



Uncovering Supply Chain Attack with Code Genome Framework

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

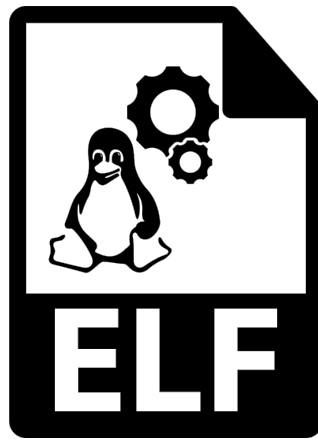


Dhillung Kirat



Jiyong Jang

AI Supply Chain Security Team
IBM **Research**



```
$ 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. 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.

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

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

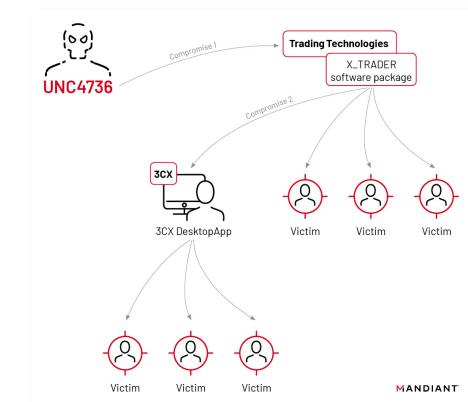
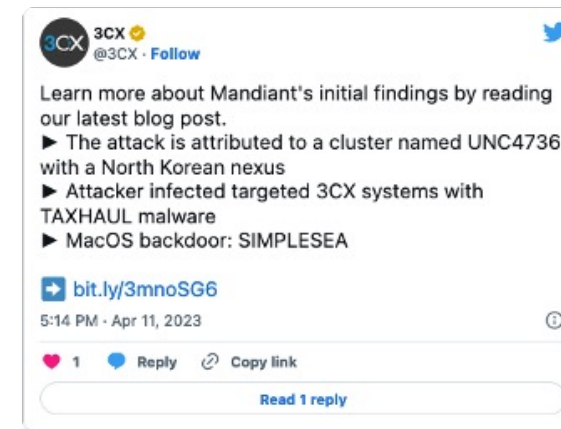
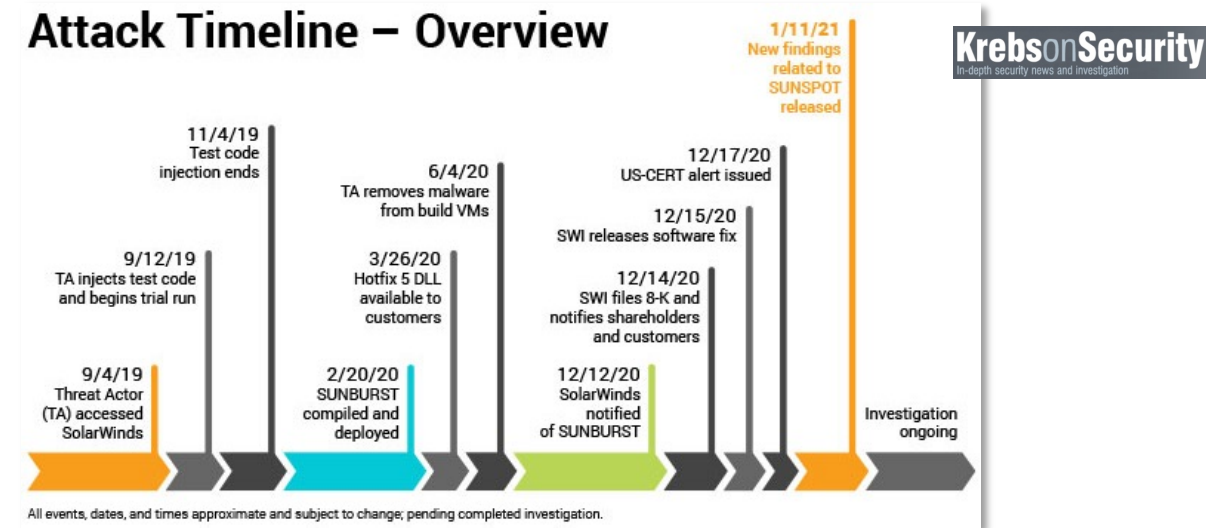
Protestware (Mar 2022)

- 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



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

xz Backdoor

<https://www.openwall.com/lists/oss-security/2024/03/29/4>

Date: Fri, 29 Mar 2024 08:51:26 -0700
From: Andres Freund <andres@...razel.de>
To: oss-security@...ts.openwall.com
Subject: backdoor in upstream xz/liblzma leading to ssh server compromise

Hi,

After observing a few odd symptoms around liblzma (part of the xz package) on Debian sid installations over the last weeks (logins with ssh taking a lot of CPU, valgrind errors) I figured out the answer:

The upstream xz repository and the xz tarballs have been backdoored.

At first I thought this was a compromise of debian's package, but it turns out to be upstream.

== Compromised Release Tarball ==

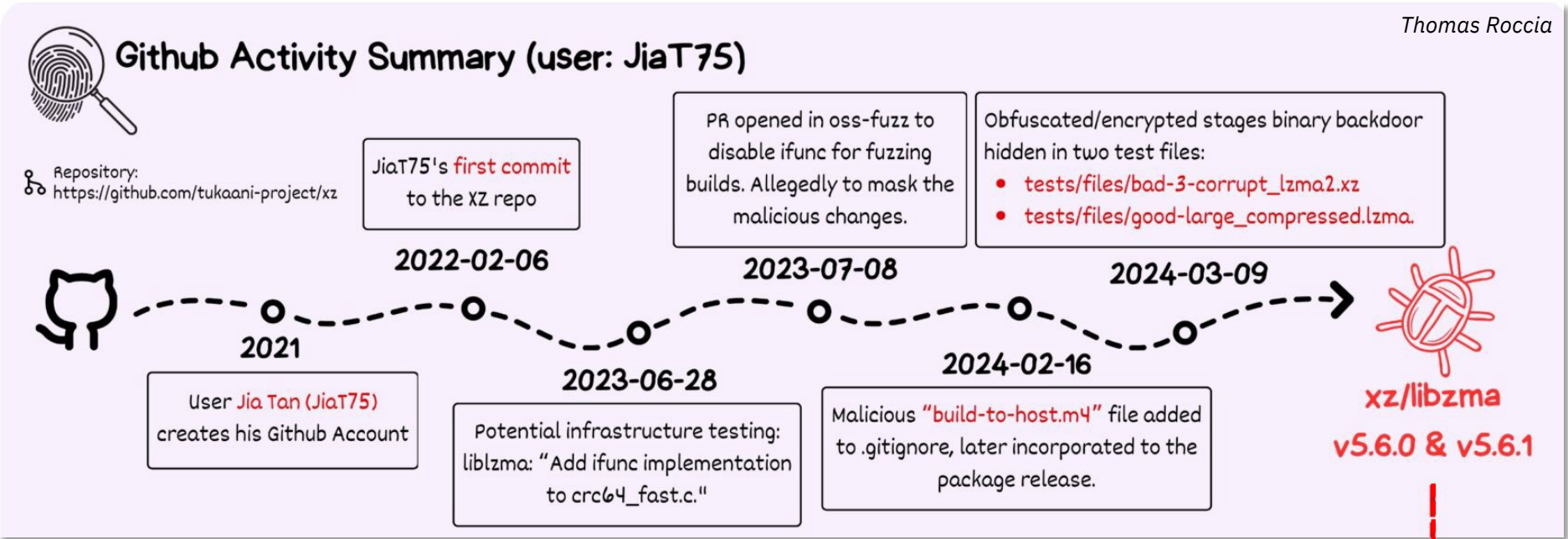
One portion of the backdoor is *solely in the distributed tarballs*. For easier reference, here's a link to debian's import of the tarball, but it is also present in the tarballs for 5.6.0 and 5.6.1:

https://salsa.debian.org/debian/xz-utils/-/blob/debian/unstable/m4/build-to-host.m4?ref_type=heads#L63

That line is **not** in the upstream source of build-to-host, nor is build-to-host used by xz in git. However, it is present in the tarballs released upstream, except for the "source code" links, which I think github generates directly from the repository contents:

