



SUPPLYCHAIN
SECURITYCON

@



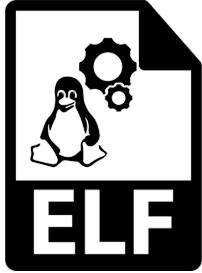
OPEN SOURCE SUMMIT
NORTH AMERICA

THE LINUX FOUNDATION

Code Genome: Fingerprinting Code to Build Trustworthy SBOMs

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





- \$ foo install bar
 - Signed with a certificate.
 - Lists dependencies.
- Do you trust it?

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.

“You can’t trust code that you did not totally create yourself.”

—Ken Thompson

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. UNIX¹ 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.

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.

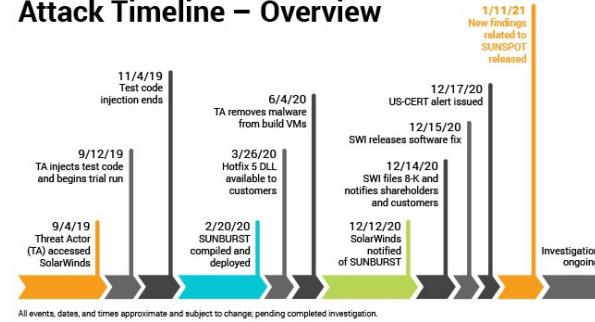
More precisely stated, the problem is to write a source program that, when compiled and executed, will produce as output an exact copy of its source. If you have never done this, I urge you to try it on your own. The discovery of how to do it is a revelation that far surpasses any benefit obtained by 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 C³ 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

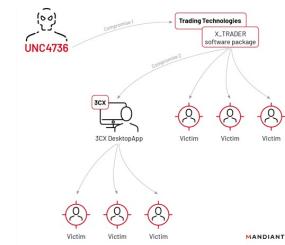
Attack Timeline – Overview



KrebsOnSecurity
In-depth security news and investigation

Learn more about Mandiant's initial findings by reading our latest blog post.
► The attack is attributed to a cluster named UNC4736 with a North Korean nexus
► Attacker infected targeted 3CX systems with TAXHAUL malware
► MacOS backdoor: SIMPLESEA

bit.ly/3mnoSG6



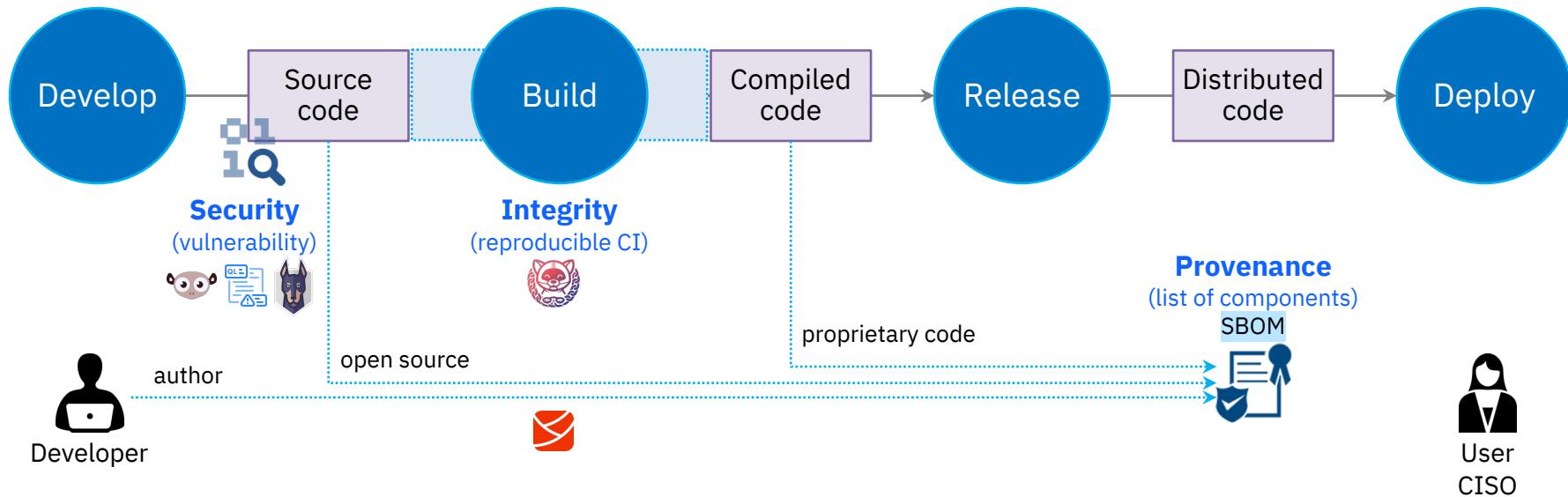
Codecov breach impacted 'hundreds' of customer networks: report

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

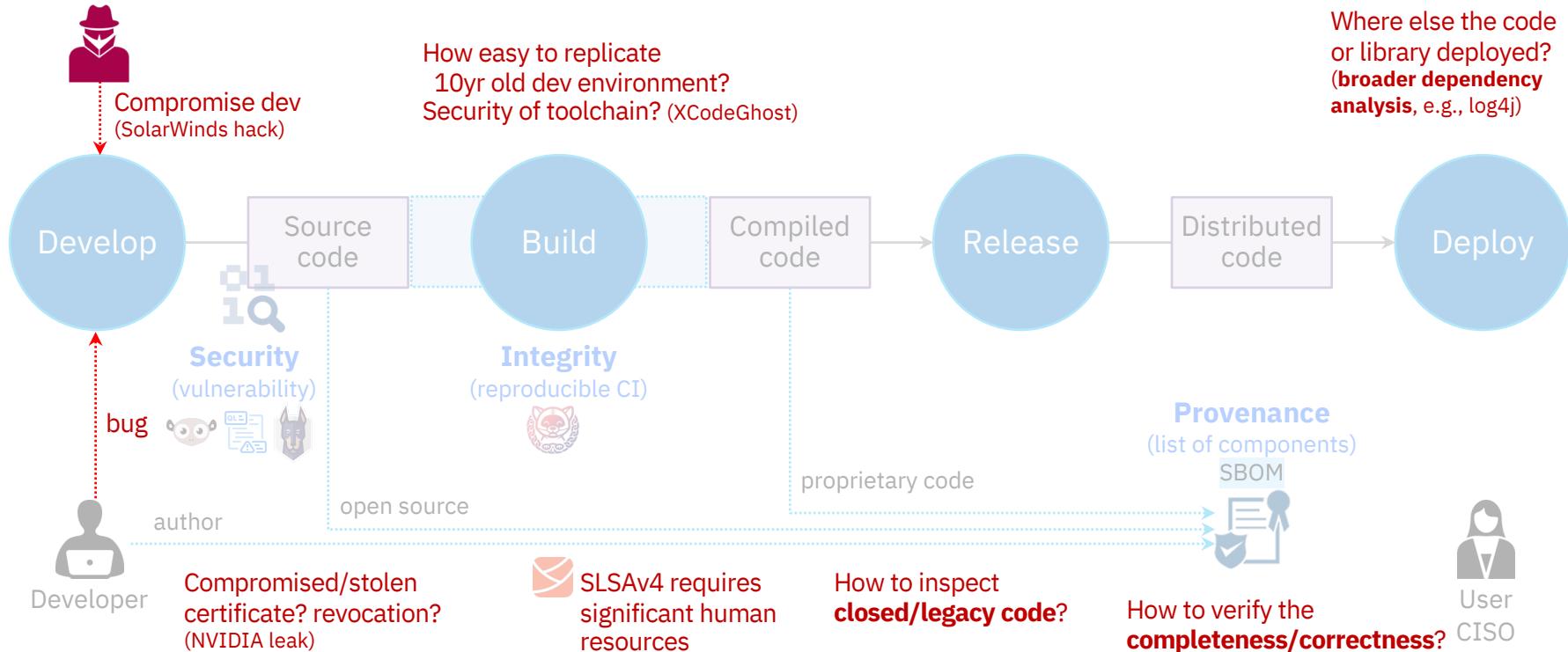


CISA-FBI Guidance for MSPs and their Customers Affected by the Kaseya VSA Supply-Chain Ransomware Attack

