High quality SBOMs for C/C++

Andreas Bielk

2025-03-28

https://sbom.observer/

Funded by

MSB Myndigheten för samhällsskydd och beredskap

NCC-SE NATIONAL CYBERSECURITY COORDINATION CENTRE SWEDEN

The Problem

Compiled languages like C/C++ without a standard package manager means that the tactics used to build SBOMs in other ecosystems doesn't really work

(this includes a lot of important infrastructure today)

The Challenge

- Minimal changes to the existing build system
- Reasonably exact - i.e. we include all dependencies*
- Support custom build configurations (i.e. optional modules)
- Document build tools used

The Opinion

All tools that provide material to build artifacts are potential attack vectors!

Which means that in most cases SBOMs should include (with documented scope):

- Library dependencies
- Standard library & Runtime dependencies
- Compilers & Linkers

Possible solutions

- Manually updated BOM, maybe with the help of some internal tooling
- Retrofit a dependency manager into the project (vcpkg, conan)
- Instrument the test-suite
- Instrument the compiler and linker

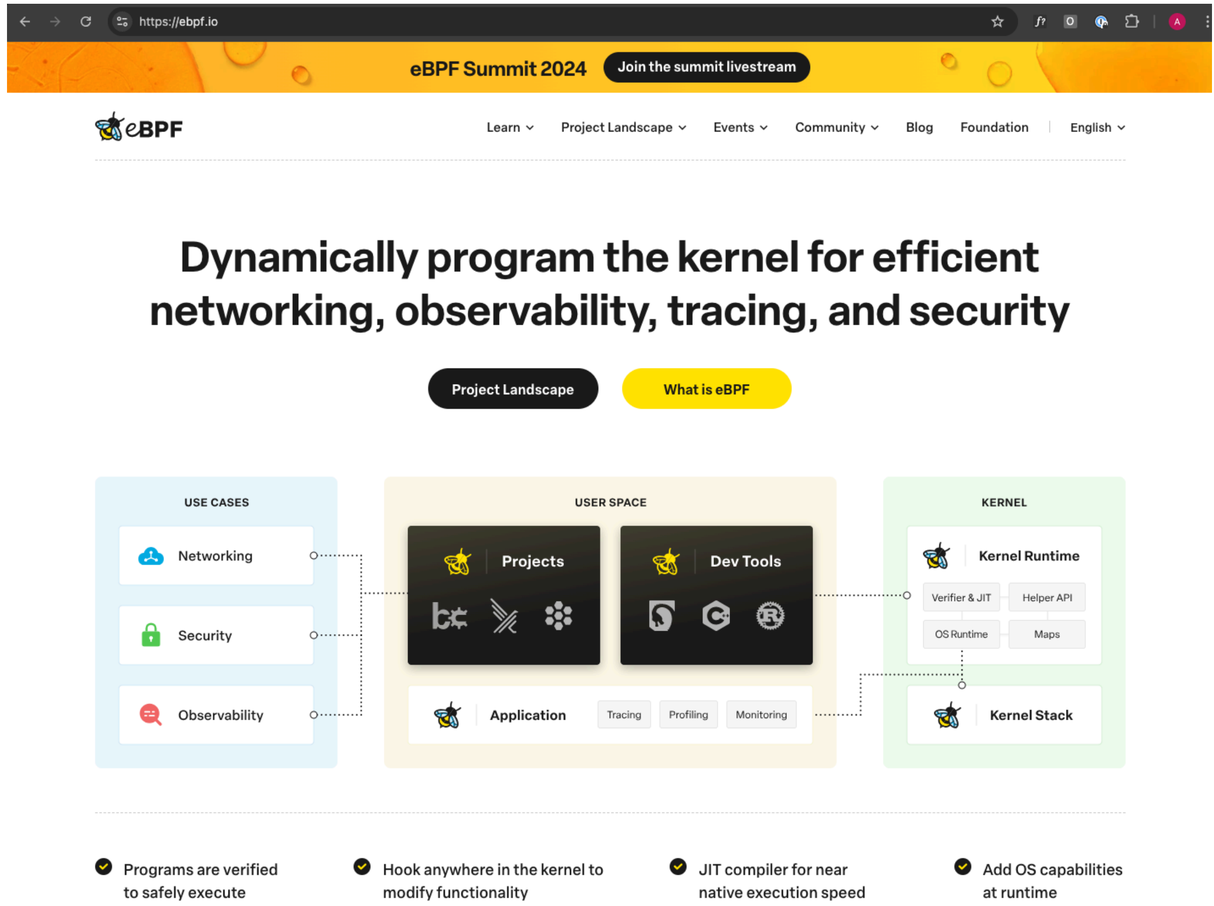
The Plan

- Watch all files the build process opens and executes to complete the build
- Map observed files to dependencies
- Report any files that can’t be mapped
- Create an SBOM

eBPF

"eBPF is a revolutionary technology with origins in the Linux kernel that can run sandboxed programs in a privileged context such as the operating system **kernel**.