<https://github.com/tukaani-project/xz/releases/tag/v5.6.0>
<https://github.com/tukaani-project/xz/releases/tag/v5.6.1>

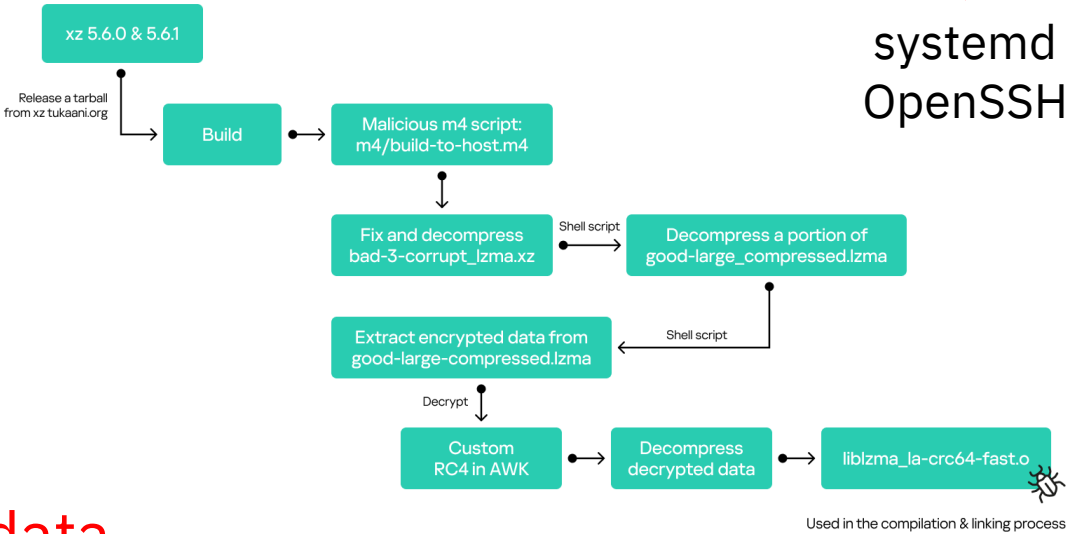
This injects an obfuscated script to be executed at the end of configure. This script is fairly obfuscated and data from "test" .xz files in the repository.



Re: [xz-devel] XZ for Java

Jigar Kumar | Tue, 07 Jun 2022 09:00:18 -0700

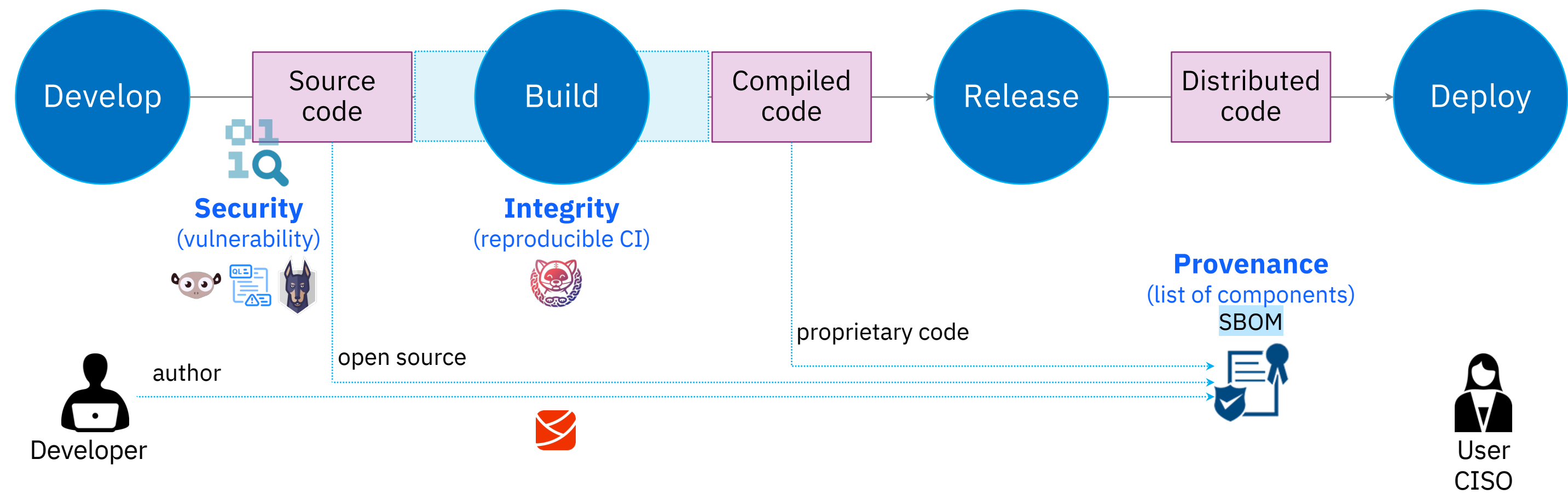
Progress will not happen until there is new maintainer. XZ for C has sparse commit log too. Dennis you are better off waiting until new maintainer happens or fork yourself. Submitting patches here has no purpose these days. The current maintainer lost interest or doesn't care to maintain anymore. It is sad to see for a repo like this.



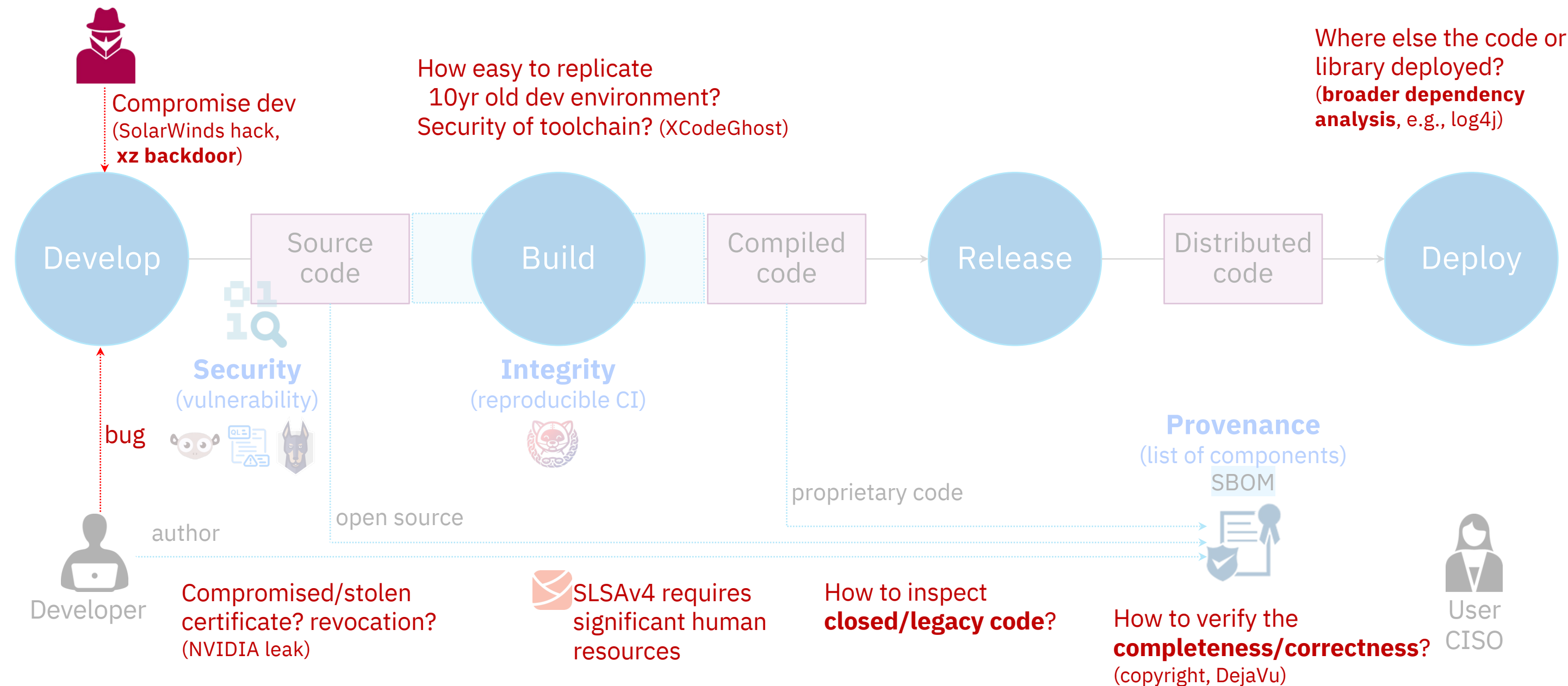
<https://securelist.com/xz-backdoor-story-part-1/112354/>

Semantic gap between compiled code behavior and its metadata

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

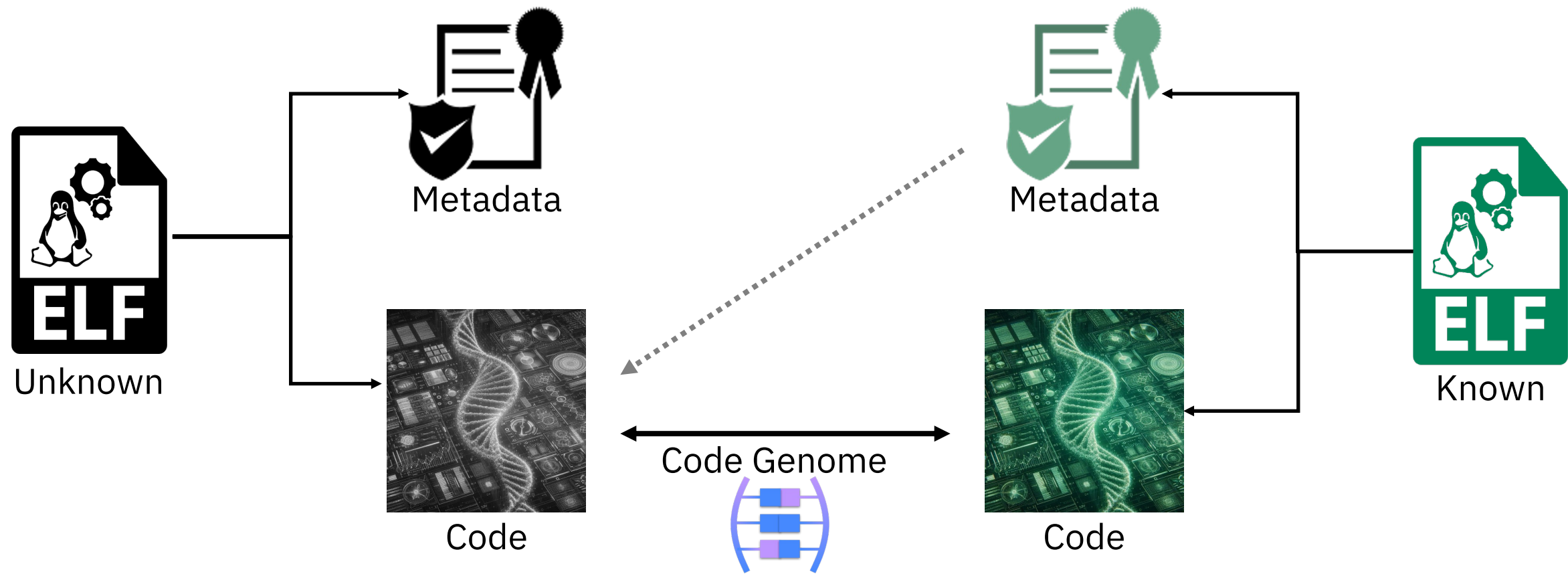


Supply Chain Security: Open security issues and residual risks



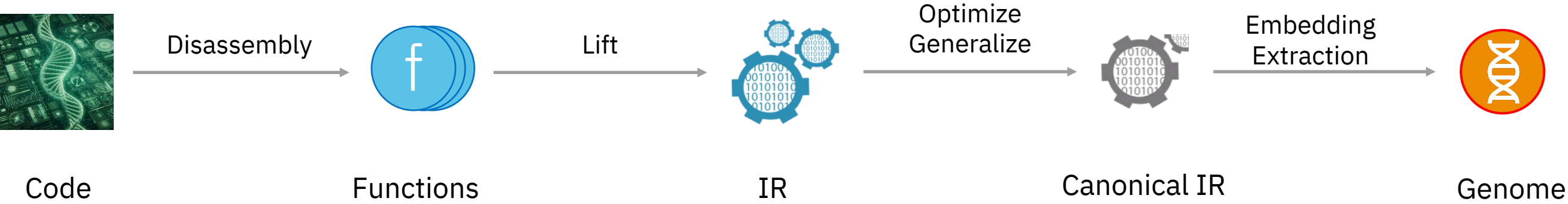
Code **Genome**

The Semantic Gap

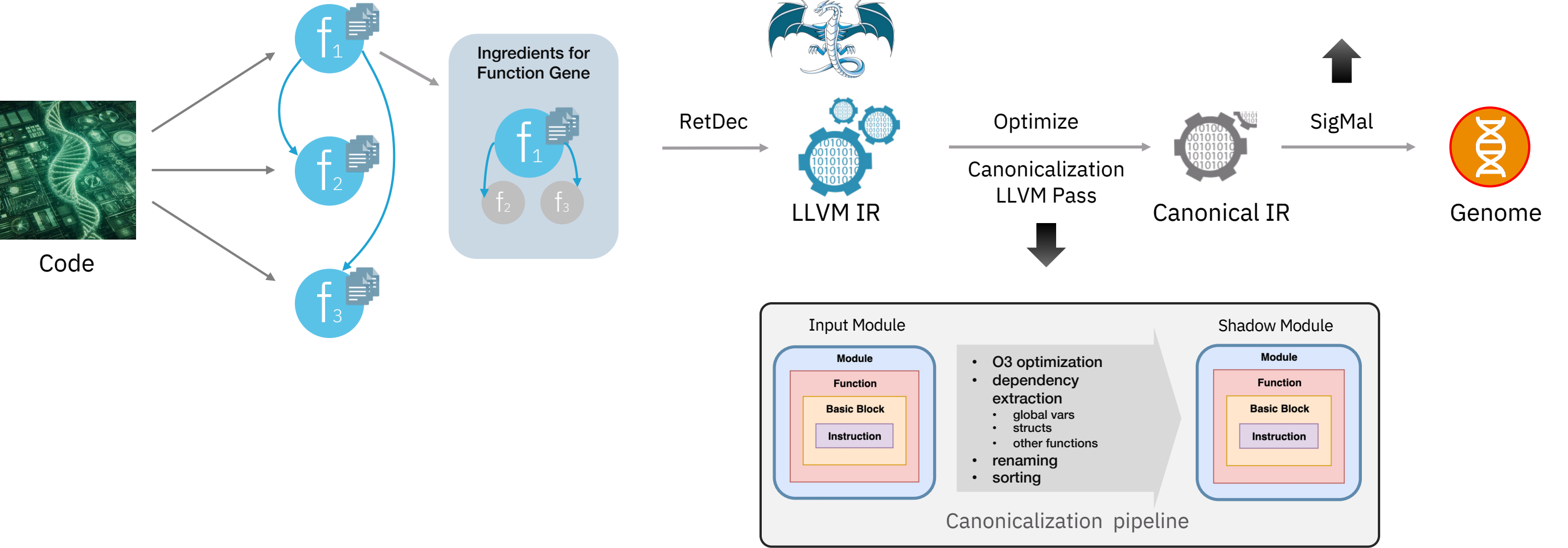


Build chain of trust
by following code equivalency

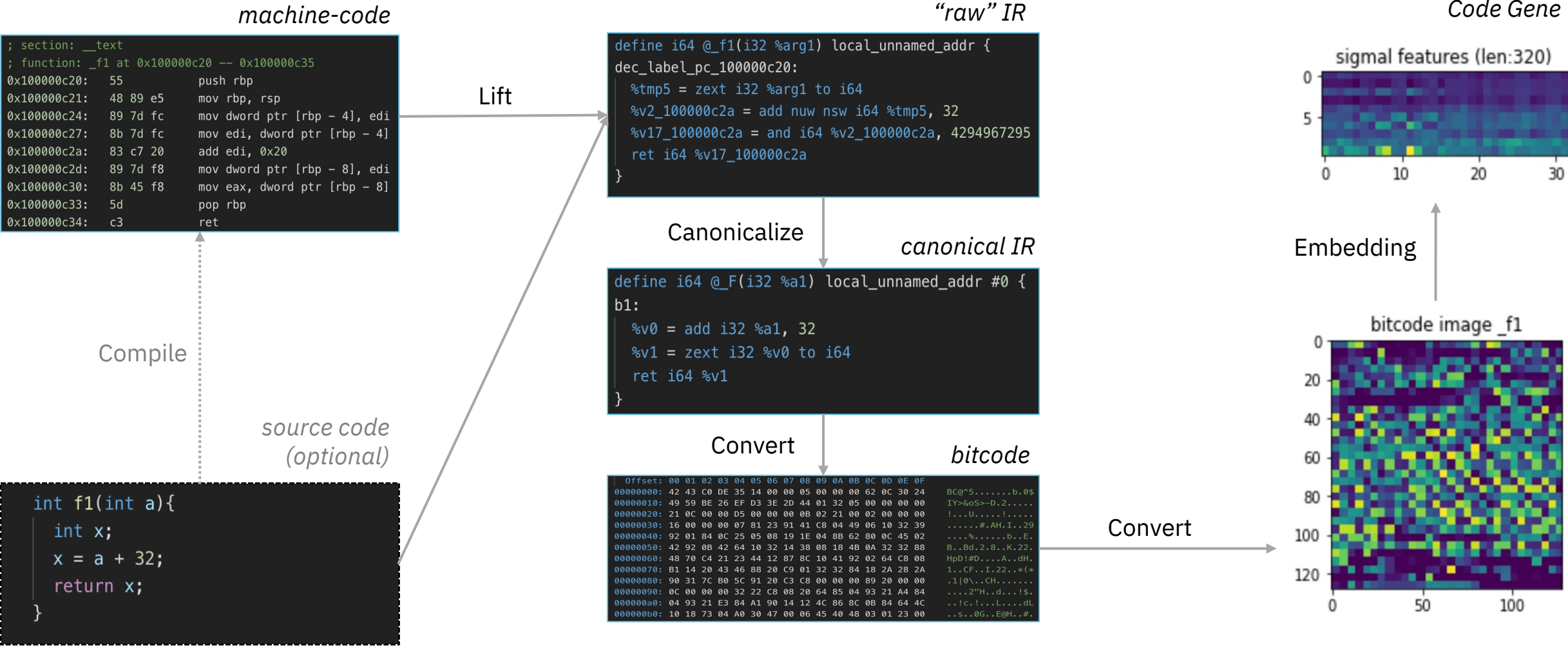
Code Genome Pipeline



Code Genome Pipeline

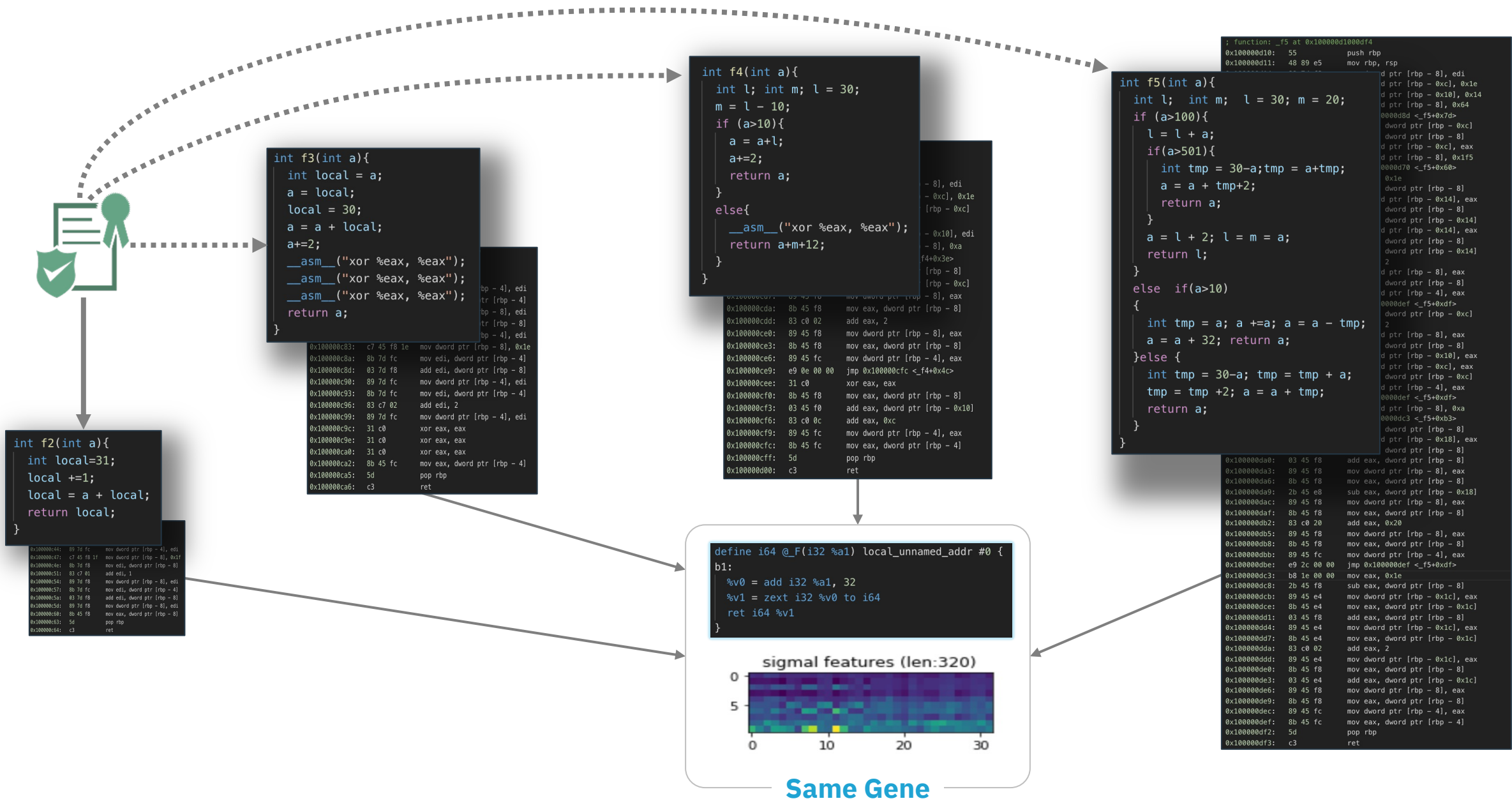


Code Genome Pipeline



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

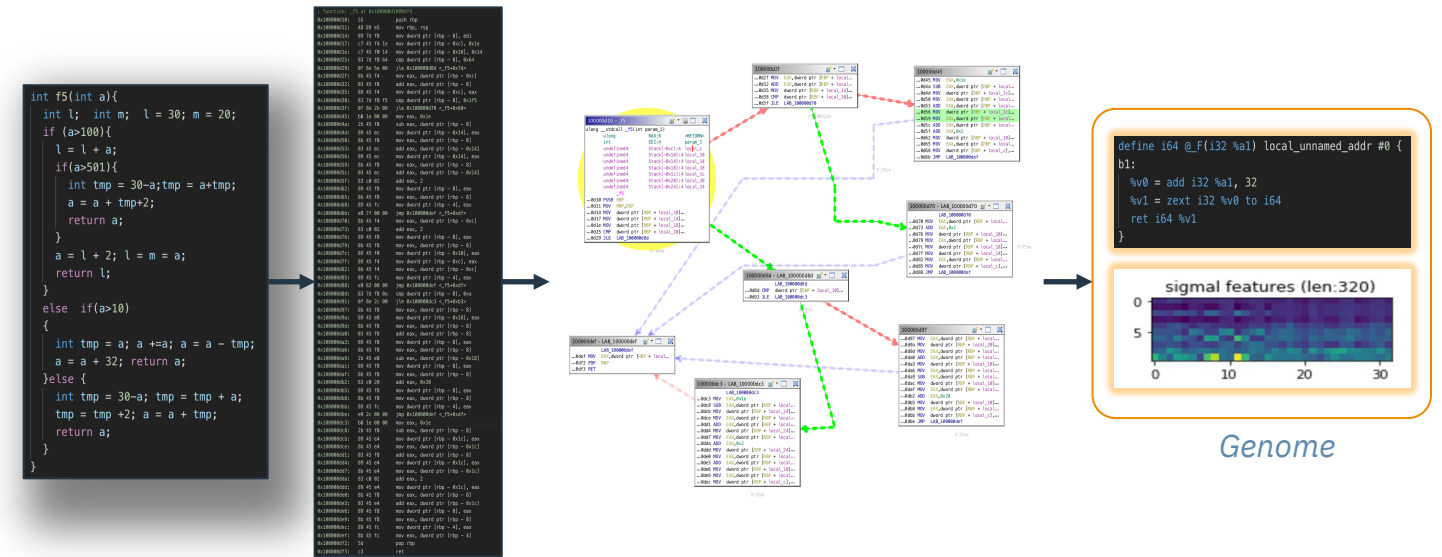
Code Genome: Semantically meaningful fingerprint



Advantages and Challenges

Advantages

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



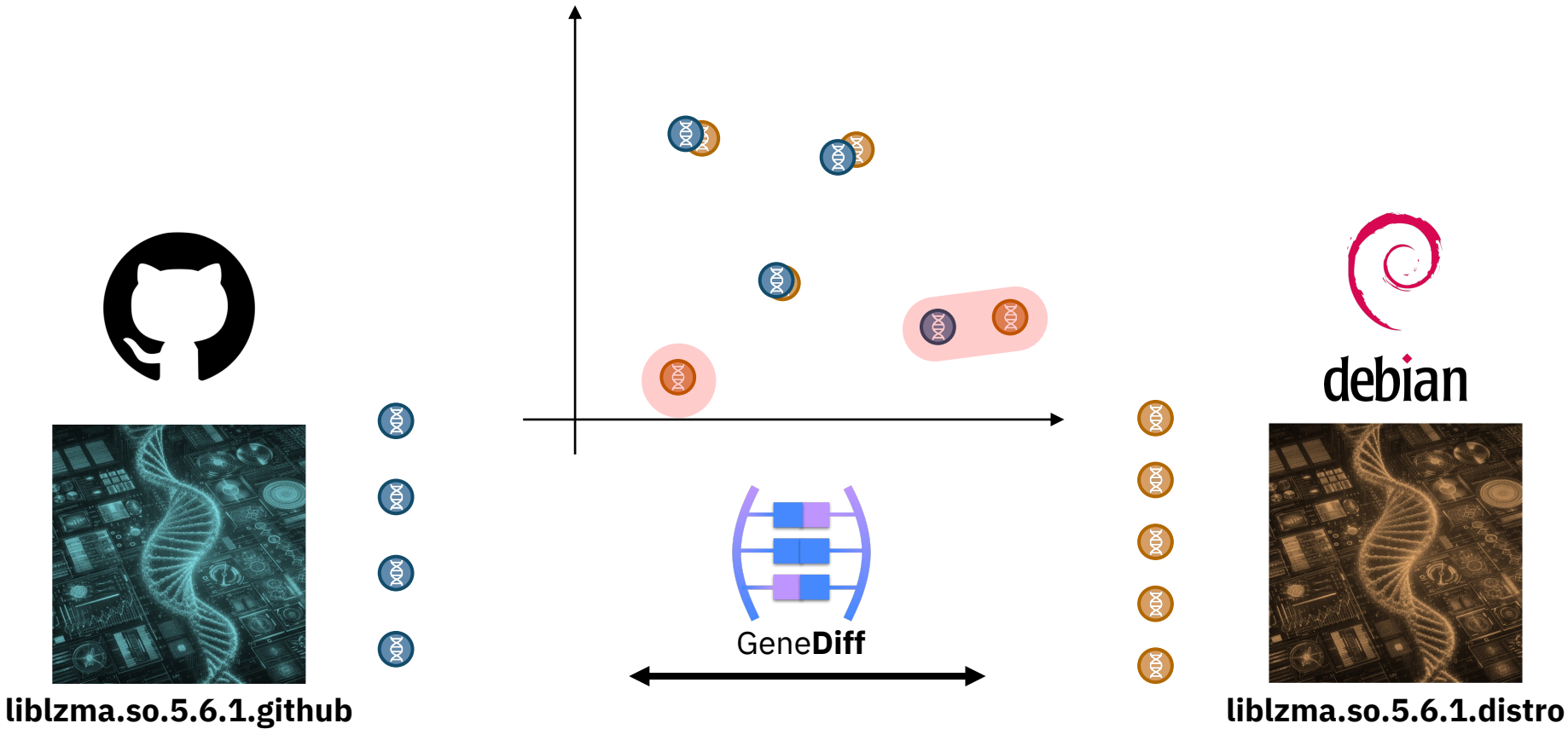
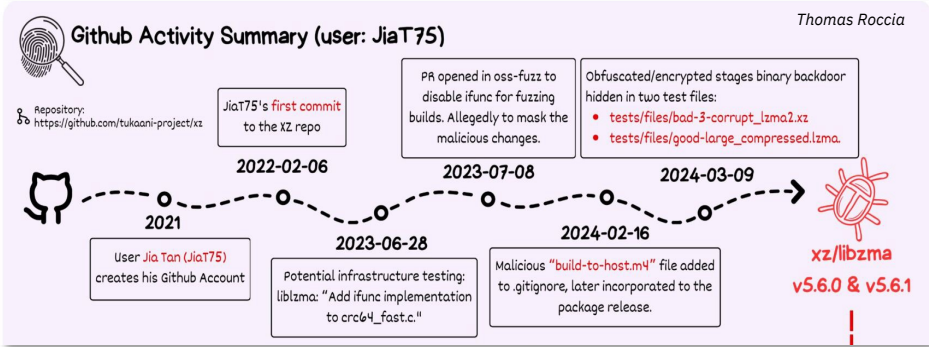
Challenges

- Disassembly is undecidable
- Function boundary identification
- Loss of architecture specific nuances
- Canonicalization cannot completely recover high-level abstraction

objdump	Ghidra	retdec	IDA