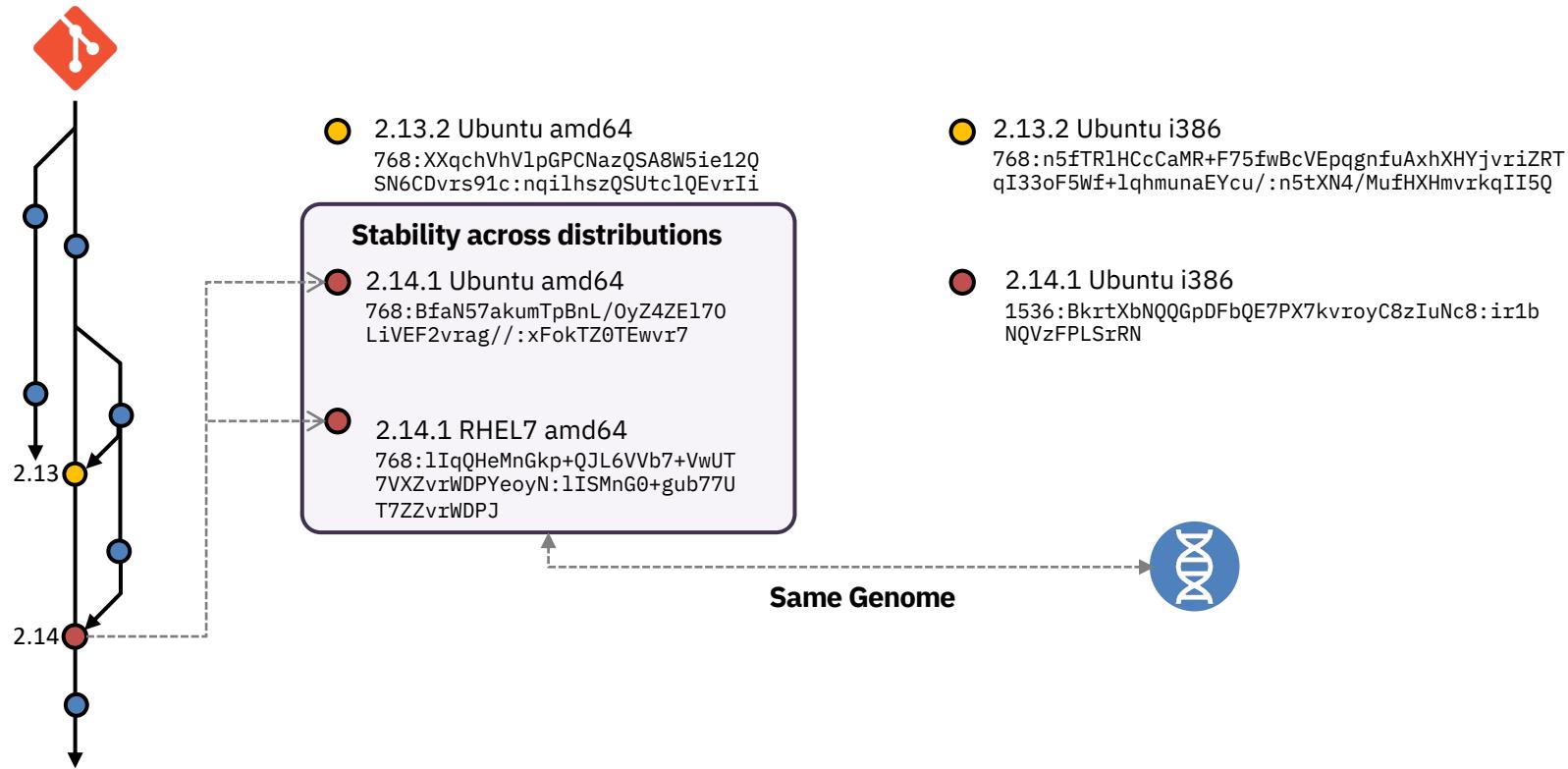
Supply Chain Security: Industry approach to protecting CI/CD pipelines



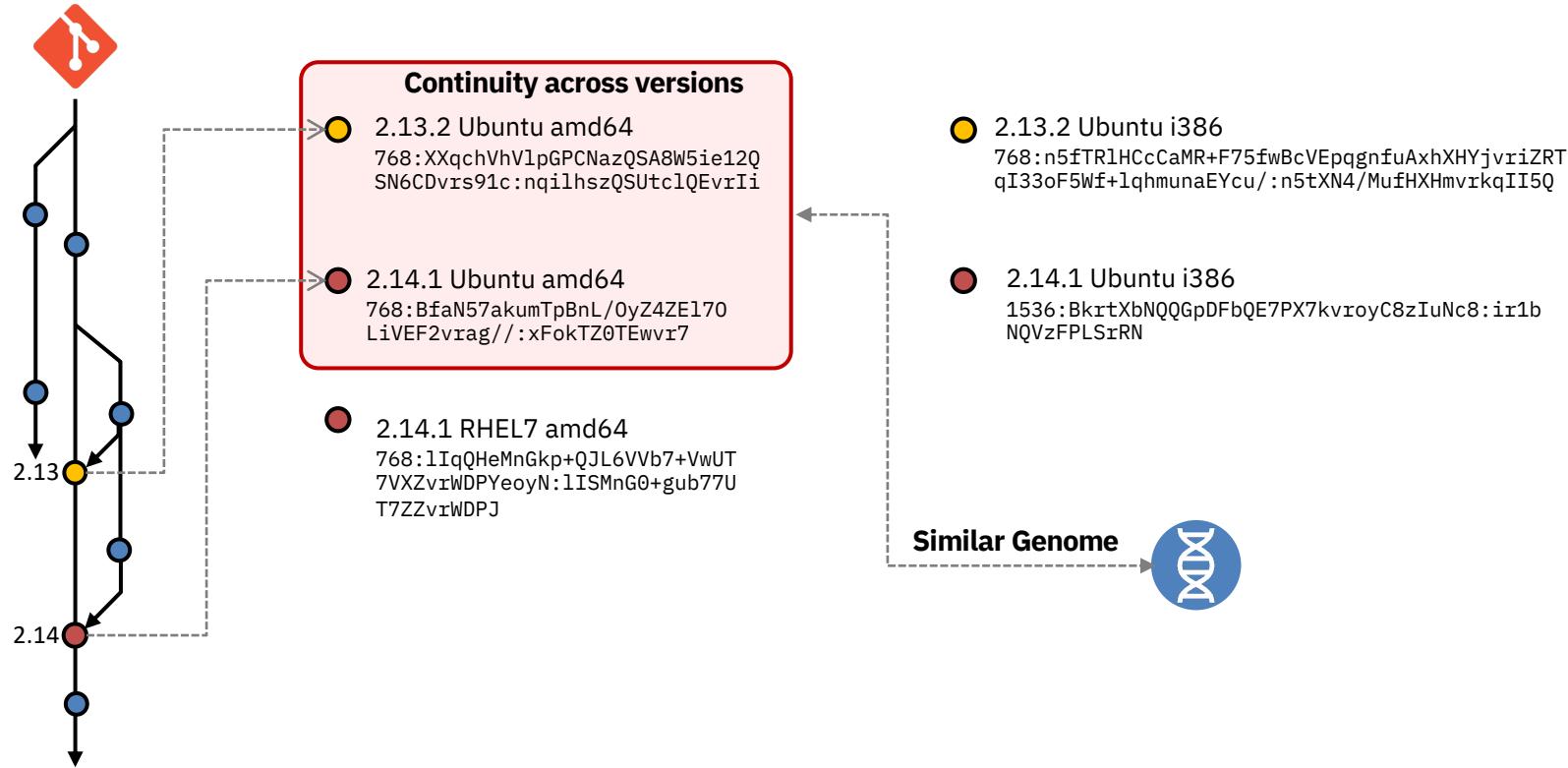
Supply Chain Security: Open security issues and residual risks



