Code Genome:
Fingerprinting Code to Help Secure the Software Supply Chain

—

Jiyong Jang

IBM Research Security

IBM

IBM Team

Jeff Borek

Carlos Fonseca

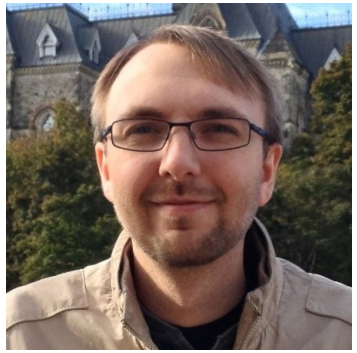
Ted Habeck

Nigel Hinds

Jiyong Jang

Dhilung Kirat

Arnaud Le Hors

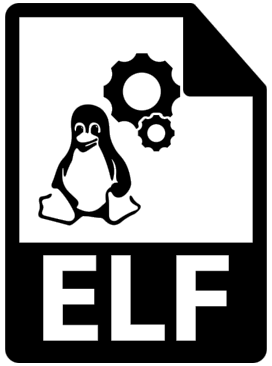


Ian Molloy

JR Rao

Matt Rutkowski

Doug Schales



- Here is my application.
- I signed it with this certificate.
- It uses these libraries.
- Please install it.
- What do you do?

"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.

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.

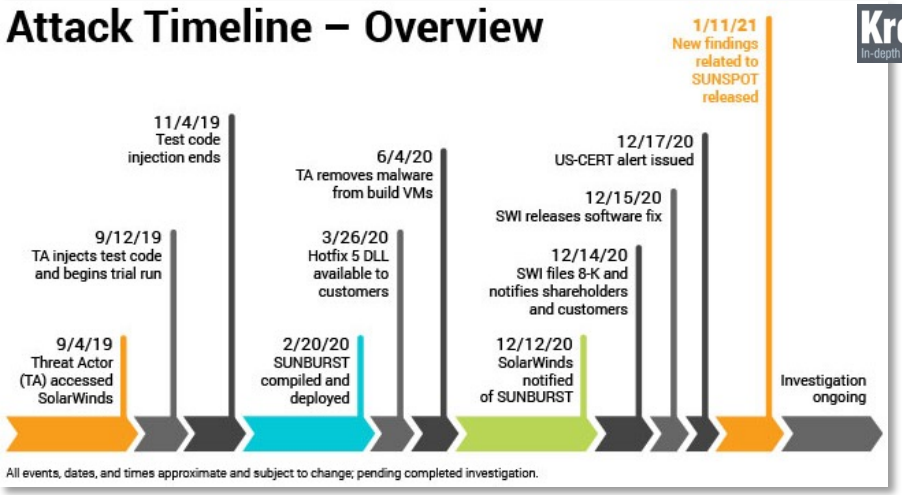
More precisely stated, the problem is to write a source program that, when compiled and executed, will produce as output an exact copy of its source. If you have never done this, I urge you to try it on your own. The discovery of how to do it is a revelation that far surpasses any benefit obtained by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

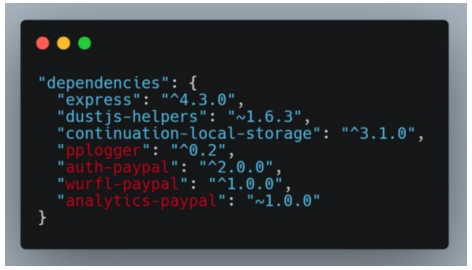
Figure 1 shows a self-reproducing program in the C[3] programming language. (The purist will note that the program is not precisely a self-reproducing program, but will produce a self-reproducing program.) This entry is much too large to win a prize, but it demonstrates the technique and has two important 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.

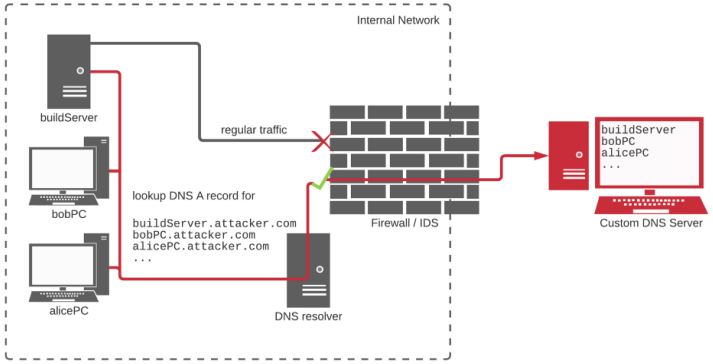
[1] UNIX is a trademark of AT&T Bell Laboratories.