Disassembly of section <code>__eolnwiw:</code> <pre> 007d1000: 00 7d1000 <eolnwiw>: 7d1000: 56 push %esi 7d1001: 58 push %eax 7d1002: 53 push %ebx 7d1003: e8 01 00 00 00 call 0x7d1009 7d1008: 00 58 09 add %bl, -0x77(%eax) 7d100b: c3 ret 7d100c: 40 inc %eax 7d100d: c2 ret 7d100e: 2d 00 00 16 sub \$0x168800,%eax 7d1012: 2d ac b0 0b 10 sub \$0x100bb0ac,%eax 7d1017: 05 03 b0 0b 10 add \$0x100bb0a3,%eax 7d101c: 00 3c cc cmp %kcc, (%ebx) 7d101f: 75 10 jnb 0x7d103a 7d1021: c6 03 00 movb %0xb, (%ebx) 7d1024: b6 10 00 00 00 mov %0x1000,%eax 7d1029: 68 b5 f3 44 34 push \$0x3444f3b5 7d102e: 68 f3 6d 5f 1c push \$0x1c5fd6f3 7d1033: 53 push %ebx 7d1034: 50 push %eax 7d1035: e8 0a 00 00 00 call 0x7d1044 7d103a: 83 c0 00 add \$0x0,%eax 7d103d: 89 44 24 08 mov %eax,0x8(%esp) 7d1041: 5b pop %ebx 7d1042: 58 pop %eax 7d1043: c3 ret 7d1044: 53 push %ebp 7d1045: 89 e5 mov %esp,%ebp 7d1047: 50 push %eax 7d1048: 53 push %ebx </pre>	<pre> undefined entry() AL: 1 <RETURN> entry 007d1000 56 PUSH ESI 007d1001 58 PUSH EAX 007d1002 53 PUSH EBX 007d1003 e8 01 00 00 CALL LAB_007d1009+1 007d1008 00 00 LAB_007d1008+1 007d1008 00 58 09 ADD byte ptr [EAX + -0x77],BL 007d100b c3 RET 007d100c 40 40h 0 007d100d 2d 2Dh - 007d100e 05 00h 0 007d100f 00 00h 0 007d1010 75 10h 0 007d1011 00 00h 0 007d1012 2d 2Dh - 007d1013 ac Ach 0 007d1014 b6 B6h 0 007d1015 05 00h 0 007d1016 10 10h 0 007d1017 05 05h 0 007d1018 43 43h 0 007d1019 75 00h 0 007d101a 06 06h 0 007d101b 10 10h 0 007d101c 00 00h 0 007d101d 3b 3Bh 0 007d101e 75 00h 0 007d101f 75 75h u 007d1020 19 19h 0 007d1021 c6 C6h 0 007d1022 03 03h 0 007d1023 53 00h 0 </pre>	<pre> section: eolnwiw function: entry_point at 0x7d1000 -- 0x7d1008 0x7d1000: 56 push esi 0x7d1001: 58 push eax 0x7d1002: 53 push ebx 0x7d1003: e8 01 00 00 00 call 0x7d1009 <function_7d1009> ; data inside code section at 0x7d1008 -- 0x7d1009 0x7d1008: 00 58 09 ; function: function_7d1008 at 0x7d1009 -- 0x7d1044 0x7d1009: 53 pop eax 0x7d100a: 58 mov ebx, eax 0x7d100b: c3 inc eax 0x7d100c: 40 sub eax, 0x168800 0x7d100d: c2 sub