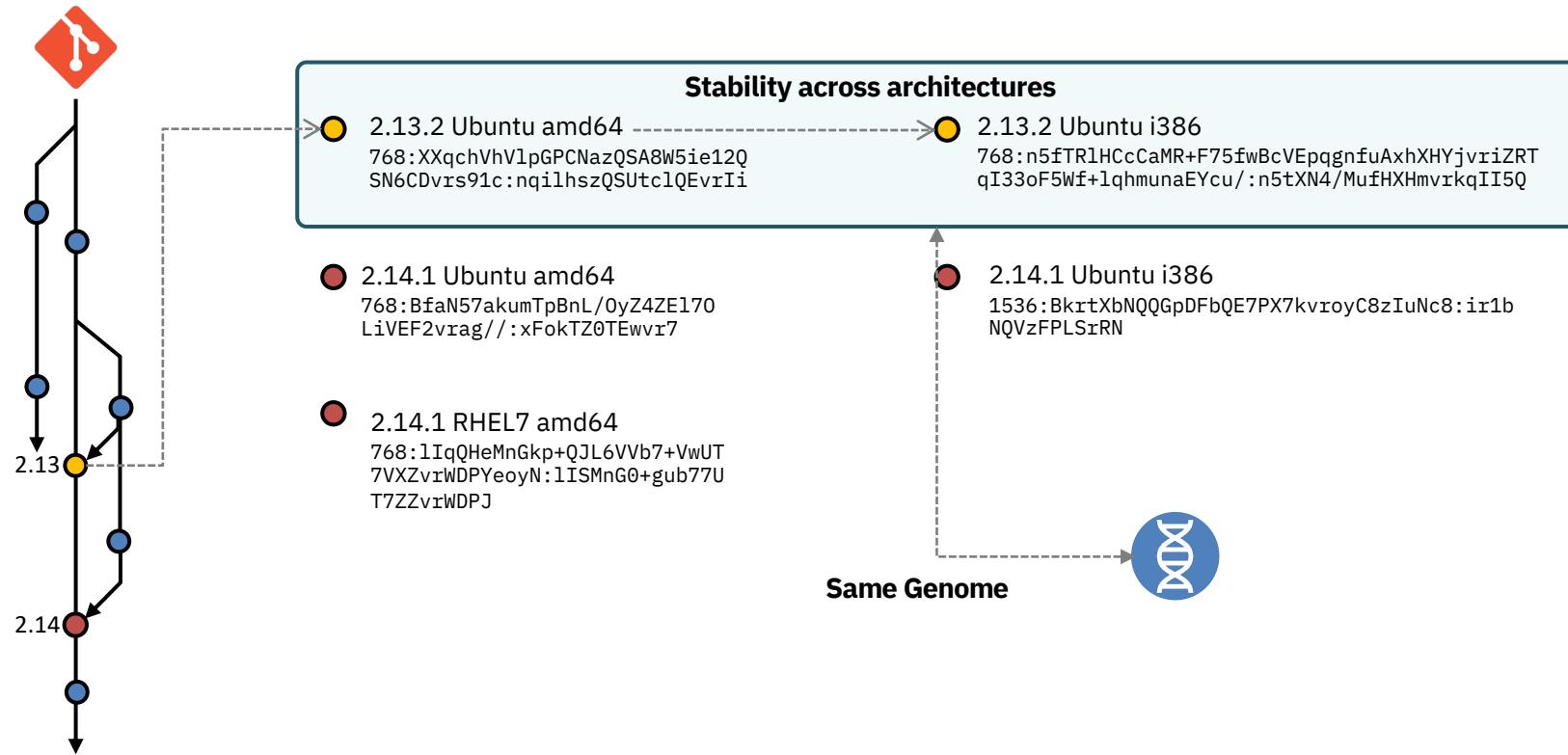
Software Fingerprints for Software Assurance



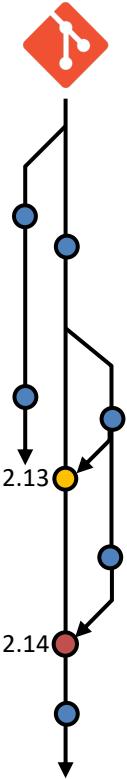
Software Fingerprints for Software Assurance



Software Fingerprints for Software Assurance

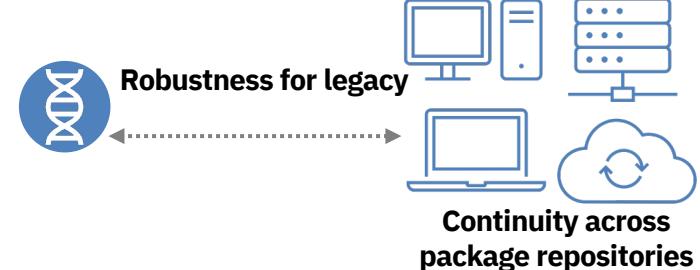


Software Fingerprints for Software Assurance



- 2.13.2 Ubuntu amd64
768:XXqchVhVlpGPCNazQSA8W5ie12Q
SN6CDvrs91c:nqilhszQSUtc1QEvrIi
- 2.14.1 Ubuntu amd64
768:BfaN57akumTpBnL/0yZ4ZE170
LiVEF2vrag//xFokTZ0TEwvr7
- 2.14.1 RHEL7 amd64
768:lIiqQHeMnGkp+QJL6VVb7+VwUT
7VXvrxWDPYeoN:lISMnG0+gub77UT
T7ZZvrxWDPJ

- 2.13.2 Ubuntu i386
768:n5fTR1HCcCaMR+F75fwBcVEpqgnfuAxhXHYjvriZRT
qI33oF5Wf+lqhmunaeYcu/:n5tXN4/MufHXHmvirkqII5Q
- 2.14.1 Ubuntu i386
1536:BkrxtXbNQQGpDFbQE7PX7kvroyC8zIuNc8:ir1b
NQVzFPLSiRN



Timeline

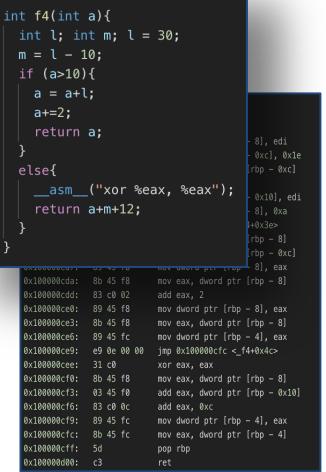
- 
- 2011 BitShred: Feature Hashing Malware for Scalable Triage and Semantic Analysis
 - 2012 ReDeBug: Finding Unpatched Code Clones in Entire OS Distributions
 - 2013 Sigmal: A static signal processing-based malware triage
Towards automatic software lineage inference
 - 2015 Malgene: Automatic extraction 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 **Code Genome: Open Source Summit North America**
(today)