It is used to **safely** and **efficiently** extend the capabilities of the kernel without requiring to change kernel source code or load kernel modules."



eBPF is fast

- Instrument kernel to observe all opened and executed files (syscalls)
- 10% build time overhead (~30 sec added for 5 min Asterisk build)

```
$ sudo build-observer --user cicd -- make -f Makefile.linux build-examples
```

build-observer output

```
"start": "2025-03-24T14:08:58.975120568Z",
"workingDirectory": "/usr/local/src/nginx-1.25.3",
 "opened": [
   ...
   "/usr/include/crypt.h",
   "/usr/include/ctype.h",
   ...
   "/usr/include/openssl/asn1.h",
   "/usr/include/openssl/asn1err.h",
   "/usr/include/openssl/async.h",
   "/usr/include/openssl/asyncerr.h",
   "/usr/include/openssl/bio.h",
   "/usr/include/openssl/bioerr.h",
   "/usr/include/openssl/bn.h",
   "/usr/include/openssl/bnerr.h",
  ...
   "/usr/local/src/nginx-1.25.3/src/core/nginx.c",
   "/usr/local/src/nginx-1.25.3/src/core/nginx.h",
   "/usr/local/src/nginx-1.25.3/src/core/ngx_array.c",
   "/usr/local/src/nginx-1.25.3/src/core/ngx_array.h",
   "/usr/local/src/nginx-1.25.3/src/core/ngx_bpf.c",
   "/usr/local/src/nginx-1.25.3/src/core/ngx_bpf.h",
   "/usr/local/src/nginx-1.25.3/src/core/ngx_buf.c",
   ...
]
```

```
...
"executed": [
  "/usr/bin/cc",
  "/usr/bin/ld",
  "/usr/bin/make",
  "/usr/bin/sed",
  "/usr/lib/gcc/x86_64-linux-gnu/12/cc1",
  "/usr/lib/gcc/x86_64-linux-gnu/12/collect2"
  ...
]
```

Creating an SBOM

- We now have a logfile of syscalls (opens, and executions)
- Use some heuristics to prune and deduplicate file dependencies
- Lookup which OS package each file belongs to
  - Optionally track transitive dependencies (depending on use case)
- Lookup any Git repo dependencies
- Create a dependency tree
- Output an SBOM

Example: nginx

- 14 source dependencies
- 3 tool dependencies
- 51 transitive dependencies
- 0 unattributed dependencies

```
~/src/nginx-1.26.0 $  sudo observer build —user cicd —merge nginx.cdx.json  —  make
```

Library Dependency


```
{
  “type": "library",
  "name": “libcodec2-dev",
  "version": "1.0.5-1",
  "purl": “pkg:deb/debian/libcodec2-dev@1.0.5-1?arch=amd64&distro=debian-12.5",
   "licenses": [
    {
      "license": {
        "id": "LGPL-2.1-or-later"
      }
    },
    {
      "license": {
        "id": "BSD-3-Clause"
      }
    }
  ],
},
```

Compiler Dependency


```
{
  "type": "application",
  "scope": “excluded”,
  "name": “cpp-12",
  "version": "12.2.0-14",
  "purl": "pkg:deb/debian/cpp-12@12.2.0-14?arch=amd64&distro=debian-12.5",
  "components": [
    {
      "type": "file",
      "name": "/usr/lib/gcc/x86_64-linux-gnu/12/cc1",
      "hashes": [
        {
          "alg": "SHA-256",
          "content": "a9d27bdb13cfcae82035677c84a28ad3b35fd5e6b3f5e19060d3ba1a69f3c3fe"
        }
      ]
    }
  ]
},
```

Mixed language applications

- Applications built from a mix of languages, like Javascript or Python, mixed with C/C++
- Make sure to observe the complete build, including installing dependencies
- Force recompilation of dependencies (you should already be doing this)
- Merge build-observer SBOM with dependencies for the other ecosystem
- Works surprisingly well!