eax, 0x100bb0ac 0x7d100e: 05 sub eax, 0x100bb0a3 0x7d100f: 75 10 sub eax, 0x100bb0b5 0x7d1010: c6 03 mov byte ptr [ebx], 0xb 0x7d1011: 00 3c cmp byte ptr [ebx], 0xcc 0x7d1012: 75 10 jnb byte ptr [ebx], 0x7d103a 0x7d1013: 68 b5 f3 44 34 mov byte ptr [ebx], 0x3444f3b5 0x7d1014: 68 f3 6d 5f 1c mov byte ptr [ebx], 0x1c5fd6f3 0x7d1015: 53 push ebx 0x7d1016: 50 push eax 0x7d1017: e8 0a 00 00 00 call 0x7d1044 <function_7d1044> 0x7d1018: 83 c0 add eax, 0 0x7d1019: 89 44 24 08 mov byte ptr [ebx], 0x8 0x7d101a: 5b pop ebx 0x7d101b: 58 pop eax 0x7d101c: c3 push [esp + 0], eax 0x7d101d: 53 pop ebx 0x7d101e: 50 pop eax 0x7d101f: c3 ret </pre>	<pre> eolnwiw:007d1000 public start eolnwiw:007d1000 start proc near eolnwiw:007d1001 push esi eolnwiw:007d1002 push eax eolnwiw:007d1003 push ebx eolnwiw:007d1003 call loc_7d1009 eolnwiw:007d1003 db 0 eolnwiw:007d1008 eolnwiw:007d1009 eolnwiw:007d1009 loc_7d1009: pop eax eolnwiw:007d1009 mov ebx, eax eolnwiw:007d100A mov ebx, eax eolnwiw:007d100C inc eax eolnwiw:007d100D sub eax, 168800h eolnwiw:007d1012 sub eax, 100BB0ACh eolnwiw:007d1017 add eax, 100BB0B5h eolnwiw:007d101C cmp byte ptr [ebx], 0Ch eolnwiw:007d101F jnb short loc_7d103A eolnwiw:007d1021 mov byte ptr [ebx], 0 eolnwiw:007d1021 mov ebx, 1000h eolnwiw:007d1024 push 3444F3B5h eolnwiw:007d102E push 1C5FD6F3h eolnwiw:007d1033 push ebx eolnwiw:007d1034 push eax eolnwiw:007d1035 call sub_7d1044 eolnwiw:007d103A mov eax, 0 eolnwiw:007d103A loc_7d103A: push [esp+0], eax eolnwiw:007d103B add eax, 0 eolnwiw:007d103B mov [esp+0], eax eolnwiw:007d1041 pop ebx eolnwiw:007d1042 pop eax eolnwiw:007d1043 ret eolnwiw:007d1043 start eolnwiw:007d1043 endp eolnwiw:007d1043 db 0 </pre>

Uncovering **Supply Chain Attack**

Demo 1: xz backdoor analysis using Code Genome



Demo 1: xz backdoor analysis using Code Genome

Code Genome

Compare

00:00:21

Drag and drop files here or click to upload

Reset

Swap

Drag and drop files here or click to upload

00:00:26

663500a4dbcb9faaac802eb3c4b5b0dd5c0333df473d7368b0bc53a32f0c363a