Code Genome: Semantically meaningful fingerprint

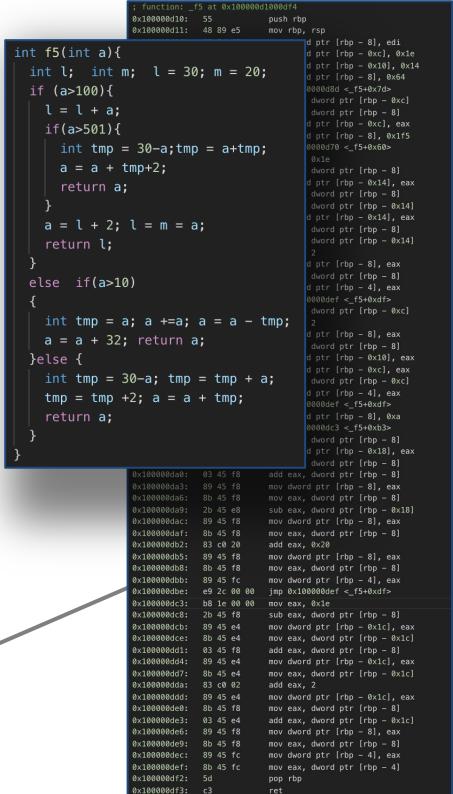
```
int f3(int a){  
    int local = a;  
    a = local;  
    local = 30;  
    a = a + local;  
    a+=2;  
    __asm__ ("xor %eax, %eax");  
    __asm__ ("xor %eax, %eax");  
    __asm__ ("xor %eax, %eax");  
    return a;  
}  
  
int f2(int a){  
    int local=31;  
    local +=1;  
    local = a + local;  
    return local;  
}
```



```
int f4(int a){  
    int l; int m; l = 30;  
    m = l - 10;  
    if (a>10){  
        a = a+l;  
        a+=2;  
        return a;  
    }  
    else{  
        __asm__("xor %eax, %eax");  
        return a+m+12;  
    }  
}
```



```
; Function: _f5 at 0x100000d10000014  
0x100000d10: 55 push rbp  
0x100000d11: 48 89 e5 mov rbp, rsp  
  
int f5(int a){  
    int l; int m; l = 30; m = 20;  
    if (a>100){  
        l = l + a;  
        if(a>501){  
            int tmp = 30-a;tmp = a+tmp;  
            a = a + tmp+2;  
            return a;  
        }  
        a = l + 2; l = m = a;  
        return l;  
    }  
    else if(a>10)  
    {  
        int tmp = a; a +=a; a = a - tmp;  
        a = a + 32; return a;  
    }  
    else {  
        int tmp = 30-a; tmp = tmp + a;  
        tmp = tmp +2; a = a + tmp;  
        return a;  
    }  
  
0x100000da: 03 45 f8 add eax, dword ptr [rbp - 8]  
0x100000dd: 89 45 f8 mov dword ptr [rbp - 8], eax  
0x100000e0: 83 c0 02 add eax, 2  
0x100000e3: 89 45 f8 mov dword ptr [rbp - 8], eax  
0x100000e6: 89 45 fc mov dword ptr [rbp - 4], eax  
0x100000e9: e9 00 00 00 jmp 0x100000fc <_f5+0x4>  
0x100000ee: 31 c0 xor eax, eax  
0x100000f0: 89 45 f8 mov eax, dword ptr [rbp - 8]  
0x100000f3: 03 45 f0 add eax, dword ptr [rbp - 0x10]  
0x100000f6: 89 45 fc mov dword ptr [rbp - 4], eax  
0x100000f9: 89 45 fc mov dword ptr [rbp - 4], eax  
0x100000fc: 89 45 fc mov eax, dword ptr [rbp - 4]  
0x100000ff: 5d pop rbp  
0x10000000: c3 ret  
  
0x100000da0: 03 45 f8 add eax, dword ptr [rbp - 8]  
0x100000da3: 89 45 f8 mov dword ptr [rbp - 8], eax  
0x100000da6: 89 45 f8 mov eax, dword ptr [rbp - 8]  
0x100000da9: 2b 45 e8 sub eax, dword ptr [rbp - 0x18]  
0x100000db2: 89 45 f8 mov eax, dword ptr [rbp - 8]  
0x100000db5: 89 45 f8 mov eax, dword ptr [rbp - 8]  
0x100000db8: 89 45 fc mov dword ptr [rbp - 4], eax  
0x100000dbb: 89 45 fc mov dword ptr [rbp - 4], eax  
0x100000dbd: 09 2c 00 00 jmp 0x100000dc <_f5+0xd>  
0x100000dc0: 31 c0 xor eax, eax  
0x100000dc3: 2b 45 f8 sub eax, dword ptr [rbp - 8]  
0x100000dc6: 89 45 e4 mov dword ptr [rbp - 0x1c], eax  
0x100000dc9: 89 45 e4 mov eax, dword ptr [rbp - 0x1c]  
0x100000dcf: 03 45 f8 add eax, dword ptr [rbp - 8]  
0x100000dd2: 89 45 e4 mov dword ptr [rbp - 0x1c], eax  
0x100000dd5: 89 45 e4 mov eax, dword ptr [rbp - 0x1c]  
0x100000dd8: 89 45 f8 add eax, dword ptr [rbp - 8]  
0x100000ddc: 89 45 f8 mov eax, dword ptr [rbp - 8]  
0x100000dec: 89 45 fc mov dword ptr [rbp - 4], eax  
0x100000def: 89 45 fc mov eax, dword ptr [rbp - 4]  
0x100000d02: 5d pop rbp  
0x100000d03: c3 ret
```



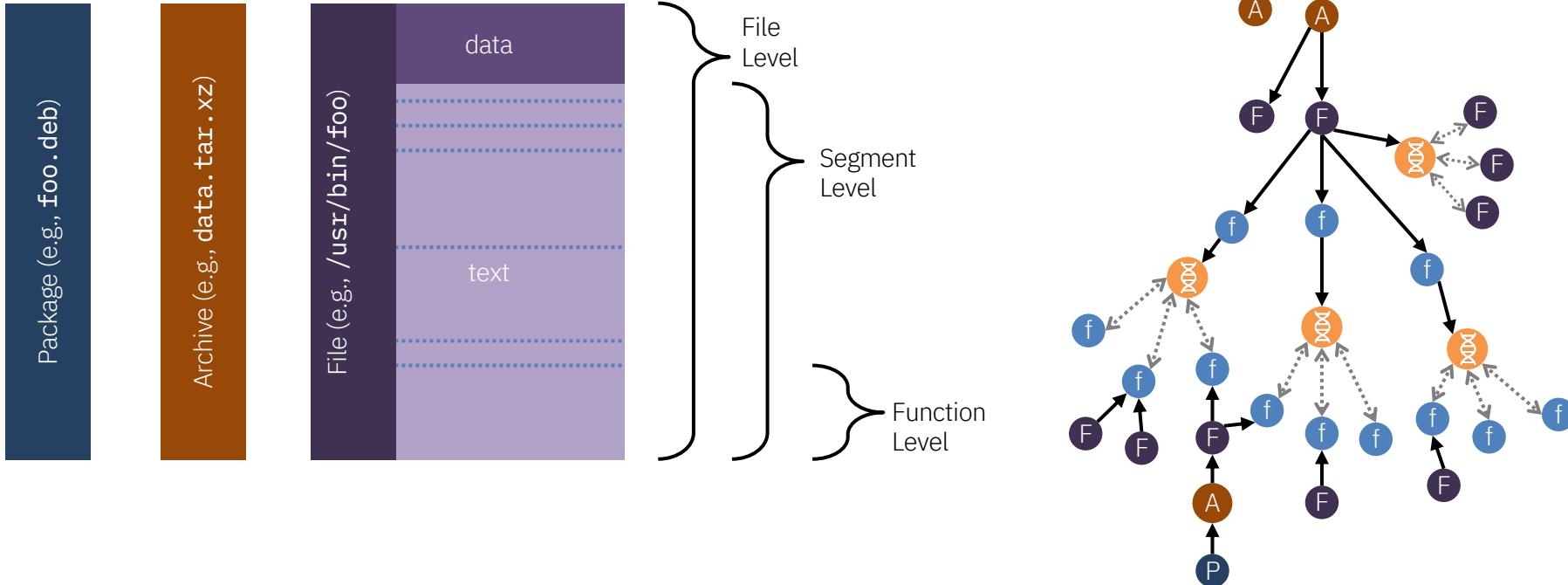
- Across multiple *architectures* (x86, ARM, ...)
- Across multiple *compilers* (gcc, clang, ...)
- Across multiple *optimization levels*
- Handling *obfuscation*

Key Idea: Code Genome construction



Genome can be constructed from [closed-source/legacy code](#) where source code is not easily available.