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







Codecov breach impacted 'hundreds' of customer networks: report

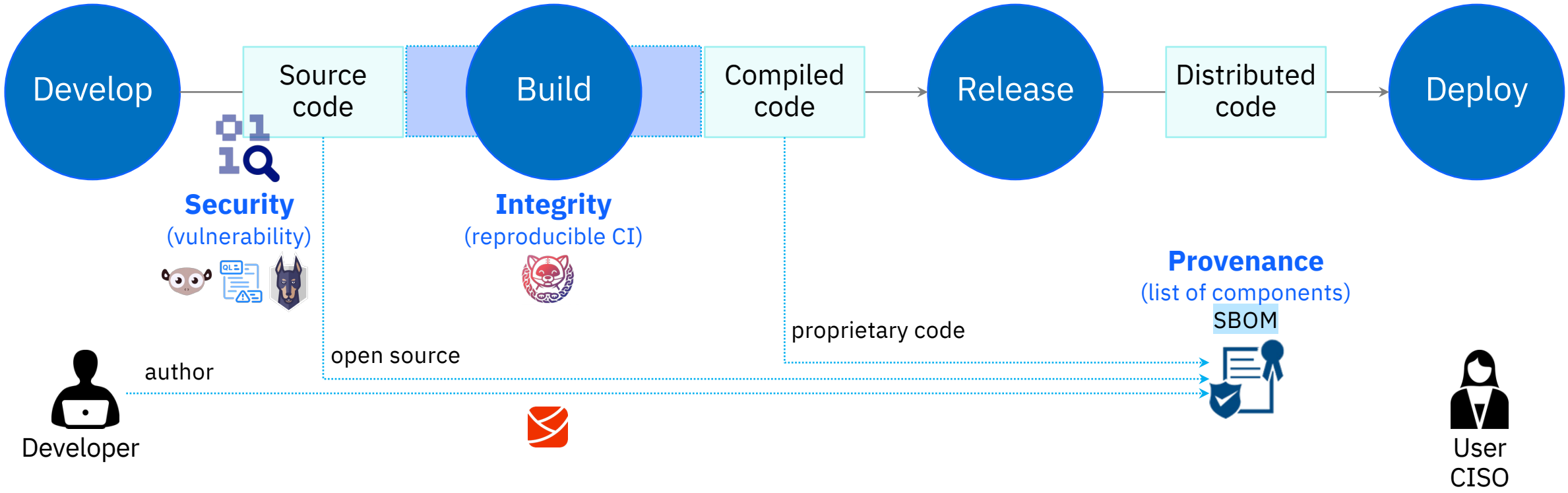
ZDNet

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

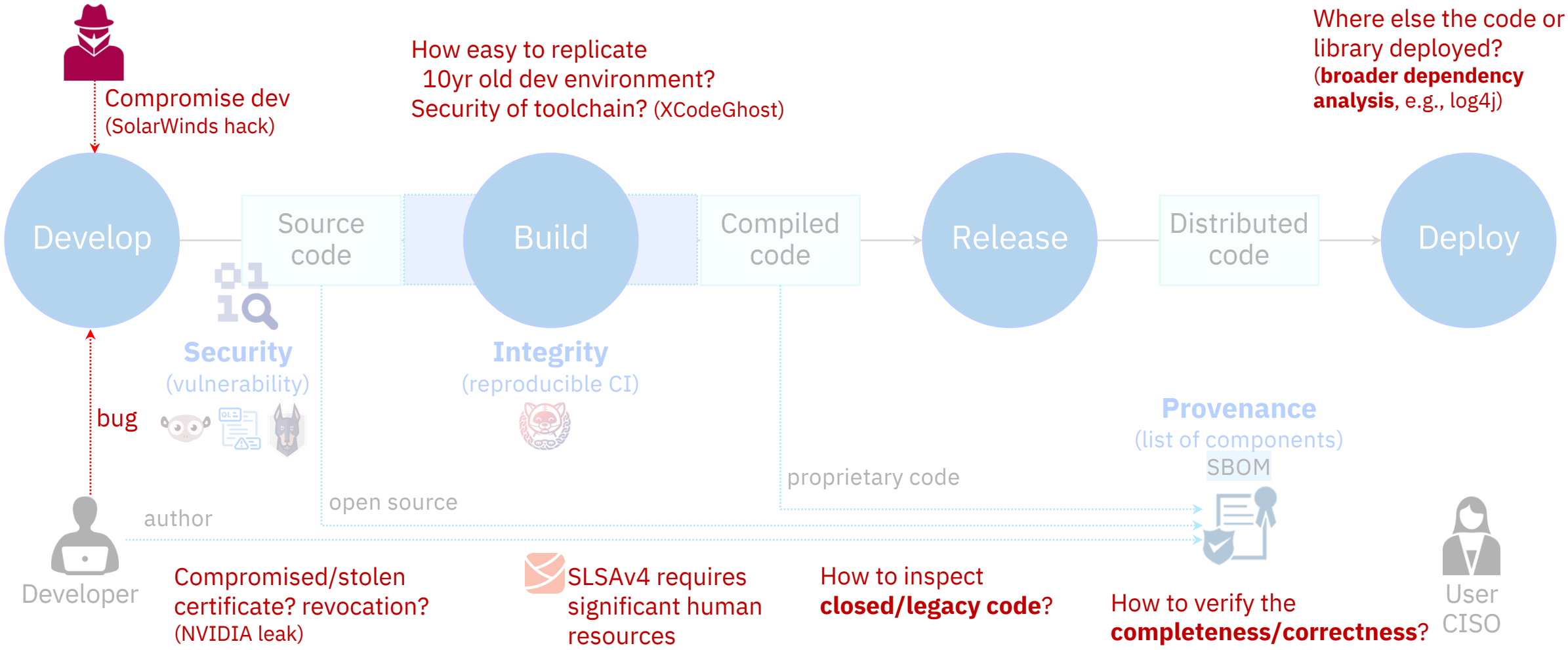
Supply Chain Security

Industry approach to protecting CI/CD pipelines

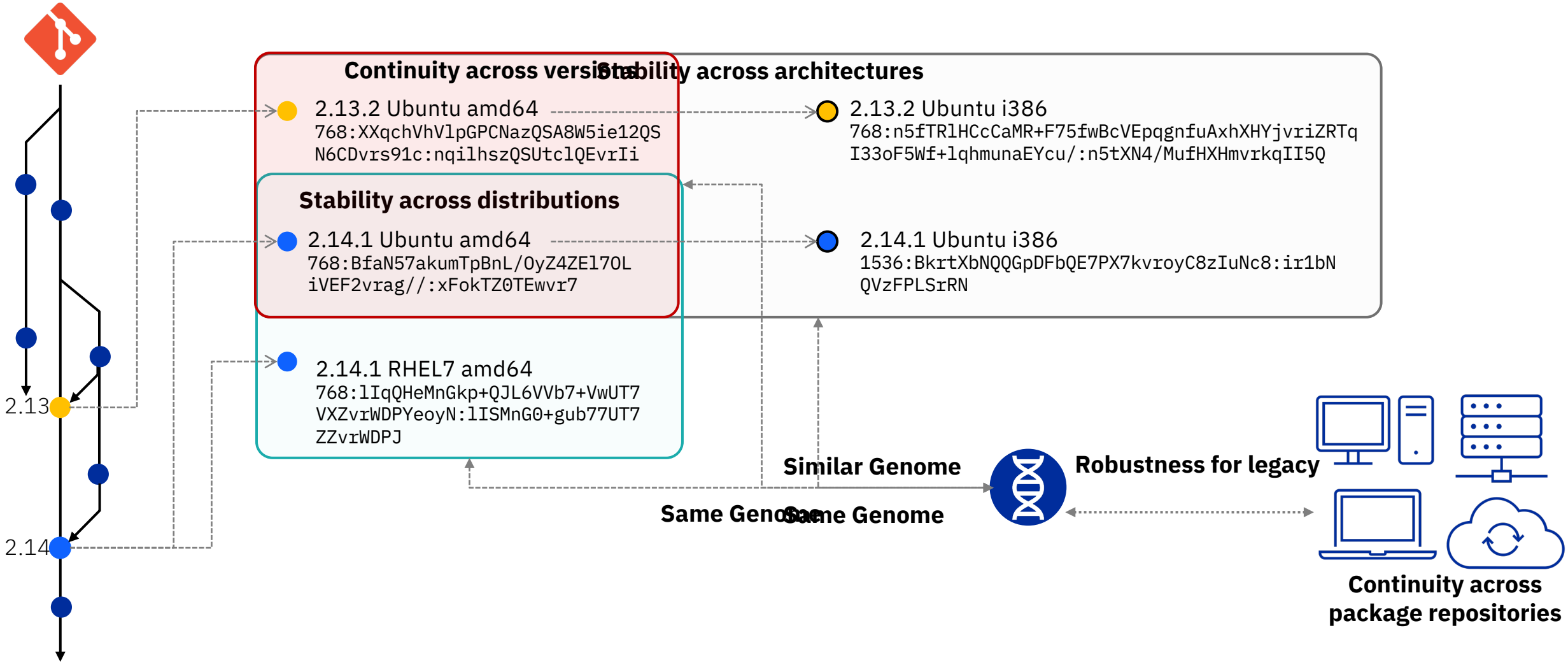


Supply Chain Security

Open security issues and residual risks



Software Fingerprints for Software Assurance



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

Code Genome: Semantically meaningful fingerprint

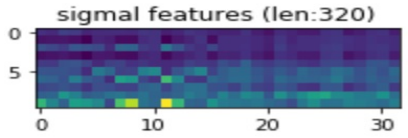
```
int f2(int a){
  int local=31;
  local +=1;
  local = a + local;
  return local;
}
```

```
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 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;
  }
}
```

```
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;
  }
}
```

```
define i64 @_F(i32 %a1) local_unnamed_addr #0 {
b1:
  %v0 = add i32 %a1, 32
  %v1 = zext i32 %v0 to i64
  ret i64 %v1
}
```



Same Genome

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

Key Idea: Code Genome construction

machine-code

```
; section: __text
; function: _f1 at 0x100000c20 -- 0x100000c35
0x100000c20:   55           push rbp
0x100000c21:   48 89 e5     mov rbp, rsp
0x100000c24:   89 7d fc     mov dword ptr [rbp - 4], edi
0x100000c27:   8b 7d fc     mov edi, dword ptr [rbp - 4]
0x100000c2a:   83 c7 20     add edi, 0x20
0x100000c2d:   89 7d f8     mov dword ptr [rbp - 8], edi
0x100000c30:   8b 45 f8     mov eax, dword ptr [rbp - 8]
0x100000c33:   5d           pop rbp