4bcc50a99a4cae2e1053ea964540f249f97ca35118ebe61d2789be1c84c3e93

Gene similarity: 70

Identical: 191

Similar: 95

Mismatch: 27

Deletions:

Additions: 101

File Name:liblzma.so.5.6.1.github

File Hash663500a4dbcb9faaac802eb3c4b5b0dd5c0333df473d7368b0bc53a32f0c363a

File Type

Last updated:2024-08-05T03:51:35.000Z

File size:1240992

Gene count:306

File Name:liblzma.so.5.6.1.distro

File Hash4bcc50a99a4cae2e1053ea964540f249f97ca35118ebe61d2789be1c84c3e93

File Type

Last Updated:2024-08-05T03:51:44.000Z

File size:1306808

Gene count:407

liblzma.so.5.6.1.github (663500a4) Functions

liblzma.so.5.6.1.distro (4bcc50a) Functions

Score

op

Actions

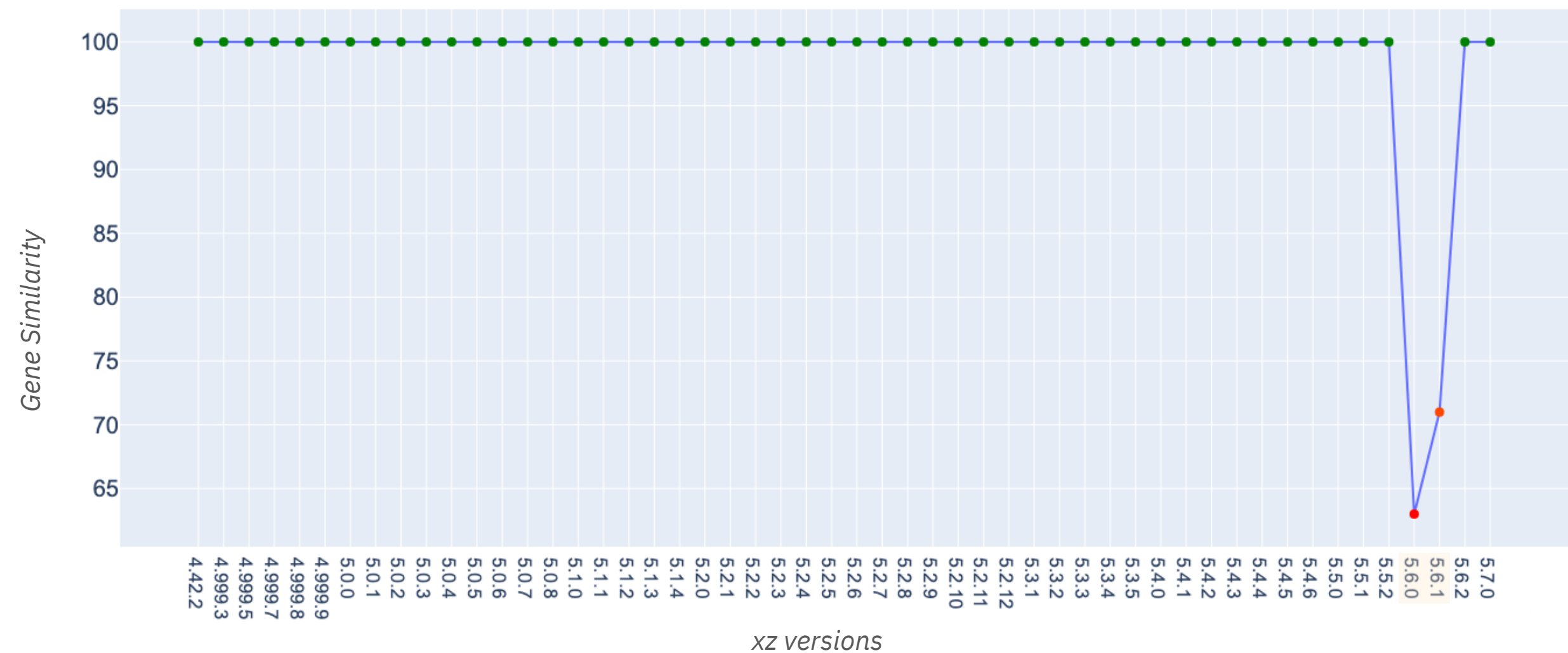
crc32_resolve	crc32_resolve	98	!	:
crc64_resolve	crc64_resolve	95	!	:
get_literal_price	get_literal_price	98	!	:
get_options	get_options	98	!	:
hash_append	hash_append	98	!	:
lz_encoder_prepare	lz_encoder_prepare	98	!	:
lzma2_bound.part.0	lzma2_bound.part.0	97	!	:
lzma_delta_encoder_init	lzma_delta_encoder_init	98	!	:
lzma_index_buffer_decode	lzma_index_buffer_decode	97	!	:
lzma_index_memusage	lzma_index_memusage	98	!	:
lzma_lz_encoder_init	lzma_lz_encoder_init	98	!	:

xz

Name	Date Modified	Size	Kind
liblzma.so.5.6.1.github	Yesterday, 10:36 AM	1.2 MB	Document
liblzma.so.5.6.1.distro	Yesterday, 10:36 AM	1.3 MB	Document
liblzma.so.5.2.9.github	Yesterday, 10:36 AM	1 MB	Document
liblzma.so.5.2.9.distro	Yesterday, 10:36 AM	1 MB	Document

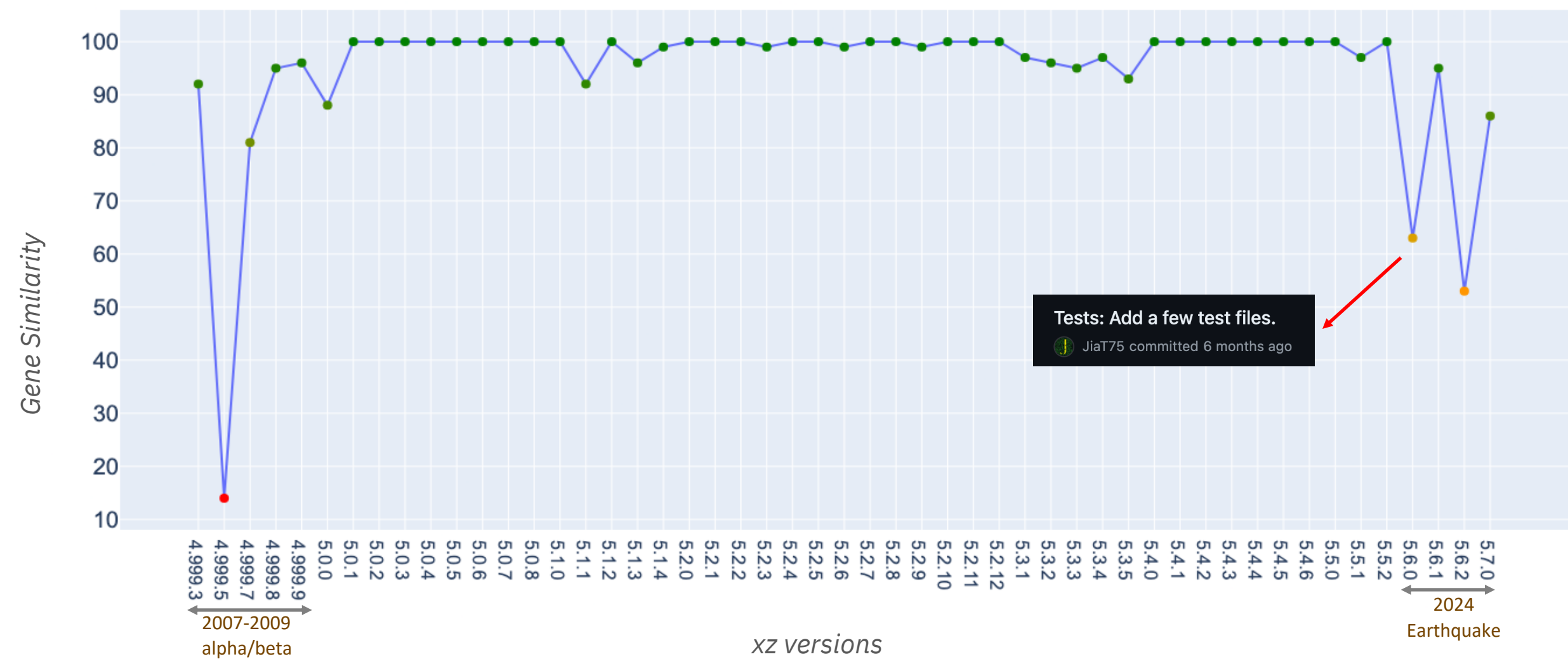
Macintosh HD > Users > dkirat > Downloads > xz
4 items, 352.3 GB available

xz backdoor Gene Similarity Analysis using GeneDiff



Local vs distribution builds of same version

xz backdoor Gene Similarity Analysis using GeneDiff



Incremental version similarity in distribution builds

Improving **Supply Chain Security**

Trust but Verify SBOM: Metadata vs. Code

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)?

\$ 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"
    }
  }
],
```