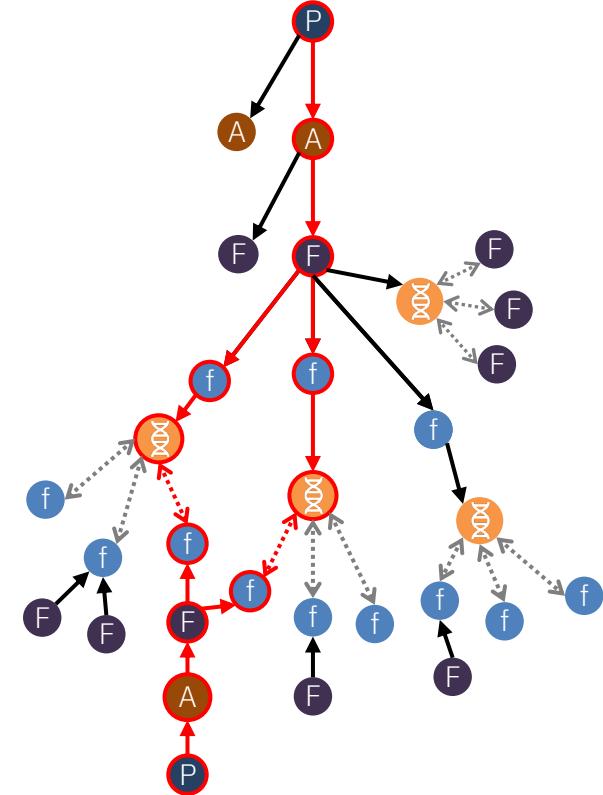
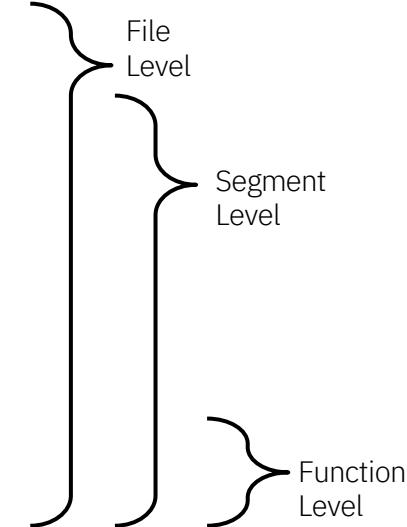
Granularity



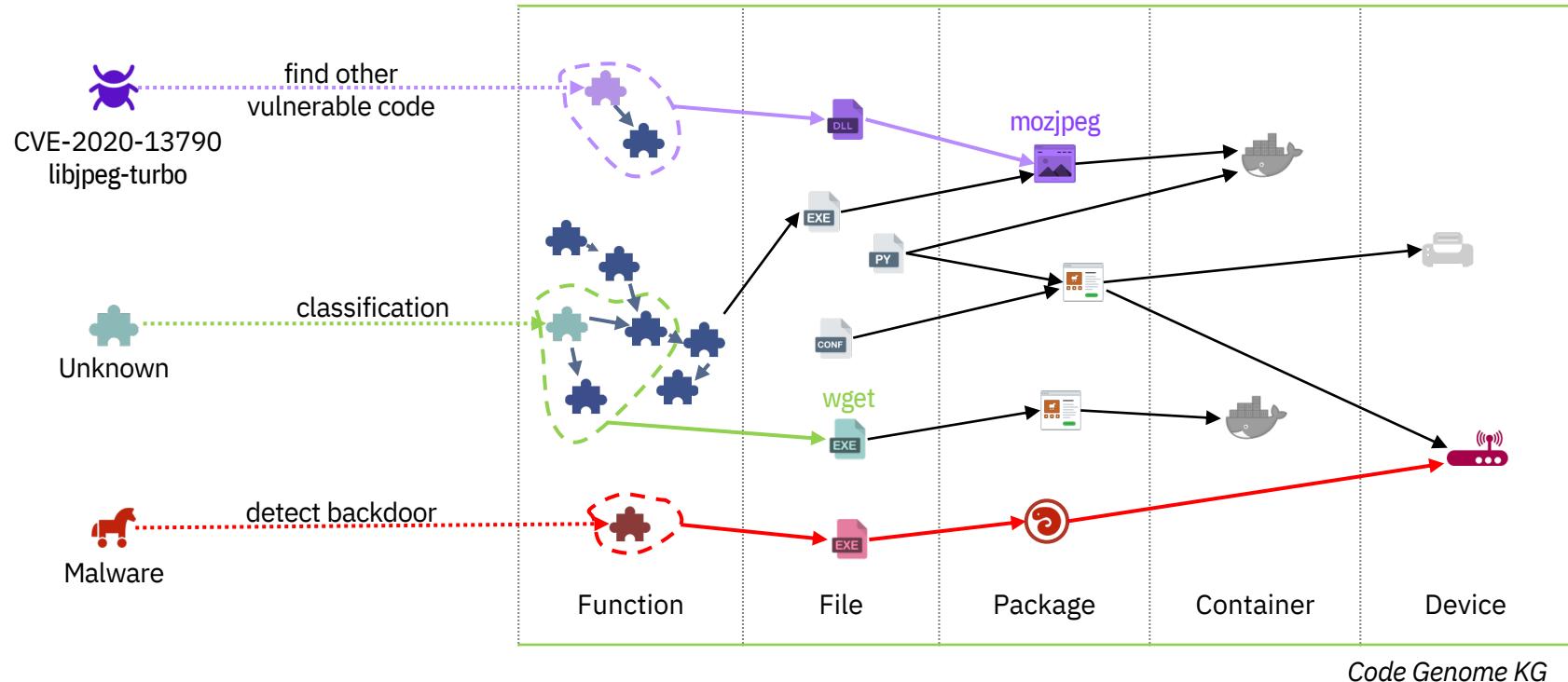
Granularity

Package (e.g., `foo.deb`)

Archive (e.g., `/usr/bin/foo`)



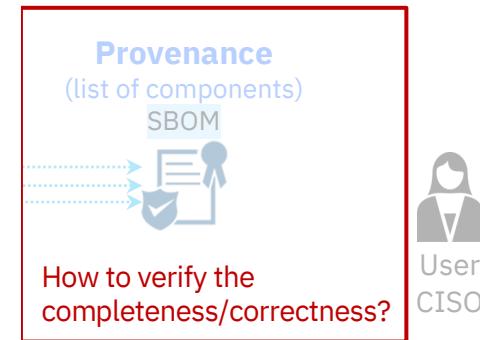
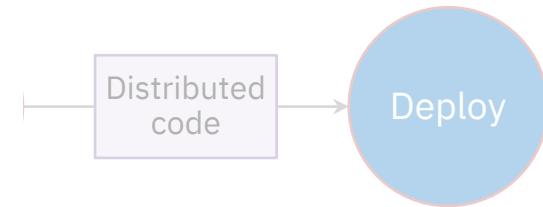
Knowledge Graph: Code Genome and Use Cases



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

Where else the code or library deployed?
(broader dependency analysis, e.g., log4j)



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
```



```
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

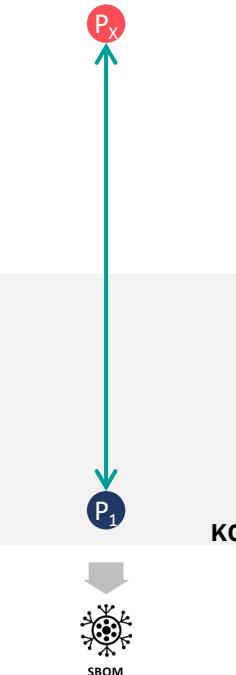
SBOM Type	Definition	Data Description	Benefits
Design	SBOM of intended, planned software project or product with included components (some of which may not yet exist) for a new software artifact.	Typically derived from a design specification, RFP, or initial concept.	
Source	SBOM created directly from 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.	Analyzed <ul style="list-style-type: none">- 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	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.	Deployed <ul style="list-style-type: none">- 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