0x100000c34:   c3           ret
```

Compile

source code (optional)

```
int f1(int a){
  int x;
  x = a + 32;
  return x;
}
```

Lift

"raw" IR

```
define i64 @_f1(i32 %arg1) local_unnamed_addr {
dec_label_pc_100000c20:
  %tmp5 = zext i32 %arg1 to i64
  %v2_100000c2a = add nuw nsw i64 %tmp5, 32
  %v17_100000c2a = and i64 %v2_100000c2a, 4294967295
  ret i64 %v17_100000c2a
}
```

Canonicalize

canonical IR

```
define i64 @_F(i32 %a1) local_unnamed_addr #0 {
b1:
  %v0 = add i32 %a1, 32
  %v1 = zext i32 %v0 to i64
  ret i64 %v1
}
```

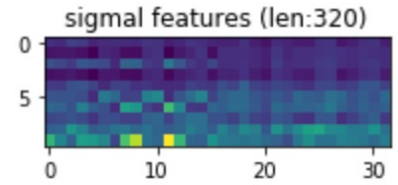
Convert

bitcode

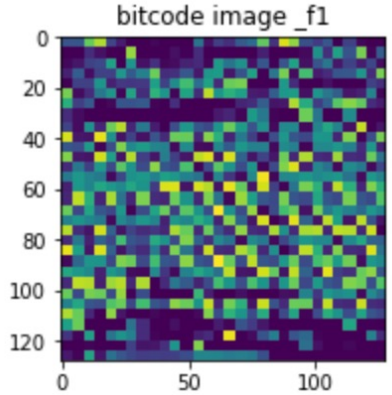
```
  Offset: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000: 42 43 C0 DE 35 14 00 00 05 00 00 00 62 0C 30 24    BC@^5.......b.0$
00000010: 49 59 BE 26 EF D3 3E 2D 44 01 32 05 00 00 00 00    IY>&oS>-D.2.....
00000020: 21 0C 00 00 D5 00 00 00 0B 02 21 00 02 00 00 00    !...U.....!.....
00000030: 16 00 00 00 07 81 23 91 41 C8 04 49 06 10 32 39    ......#.AH.I..29
00000040: 92 01 84 0C 25 05 08 19 1E 04 8B 62 80 0C 45 02    ....%......b..E.
00000050: 42 92 0B 42 64 10 32 14 38 08 18 4B 0A 32 32 88    B..Bd.2.8..K.22.
00000060: 48 70 C4 21 23 44 12 87 8C 10 41 92 02 64 C8 08    HpD!#D....A..dH.
00000070: B1 14 20 43 46 88 20 C9 01 32 32 84 18 2A 28 2A    1..CF..I.22..*(*
00000080: 90 31 7C B0 5C 91 20 C3 C8 00 00 00 89 20 00 00    .1|0\..CH......
00000090: 0C 00 00 00 32 22 C8 08 20 64 85 04 93 21 A4 84    ....2"H..d...!$.
000000a0: 04 93 21 E3 84 A1 90 14 12 4C 86 8C 0B 84 64 4C    ..!c.!...L....dL
000000b0: 10 18 73 04 A0 30 47 00 06 45 40 48 03 01 23 00    ..s..0G..E@H..#.
```