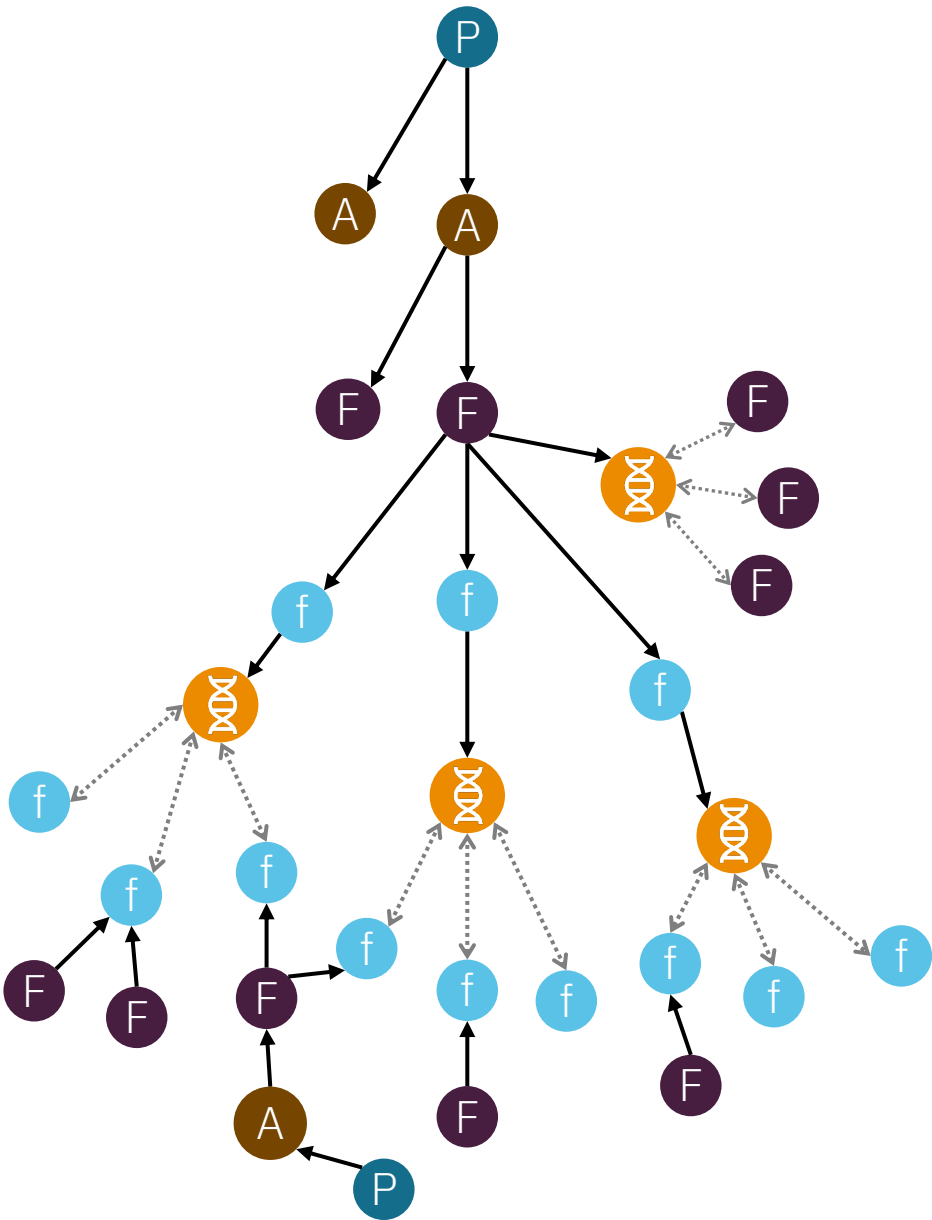
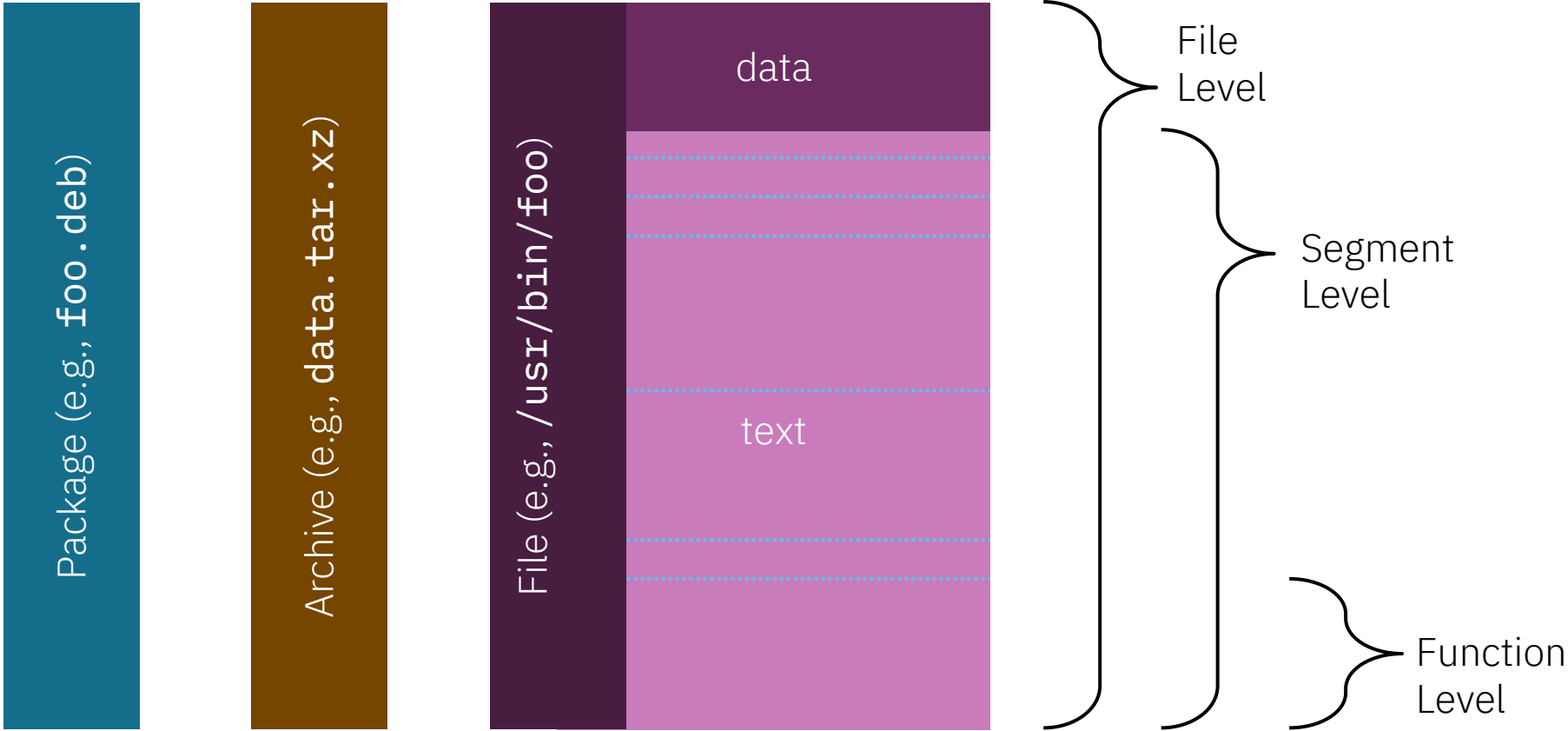
```
Dockerfile > ...
1 FROM ubuntu:focal
2
3 RUN apt-get update
4 RUN apt-get install -y wget
5
6 RUN mv /var/lib/dpkg/status /var/lib/dpkg/status.bak
7 RUN touch /var/lib/dpkg/status
8
9 RUN apt-get update
10

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

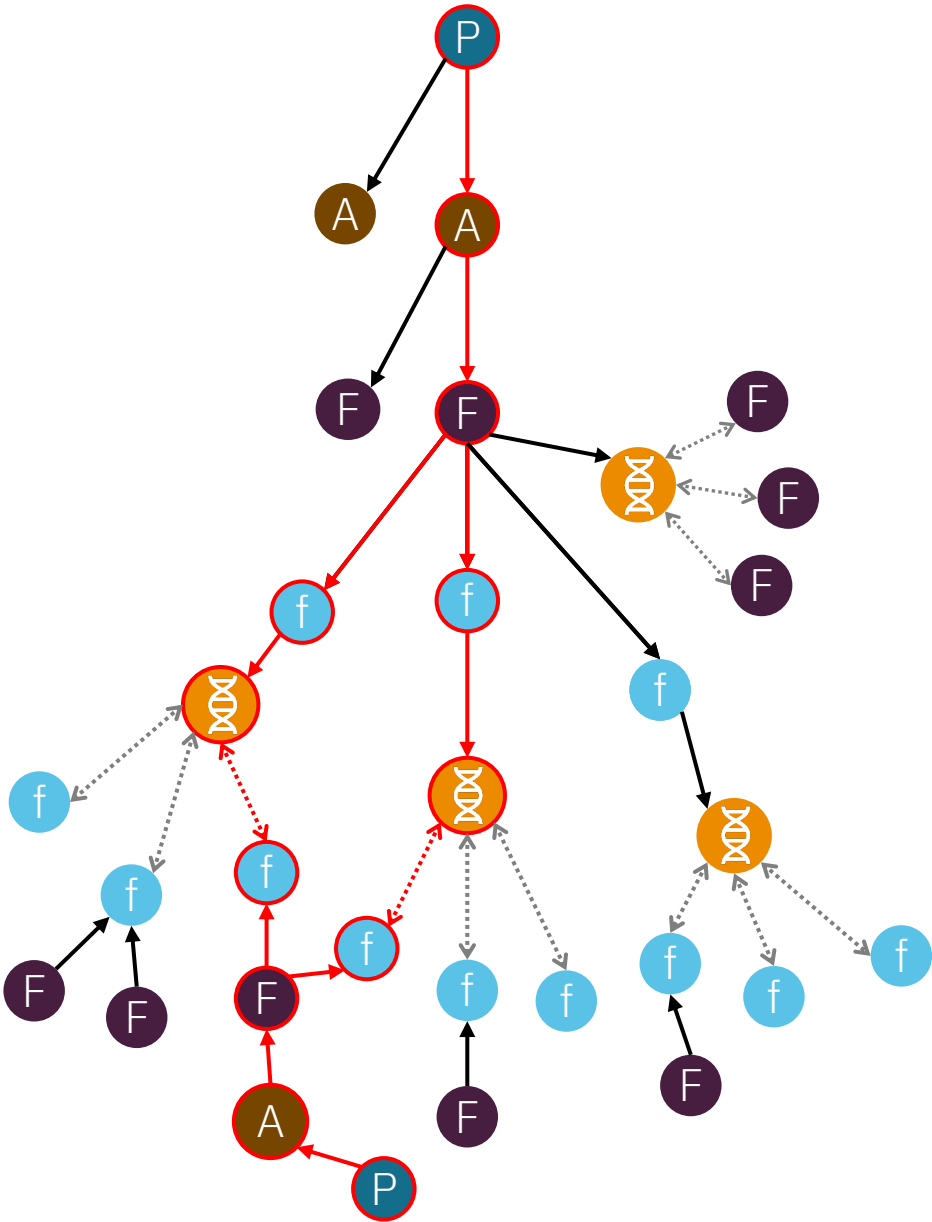