Example: npm (Javascript)

- 7 (debian) + 59 (npm) source dependencies
- 4 tool dependencies
- 49 transitive dependencies
- 0 unattributed dependencies

```
{
  "name": "npm-with-native-dependencies",
  "version": "1.0.0",

  "dependencies": {
    "leftpad": "^0.0.1",
    "bcrypt": “5.1.1" <— contains C++ cod
  }
}
```

```
$  sudo observer build —user cicd  —  npm install --build-from-source --no-progress
```

Current Status

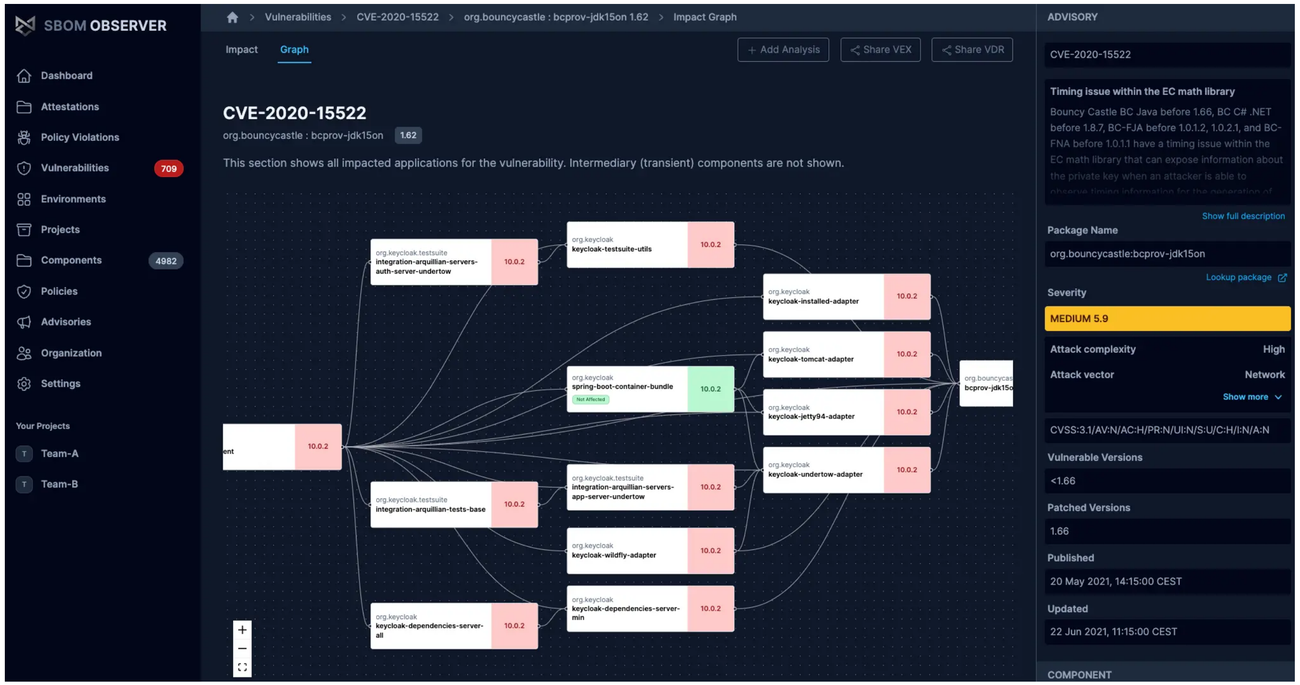
- Production Ready
  - Support for Debian and RPM based build machines
  - Tested on a growing number of real-world projects
- Work in progress
  - Support for Git dependencies
  - Support for vendored dependencies
  - Windows and FreeBSD support

Future work

- Generating Runtime SBOMs using eBPF
  - Dynamic/late linking introduces another supply chain
  - Runtimes (nodejs, python, php etc)
- Runtime SBOM compliance/enforcement
- Capture other assertions about build environment
  - Compiler flags
  - Security related build steps (SAST tools etc)

Resources

- Find me
  - andreas@sbom.observer
  - https://www.linkedin.com/in/andreas-bielk/
- Day job
  - https://sbom.observer/
  - Consulting
- Source Code
  - https://github.com/sbom-observer/build-observer
  - https://github.com/sbom-observer/observer-cli
  - https://github.com/sbom-observer/observer-benchmarks



BYTESAFE

SBOM OBSERVER