Convert

Genome

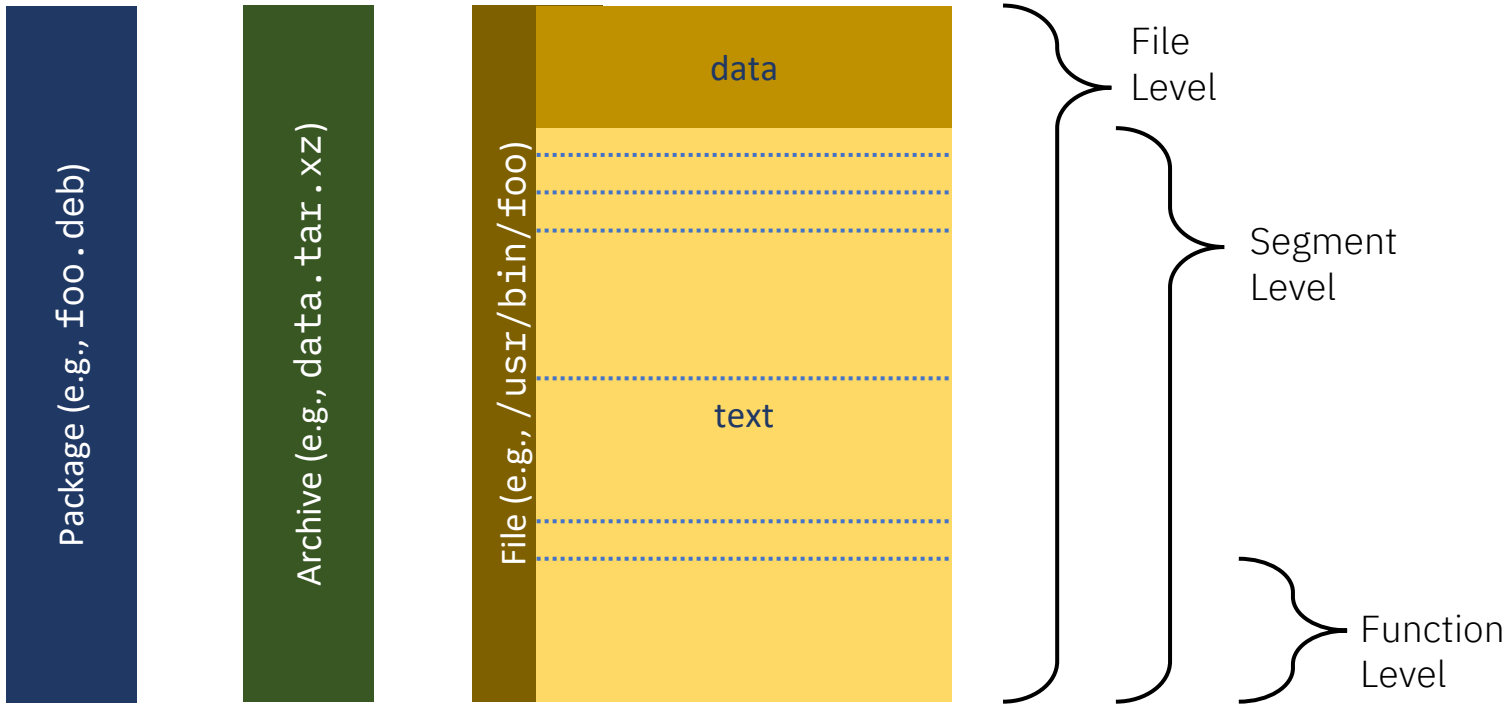


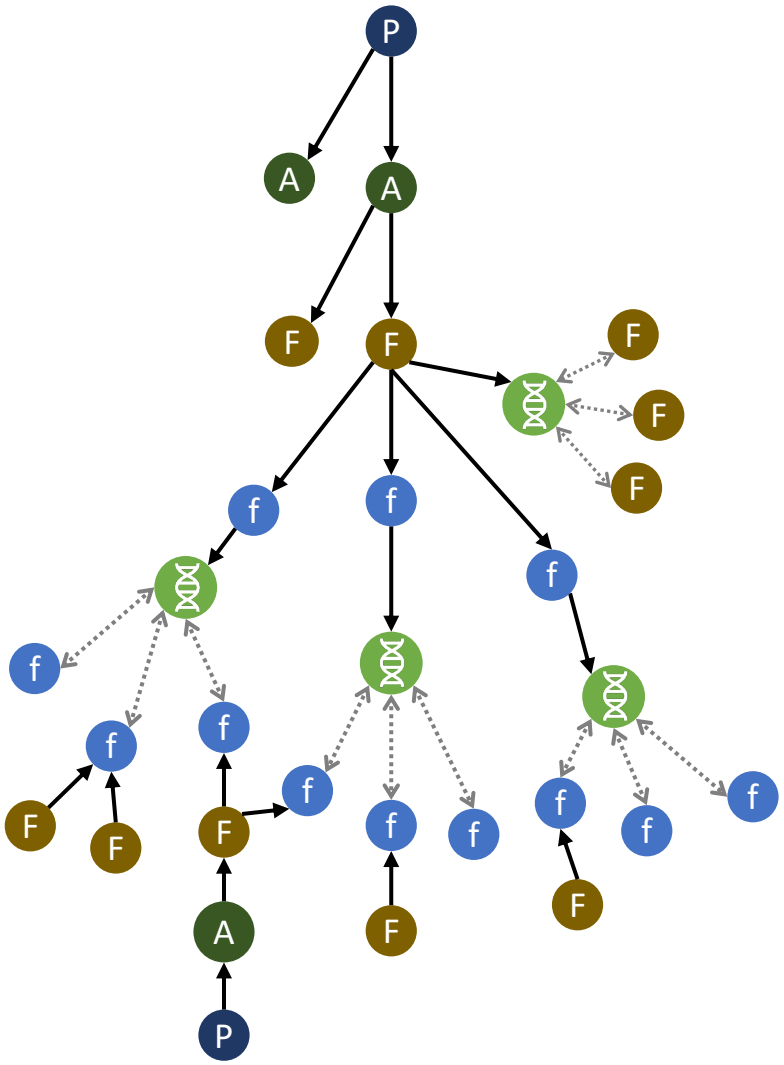
Embedding