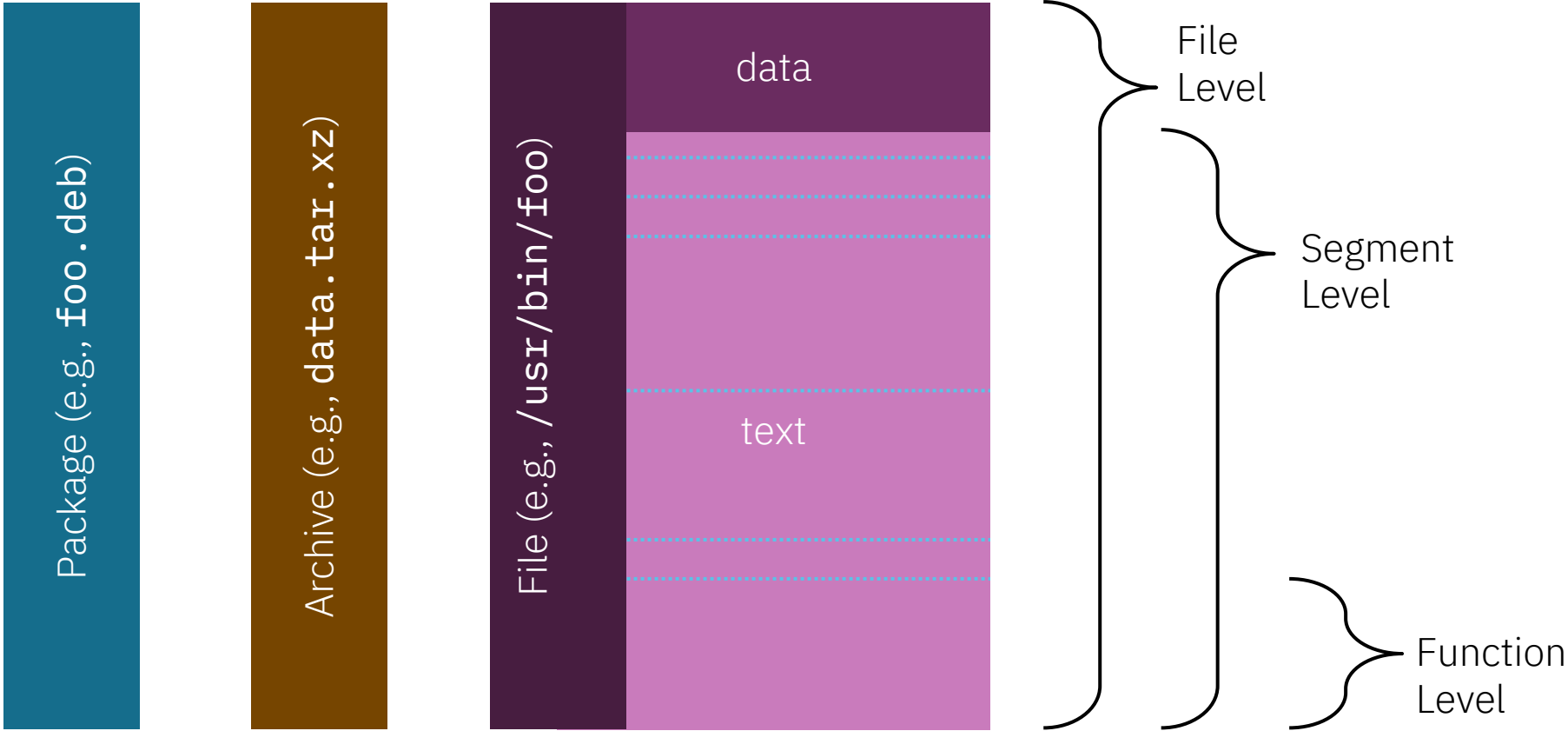
“Unfortunately, some images – such as the [official node image on Docker Hub](#) – incorrectly report the version of OpenSSL that's used by the Node.js runtime.”

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

Knowledge Graph: Gene Granularity



Knowledge Graph: Gene Granularity



Demo 2: 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
```