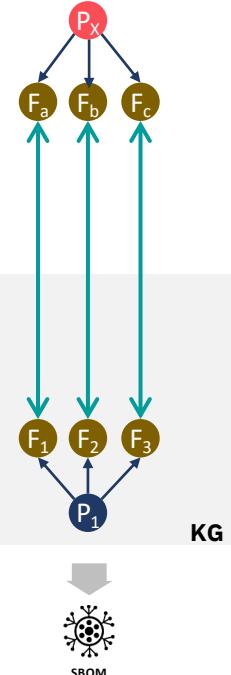
Case #0
Metadata



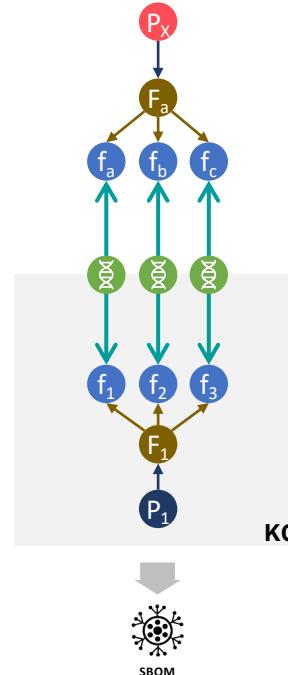
Case #1
Equivalent Package



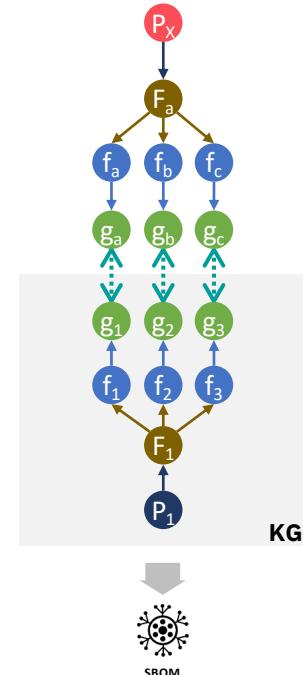
Case #2
Equivalent File



Case #3
Equivalent Function



Case #4
Similar Gene





Demo

Demo 1: SBOM generation for an unknown rpm package

Custom rpm package

```
unknown2a
├── etc
│   └── sysconfig
│       └── rdisc
└── usr
    ├── bin
    │   ├── ping
    │   │   └── ping6 -> ping
    │   ├── tracepath
    │   └── tracepath6
    ├── lib
    │   └── systemd
    │       └── system
    │           └── rdisc.service
    ├── sbin
    │   ├── arping
    │   ├── clockdiff
    │   ├── ifenslave
    │   ├── ping6 -> ../bin/ping
    │   ├── rdisc
    │   ├── tracepath -> ../bin/tracepath
    │   └── tracepath6 -> ../bin/tracepath6
    └── share
        ├── doc
        │   └── iputils-20160308
        │       ├── README.bonding
        │       └── RELNOTES
        └── man
            └── man8
                ├── arping.8.gz
                ├── clockdiff.8.gz
                ├── ifenslave.8.gz
                ├── ping.8.gz
                ├── ping6.8.gz -> ping.8.gz
                ├── rdisc.8.gz
                └── tracepath.8.gz
                    └── tracepath6.8.gz -> tracepath.8.gz
```



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



Home

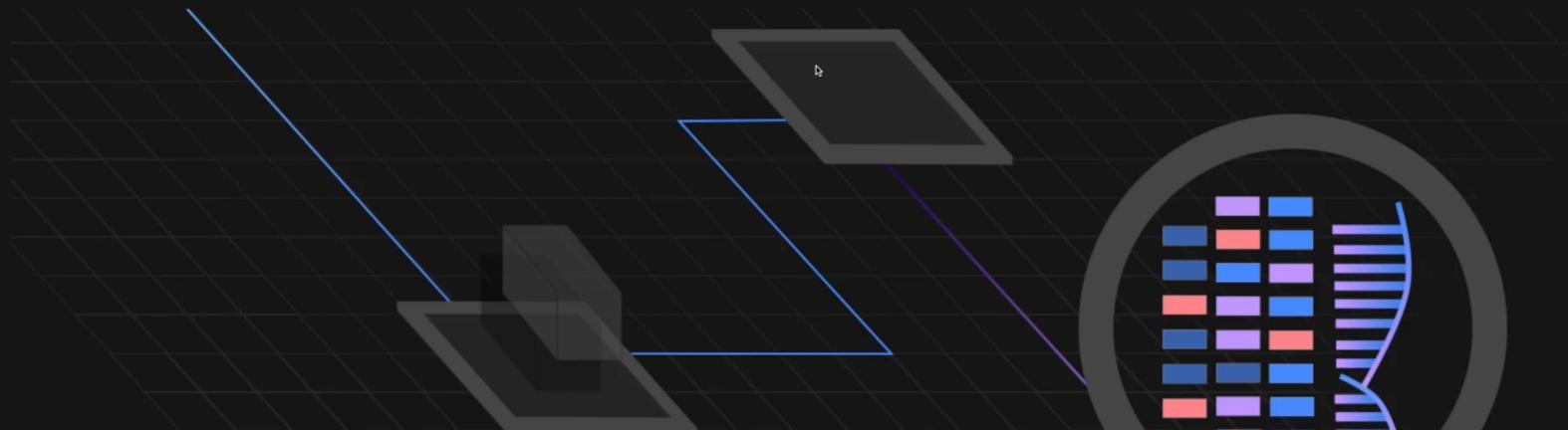
Enter a file hash to begin your search or click the blue 'Add file' push-button to scan a file...



Upload files

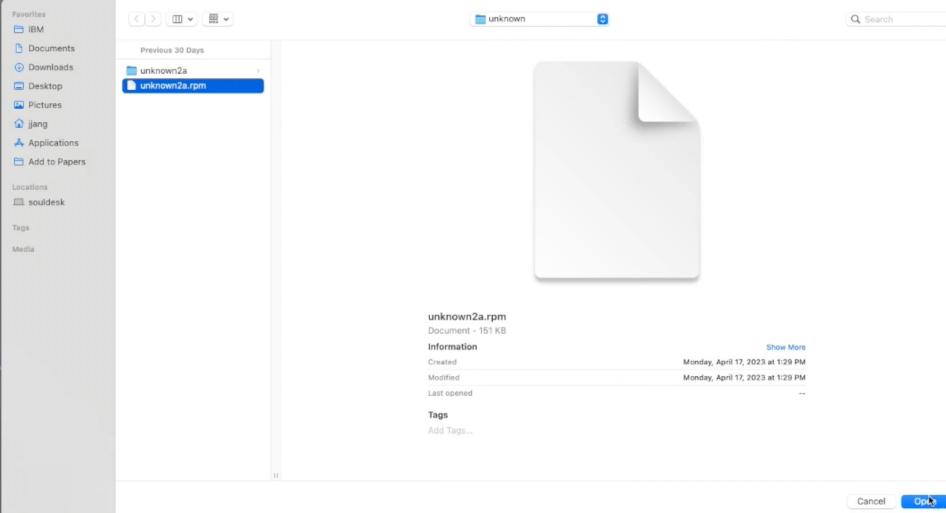
Max file size is 10mb.

Add file



Home

Enter a file hash to begin your search or click the blue 'Add file' push-button to scan a file...





Details SBOM Genome Search



File Name unknown2a.rpm

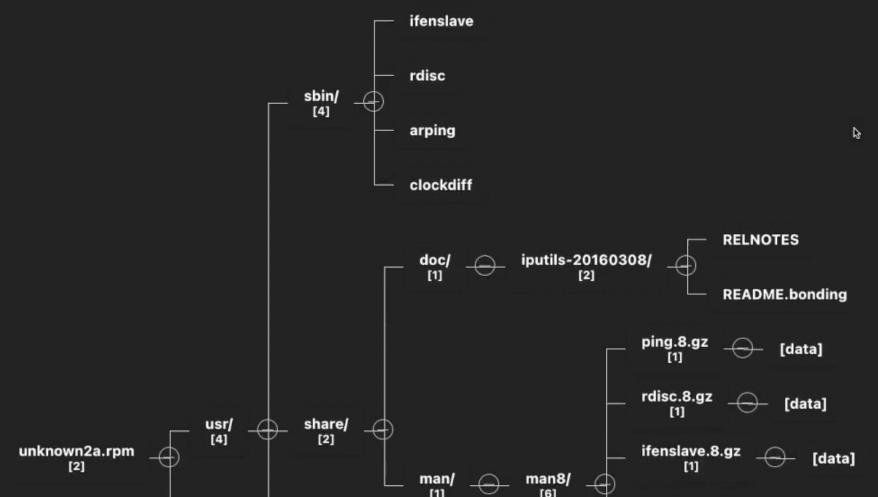
File Hash e8e096ab13b3655702d5360d4b2be3273651e35b2375619d3bef87676d232252