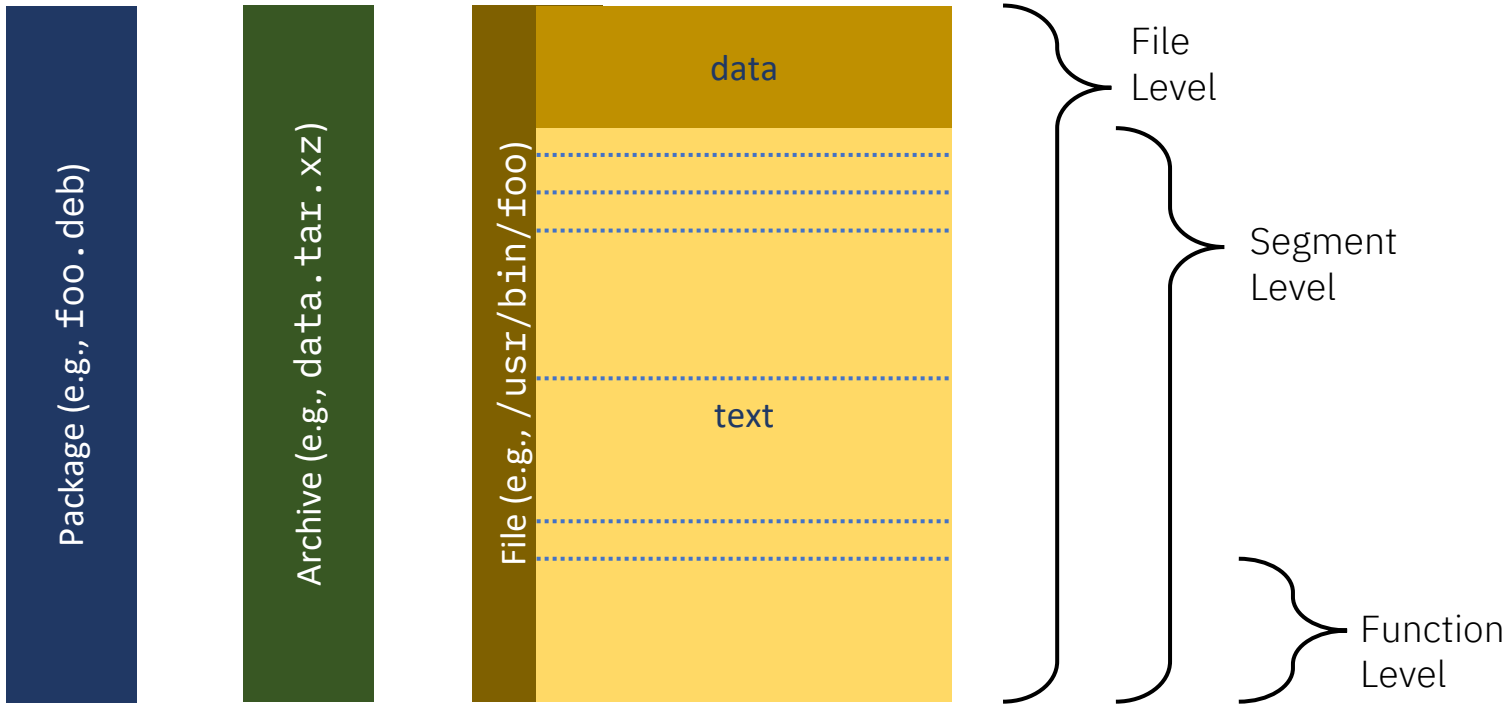
Genome can be constructed from closed-source/legacy code where source code is not easily available.