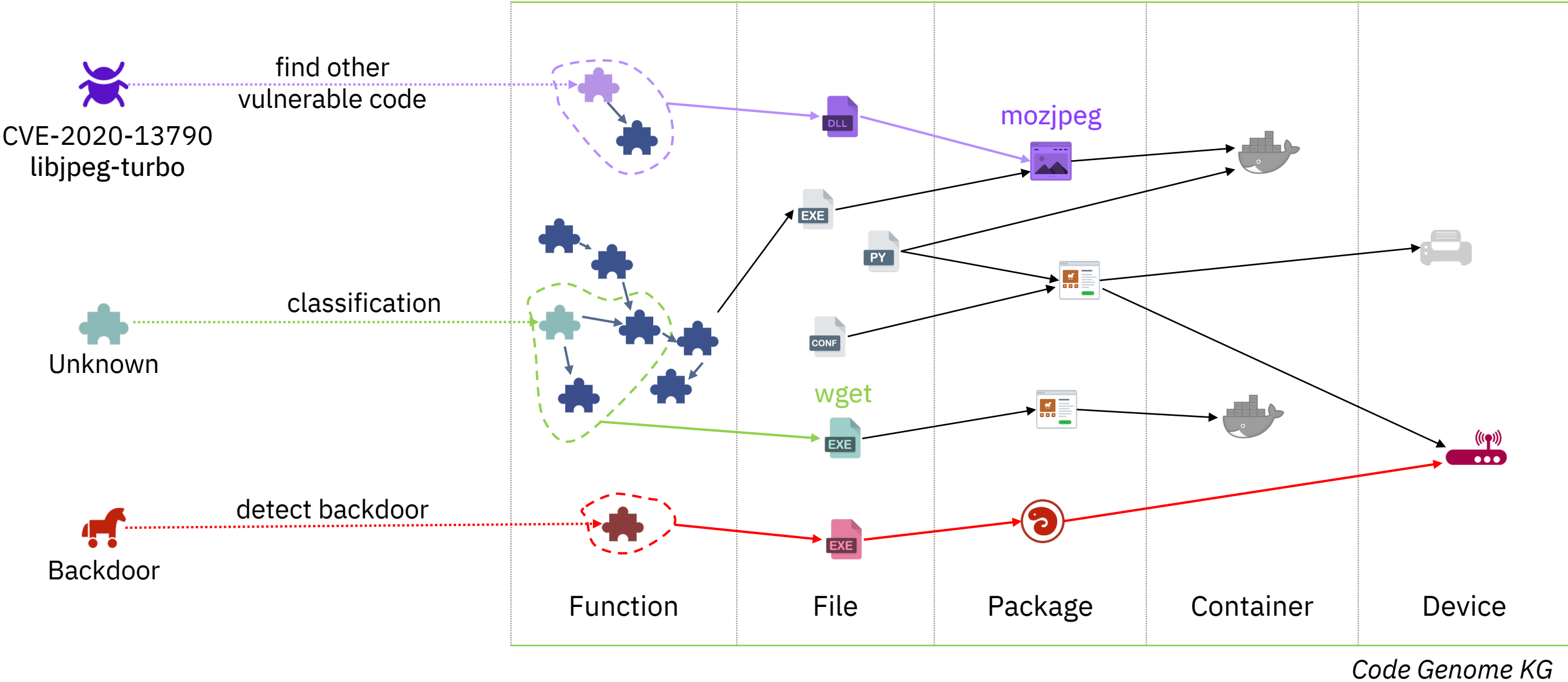


SBOM generated by Code Genome

☐	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

Knowledge Graph: Code Genome and Use Cases



Open Sourcing **Code Genome**

Status and Roadmap

Open-source tools

- **Code Genome Framework**
 - GeneDiff, Basic KG, CLI tools, and GUI
 - Currently supported
 - Binaries: ELF, PE, Mach-O
 - Architectures: x86, x86_64, arm, aarch64, mips, ppc
 - Optimized canonicalization
- **Jaudit**
 - JAR file support
 - JAR version identification
 - CVE annotation

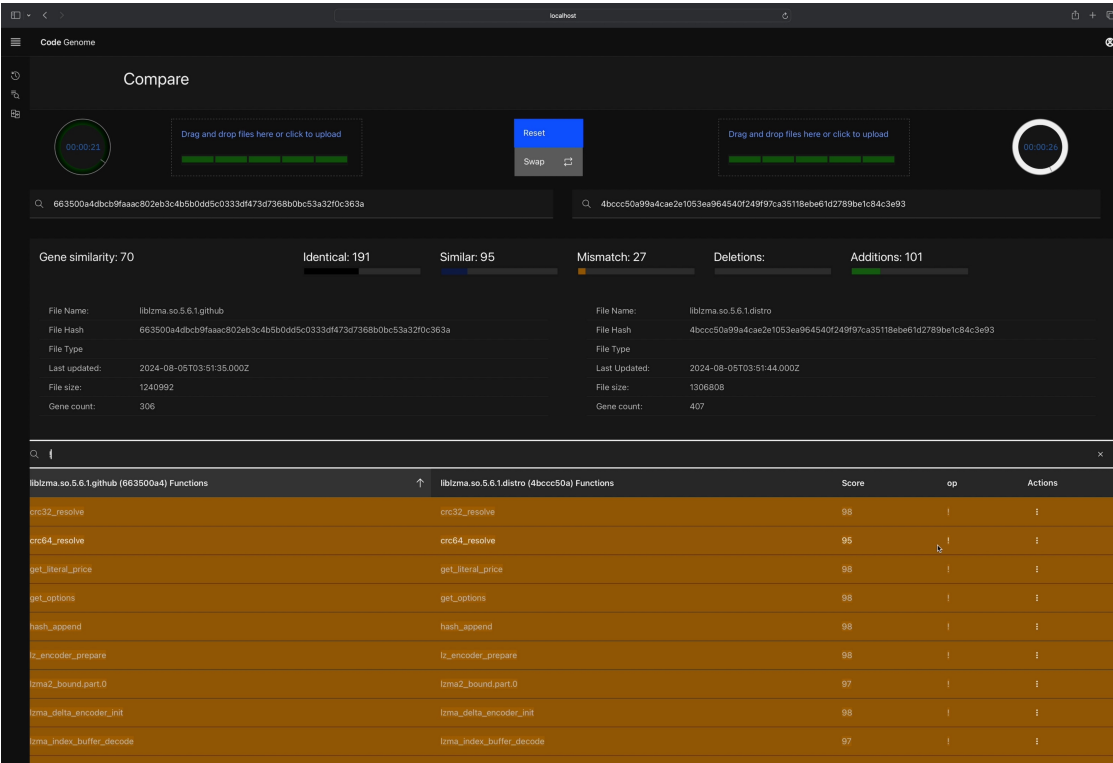
Next steps

- Support
 - Packages: deb, rpm, ipa
 - Archives: ar, cpio, tar, bzip2, gzip, zstd, xz, rar, 7zip



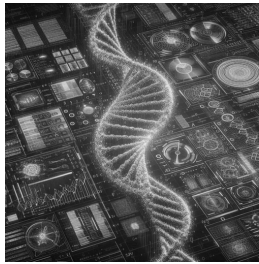
```
git clone https://github.com/code-genome/codegenome.git
cd codegenome

make start
```



Takeaways

Semantic Gap



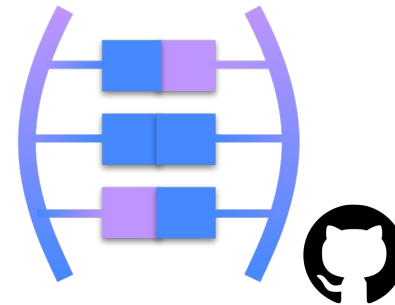
Code



Metadata

Inherent semantic gap breaks the transfer of trust from metadata to code

Code Genome



Now open-sourced Code Genome Framework can help bridge that gap

Supply Chain Security



Detection of XZ-backdoor demonstrates framework's capability in improving supply chain security



black hat[®]

USA 2024

Dhilung Kirat ✉ dkirat@us.ibm.com

Jiyong Jang ✉ jjang@us.ibm.com

IBM Research



github.com/code-genome