

Status and Roadmap

- Status
  - Currently supported
    - Binaries: `ELF, PE, Mach-O`
    - Architectures: `x86, x86_64, arm, aarch64, mips, ppc`
    - Packages: `deb, rpm, ipa`
    - Archives: `ar, cpio, tar, bzip2, gzip, zstd, xz, rar, 7zip`
  - Cloud-native process engine and knowledge graph
  - Third-generation genome computation
  - Large scale data collection
- Next steps
  - Integration into build environments, package managers, and utilities
  - Improving support for Golang binaries and Docker images
- Planning limited-service launch
  - Improving scalability and decreasing cost
- Request
  - Welcome feedback, support, and collaboration
  - Insights and use cases





SUPPLYCHAIN SECURITYCON

@

THE LINUX FOUNDATION

OPEN SOURCE SUMMIT

NORTH AMERICA

THE LINUX FOUNDATION