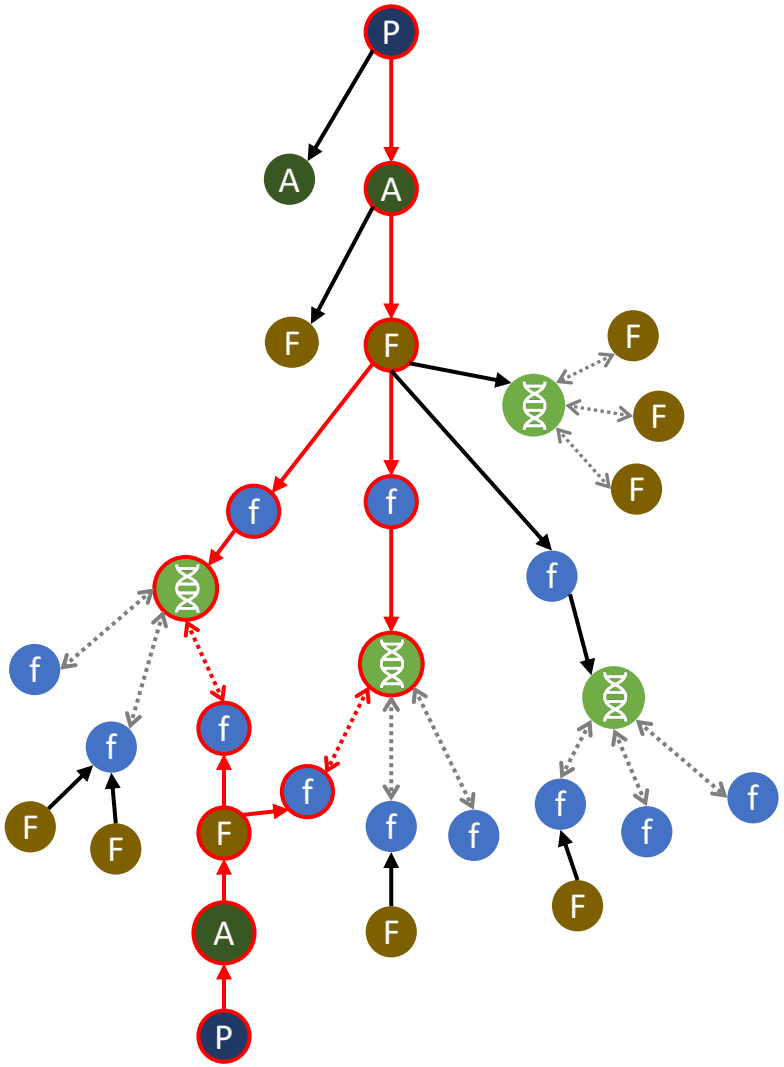
Code Genome Granularity



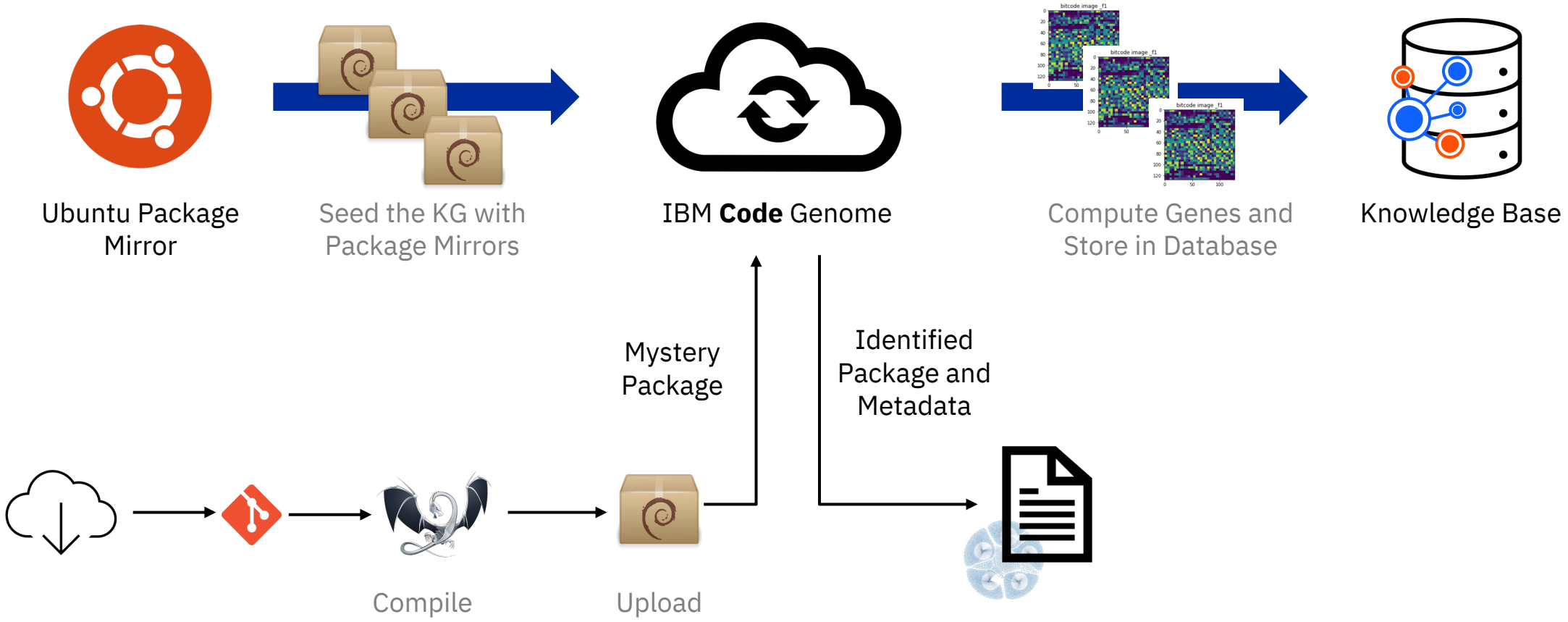


Code Genome Granularity





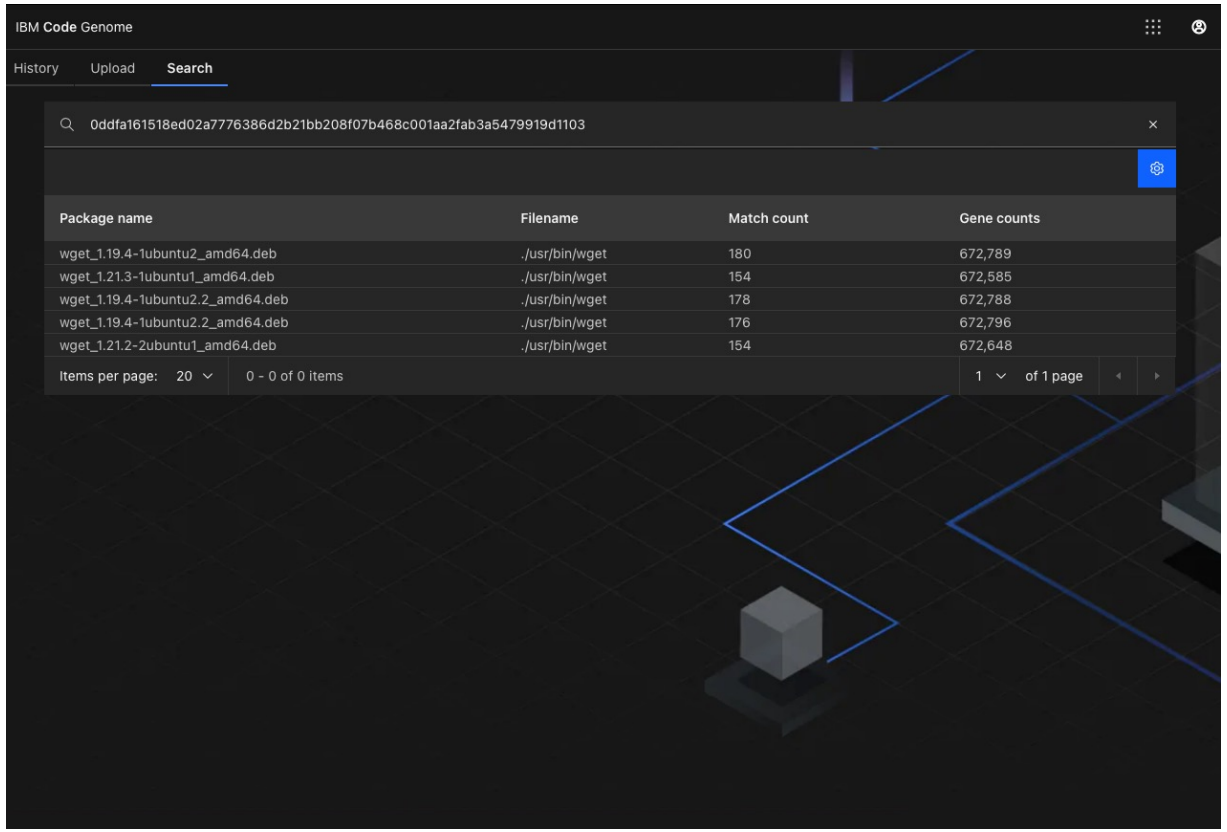
Demo Setup


Ubuntu Package Mirror
Seed the KG with Package Mirrors
IBM **Code** Genome
Compute Genes and Store in Database
Knowledge Base
Mystery Package
Identified Package and Metadata
Compile
Upload

Demo: Verifying `wget`

```
~/code-genome ❯ ls -al                                                                  jjang@code-genome
total 4300
drwxrwxr-x  3 jjang jjang    4096 Nov  7 07:24 .
drwxr-x--- 18 jjang jjang    4096 Nov  7 07:46 ..
drwxrwxr-x 14 jjang jjang    4096 Nov  7 01:25 wget-1.19.4
-rw-r--r--  1 jjang jjang   66156 Apr  9  2019 wget_1.19.4-1ubuntu2.2.debian.tar.xz
-rw-r--r--  1 jjang jjang    2226 Apr  9  2019 wget_1.19.4-1ubuntu2.2.dsc
-rw-r--r--  1 jjang jjang 4310657 Feb  5  2018 wget_1.19.4.orig.tar.gz
-rw-r--r--  1 jjang jjang    1241 Feb  5  2018 wget_1.19.4.orig.tar.gz.asc
~/code-genome ❯ sudo pbuilder --build --distribution focal --architecture amd64 --basetgz /var/cache/pbuilder/focal-amd64-base.tgz --buildresult ~/code-geno
me/amd64 wget_1.19.4-1ubuntu2.2.dsc
sudo: unable to resolve host code-genome
W: Build-result Directory /home/jjang/code-genome/amd64 does not exist
I: pbuilder: network access will be disabled during build
I: Current time: Mon Nov  7 07:46:57 CST 2022
I: pbuilder-time-stamp: 1667828817
I: Building the build Environment
I: extracting base tarball [/var/cache/pbuilder/focal-amd64-base.tgz]
I: copying local configuration
W: No local /etc/mailname to copy, relying on /var/cache/pbuilder/build/40991/etc/mailname to be correct
W: --override-config is not set; not updating apt.conf Read the manpage for details.