File Type rpm-pkg

File size 151184

File content

Collapse All





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

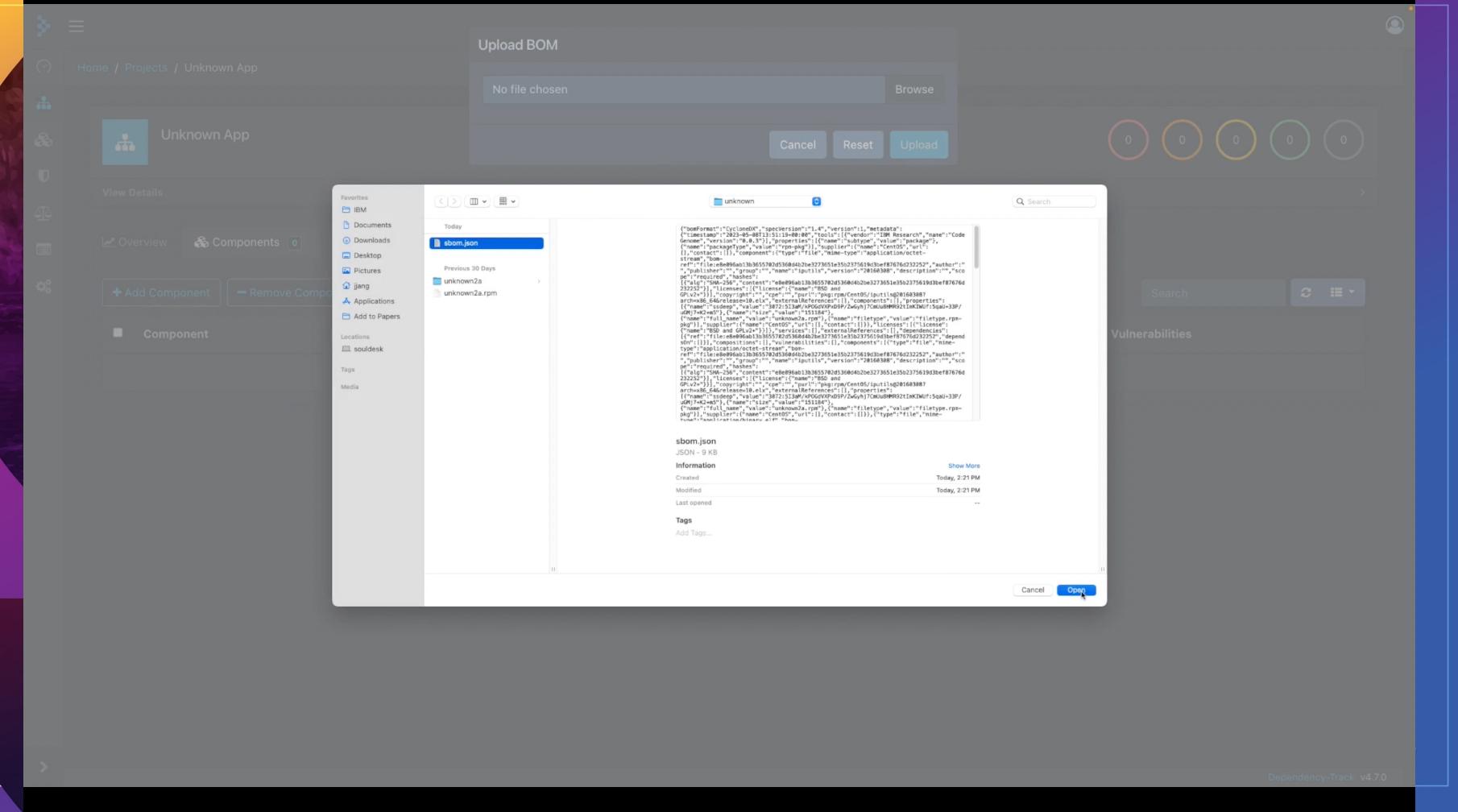
```

▼ "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





≡



Home / Projects / Unknown App



Unknown App



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
iutils	20160308			BSD and GPLv2+	10	<div style="width: 100%; background-color: #ff8080;">1</div>
clockdiff	20160308			BSD and GPLv2+	0	<div style="width: 100%; background-color: #00ffff;">0</div>
tracepath6	20160308			BSD and GPLv2+	0	<div style="width: 100%; background-color: #00ffff;">0</div>
arping	20160308			BSD and GPLv2+	0	<div style="width: 100%; background-color: #00ffff;">0</div>
rdisc	20160308			BSD and GPLv2+	0	<div style="width: 100%; background-color: #00ffff;">0</div>
tracepath	20160308			BSD and GPLv2+	0	<div style="width: 100%; background-color: #00ffff;">0</div>
ping	20160308			BSD and GPLv2+	0	<div style="width: 100%; background-color: #00ffff;">0</div>
ifenslave	20160308			BSD and GPLv2+	0	<div style="width: 100%; background-color: #00ffff;">0</div>

Showing 1 to 8 of 8 rows

Disclaimer: artificial vulnerability is added for a demo purpose

Demo 2: Reproducible build

The screenshot shows the IBM Code Genome interface with a 'Compare' view. At the top, there are two search bars with file hashes: f2fd57ea65d53697f0df0011a296cb253723b079749e2c6624c797a2f0e7a... and 8a2345ba802686e580d80eeb9ddfd45c8e0a971ffd4c3c671cc05822047e.... Below the search bars, the text 'Gene similarity: 100' is displayed.

Two file details tables are shown:

File Name:	binutils-2.30_clang-4.0_x86_64_O2_elfedit.elf	File Name:	binutils-2.30_clang-4.0_x86_64_O3_elfedit.elf
File Hash	f2fd57ea65d53697f0df0011a296cb253723b079749e2c6624c797a2f0e7a...	File Hash	8a2345ba802686e580d80eeb9ddfd45c8e0a971ffd4c3c671cc05822047e...
File Type	abi-sysv, elf-file, elf-exec, arch-x86_64	File Type	abi-sysv, elf-file, elf-exec, arch-x86_64
Last updated:	2023-05-08T19:45:49.000Z	Last Updated:	2023-05-08T19:45:58.000Z
File size:	77144	File size:	77448
Gene count:	89	Gene count:	89

Below the tables, two function tables are shown:

binutils-2.30_clang-4.0_x86_64_O2_elfedit.elf (f2fd57ea)	binutils-2.30_clang-4.0_x86_64_O3_elfedit.elf (8a2345ba)	Score	op ↓
_start	_start	99	~
byte_get_64	byte_get_64	99	~
error	error	98	~
expandarg	expandarg	99	~
function_400f30	function_400f30	98	~
function_400f40	function_400f40	98	~
function_400f60	function_400f60	99	~
function_400fa0	function_400fa0	99	~
function_400fc0	function_400fc0	99	~
function_400ff0	function_400ff0	99	~
function_401050	function_401050	98	~
function_401060	function_401060	98	~
function_401080	function_401080	98	~
function_4010a0	function_4010a0	98	~
function_4010b0	function_4010b0	98	~
function_4010c0	function_4010c0	99	~
function_401100	function_401100	98	~

- 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

IBM Code Genome

Gene similarity: 96

File Name:	coreutils-8.29_gcc-4.9.4_x86_64_O3_touch.elf	File Name:	coreutils-8.29_gcc-4.9.4_arm_64_O3_touch.elf
File Hash	d79d11381ff9be54a1585567c331813423f967f22700af33368f6fb2549abb4	File Hash	4c9299905a3bc1e3be98b92578864c2fd14477ba2a20c62b02eaf7047b3f6...