I: mounting /proc filesystem
I: mounting /sys filesystem
I: creating /{dev,run}/shm
I: mounting /dev/pts filesystem
I: redirecting /dev/ptmx to /dev/pts/ptmx
I: policy-rc.d already exists
I: created buildresult dir: /home/jjang/code-genome/amd64
I: Obtaining the cached apt archive contents
I: Copying source file
I: copying [wget_1.19.4-1ubuntu2.2.dsc]
I: copying [./wget_1.19.4.orig.tar.gz]
I: copying [./wget_1.19.4.orig.tar.gz.asc]
I: copying [./wget_1.19.4-1ubuntu2.2.debian.tar.xz]
I: Extracting source
gpgv: Signature made Mon Apr  8 19:44:51 2019 UTC
gpgv:                using RSA key 45BCE75B840B1F69
gpgv: Can't check signature: No public key
dpkg-source: warning: failed to verify signature on ./wget_1.19.4-1ubuntu2.2.dsc
dpkg-source: info: extracting wget in wget-1.19.4
dpkg-source: info: unpacking wget_1.19.4.orig.tar.gz
dpkg-source: info: unpacking wget_1.19.4-1ubuntu2.2.debian.tar.xz
dpkg-source: info: using patch list from debian/patches/series
dpkg-source: info: applying wget-doc-remove-usr-local-in-sample.wgetrc
dpkg-source: info: applying wget-doc-remove-usr-local-in-wget.texi
dpkg-source: info: applying wget-passive_ftp-default
dpkg-source: info: applying disable-SSLv3.patch
dpkg-source: info: applying CVE-2018-0494.patch
dpkg-source: info: applying CVE-2018-20483-1.patch
dpkg-source: info: applying CVE-2018-20483-2.patch
dpkg-source: info: applying CVE-2019-5953-1.patch
dpkg-source: info: applying CVE-2019-5953-2.patch
I: using fakeroot in build.
I: Installing the build-deps
I: -> Attempting to satisfy build-dependencies
Note, using file '/build/wget_1.19.4-1ubuntu2.2.dsc' to get the build dependencies
Reading package lists... 10%
```