File Type	abi-sysv, elf-file, elf-exec, arch-x86_64	File Type	abi-sysv, arch-arm64, elf-file, elf-exec
Last updated:	2023-05-08T09:01:41.000Z	Last Updated:	2023-05-08T08:17:38.000Z
File size:	314864	File size:	274264
Gene count:	200	Gene count:	192

coreutils-8.29_gcc-4.9.4_x86_64_O3_touch.elf (d79d1138)

Functions	Score	op
argmatch	99	-
debug_strerror.datetime.constprop.11	99	-
emit_bug_reporting_address	99	-
fdutimensat	99	-
mktimes_ok	99	-
mktimes_z	99	-
posix2_version	99	-
posixtime	99	-
process_long_option	99	-
rpl_fclose	99	-
rpl_fflush	99	-
rpl_fseeko	99	-
set_tz	99	-
time_zone_hhmm.isra.4	99	-
tzalloc	99	-
x2nrealloc	99	-
xrealloc	99	-
_do_global_ctors_aux	98	-
_libc_csu_init	98	-
_xargmatch_internal	98	-

coreutils-8.29_gcc-4.9.4_arm_64_O3_touch.elf (4c929990)

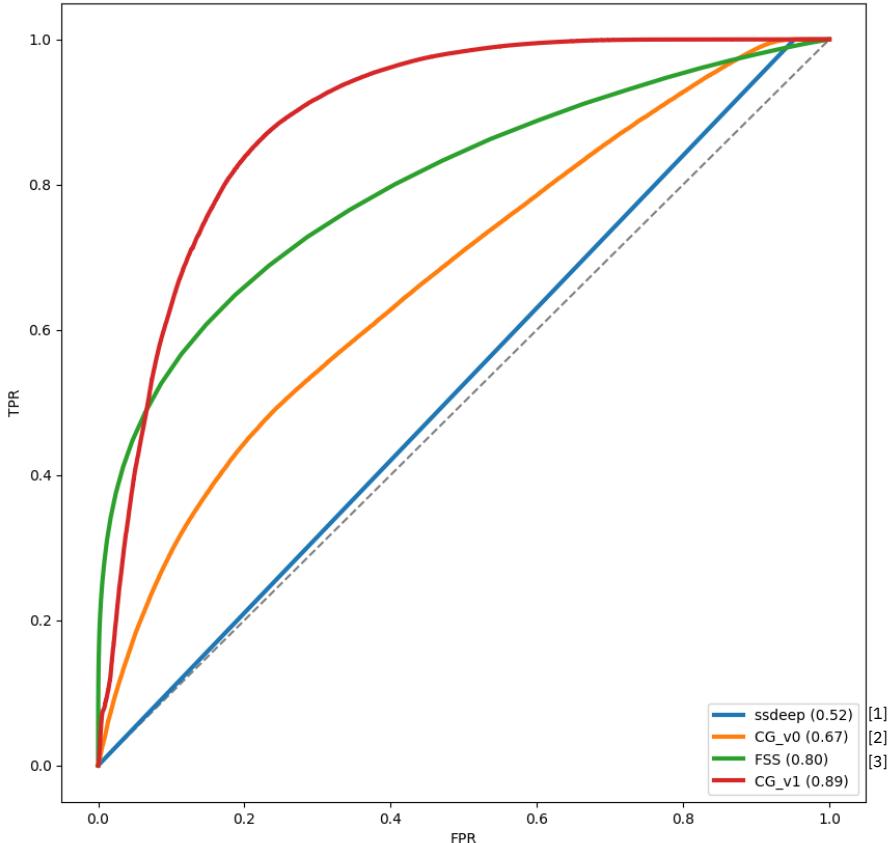
Functions	Score	op
argmatch	99	-
debug_strerror.datetime.constprop.12	99	-
emit_bug_reporting_address	99	-
fdutimensat	99	-
mktimes_ok	99	-
mktimes_z	99	-
posix2_version	99	-
posixtime	99	-
process_long_option	99	-
rpl_fclose	99	-
rpl_fflush	99	-
rpl_fseeko	99	-
set_tz	99	-
time_zone_hhmm.isra.4	99	-
tzalloc	99	-
x2nrealloc	99	-
xrealloc	99	-
get_quoting_style	98	-
_libc_csu_init	98	-
_xargmatch_internal	98	-

Items per page: 20 ▾ 1 - 20 of 248 items

1 ▾ of 13 pages ▶

- 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: 00, 01, 02, 03
- 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

The screenshot shows the IBM Code Genome interface with a 'Compare' view. It displays two binary files: openssl_1.0.1f and openssl_1.0.1g. The file details are as follows:

File Name	openssl_1.0.1f	File Name	openssl_1.0.1g
File Hash	b0cf5cbbcb674a8c6935c7ea248450c485fb6f4a44bc3e4d4ff30c5cdff...	File Hash	5f521e31e493829d35a31e23e6ed10c778cc0fb0af5015fe239f29230f335...
File Type	abi-sysv, elf-shared-object, elf-file, arch-x86_64	File Type	abi-sysv, elf-shared-object, elf-file, arch-x86_64
Last updated:	2023-05-11T21:27:51.000Z	Last Updated:	2023-05-11T21:29:37.000Z
File size:	3115016	File size:	3115048
Gene count:	4649	Gene count:	4649

Gene similarity: 99

Below the file details is a table comparing the functions of both binaries:

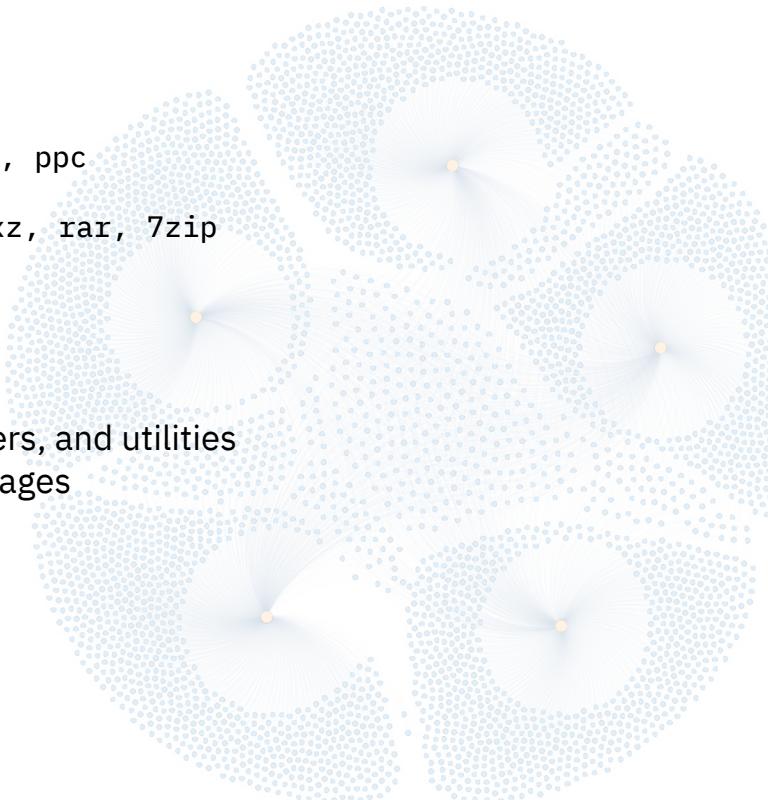
openssl_1.0.1f (b0cf5cbb) Functions	openssl_1.0.1g (5f521e31) Functions	Score	op	↑
CMS_decrypt_set1_password	CMS_decrypt_set1_password	98	!	
tls1_heartbeat	tls1_heartbeat	98	!	
dtls1_process_heartbeat	dtls1_process_heartbeat	98	!	
dtls1_heartbeat	dtls1_heartbeat	98	!	
tls1_process_heartbeat	tls1_process_heartbeat	98	!	
function_ecd87	function_ecd87	0	+	
function_ecd47		98	-	
ACCESS_DESCRIPTION_free	ACCESS_DESCRIPTION_free	100	=	
ACCESS_DESCRIPTION_new	ACCESS_DESCRIPTION_new	100	=	
AES_b1_ige_encrypt	AES_b1_ige_encrypt	100	=	
AES_decrypt	AES_decrypt	100	=	
AES_encrypt	AES_encrypt	100	=	
AES_ige_encrypt	AES_ige_encrypt	100	=	
AES_options	AES_options	100	=	
AES_set_decrypt_kw	AES_set_decrypt_kw	100		

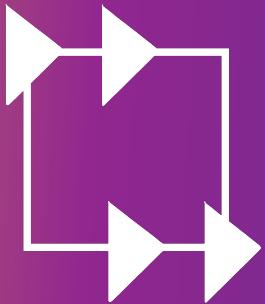
- Heartbleed vulnerability
- Highlight the differences between two binaries at the function gene level
- Examine the changes between two versions of projects

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-0
 - 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

@



OPEN SOURCE SUMMIT
NORTH AMERICA

THE LINUX FOUNDATION