Compiling *wget* for multiple archs



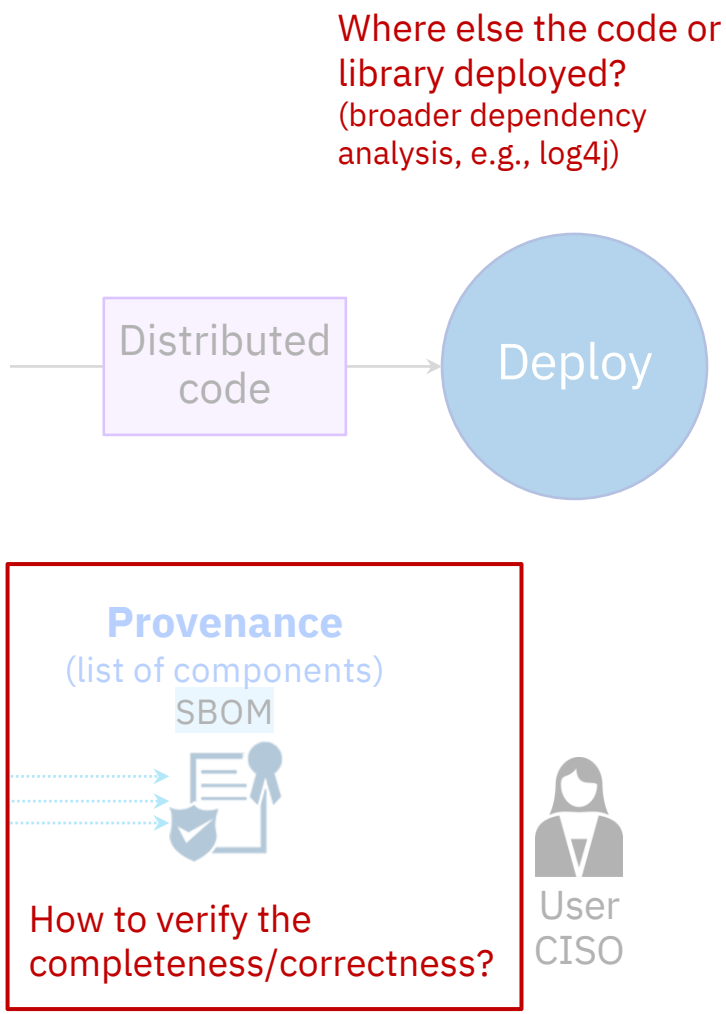
Identifying (unknown) binary

Use Case 1: Finding Log4j
Legacy Software Discovery

- Software deployments are a turducken
  - `zip`, `tar`, container image, `jar`, etc.
  - Dependencies often wrapped up
- No good provenance or CMDB
- Can’t rely on standard directories, filenames, or hashes
- Code can be repackaged
- Applies across the board: CICD, DevSecOps and Legacy

Use Case 2: 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

CycloneDX SBOM Standard

CycloneDX is a modern standard for the software supply chain. SBOM, SaaSBOM, OBOM, Advisories, VEX, and more. CycloneDX is a OWASP Flagship Project.

OWASP https://cyclonedx.org/ @CycloneDX_Spec Verified

Pinned

specification Public
Software Bill of Material (SBOM) standard designed for use in application security contexts and supply chain component analysis
XSLT 168 34

cyclonedx-dotnet Public
Creates CycloneDX Software Bill of Materials (SBOM) from .NET Projects
C# 82 41

cyclonedx-python Public
Creates CycloneDX Software Bill of Materials (SBOM) from Python projects and environments.
Python 91 35

cyclonedx-maven-plugin Public
Creates CycloneDX Software Bill of Materials (SBOM) from Maven projects
Java 125 37

cyclonedx-node-module Public
Creates CycloneDX Software Bill of Materials (SBOM) from Node.js projects
JavaScript 86 67

cyclonedx-cli Public
CycloneDX CLI tool for SBOM analysis, merging, diffs and format conversions.
C# 85 21

This project provides a runnable Python-based application for generating CycloneDX bill-of-material documents from either:

- Your current Python Environment
- Your project's manifest (e.g. `Pipfile.lock`, `poetry.lock` or `requirements.txt`)
- Conda as a Package Manager

"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 > ...

```
FROM ubuntu:focal

RUN apt-get update
RUN apt-get install -y wget

RUN apt-get update
```

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

Dockerfile > ...

```
FROM ubuntu:focal

RUN apt-get update
RUN apt-get install -y wget

RUN mv /var/lib/dpkg/status /var/lib/dpkg/status.bak
RUN touch /var/lib/dpkg/status

RUN apt-get update
```

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

Status and Roadmap

Current Status

- Several techniques for computing genes
- Support for binaries, packages, bytecode
- Scalable, cloud native processing engine
- Currently performing large-scale evaluation

Release coming soon:

- A version of genome generation
- Service demonstrating technology
- Utilities for handling genomes and querying service

Requests:

- Welcome feedback, support and collaboration
- Insights on capabilities and use cases

Questions?