

Sharpening Your Tools

Why digital forensics tools require continual updating
November 9, 2023

Simson L. Garfinkel
Chief Scientist, BasisTech

Joint work with
Jon Stewart, Aon Cyber Solutions



practice

Check for updates

DOI:10.1145/3600098

Article development led by queue.acm.org

Updating `bulk_extractor` for the 2020s.

BY SIMSON GARFINKEL AND JON STEWART

Sharpening Your Tools

DIGITAL FORENSICS (DF) is a fast-moving field with a huge subject area. A digital investigator must be able to analyze “any data that might be found on any device anywhere on the planet.”¹² As such, developers must continually update DF tools to address new file formats, new encoding schemes, and new ways that the subjects of investigations use their computers. At the same time, tools must retain the ability to analyze legacy data formats—all of them, in fact.

Most DF tools run on consumer desktop operating systems, adding another layer of complexity: These operating systems are also continually evolving. Analysts must update and upgrade their systems, lest they risk compromise by malware, which decreases productivity and can discredit an analysis in court. This is true even for workstations that are “air gapped” (not connected to the Internet), since malware in evidence can exploit bugs in forensic software.¹⁹

Surprisingly, open source forensic tools distributed as source code face a greater challenge when the underlying operating system is upgraded: Software compatibility layers typically emphasize compatibility for the application binary interface (ABI), not source code. Software compiled from source must cope with upgraded compilers, libraries, and new file locations. As a result, older open source software frequently does not run on modern systems without updating. One way around this problem is to run the old software inside a virtual machine—but older virtual machines won’t be protected against modern malware threats.

One advantage of open source software is the end user has the source code and is therefore able to update the application (or pay for a programmer to update the application). In practice, many users of DF tools lack the expertise, financial resources, and time to update the collection of open source tools they rely upon to do their jobs. Instead, that task falls upon tool developers, who must simultaneously cope with essential changes in DF best practices as well as in operating systems, compilers, and libraries, while avoiding inadvertent changes to important functionality. Developers must also resist the urge for aggressive rewrites that add new expansive functionality, lest they succumb to the “second-system effect.”⁵

This article presents our experience updating the high-performance DF tool BE (`bulk_extractor`)¹⁶ a decade after its initial release. Between 2018 and 2022, we updated the program from C++98 to C++17. We also performed a complete code refactoring and adopted a unit test framework.

The new version typically runs with 75% more throughput than the previous version, attributable to improved multithreading. This article provides lessons and recommendations for other DF tool maintainers. All developers can benefit from the detailed discussion of how embracing features in the C++17 standard and modern software engineering practices can improve the correctness, reliability, and throughput of forensic software. Businesses and funding agencies can use this experience to help justify the substantial cost

44 COMMUNICATIONS OF THE ACM | AUGUST 2023 | VOL. 66 | NO. 8

IMAGE BY ZENITRON

By Simson Garfinkel, Jon Stewart
Communications of the ACM, August 2023, Vol. 66 No. 8, Pages 44-52
10.1145/3600098

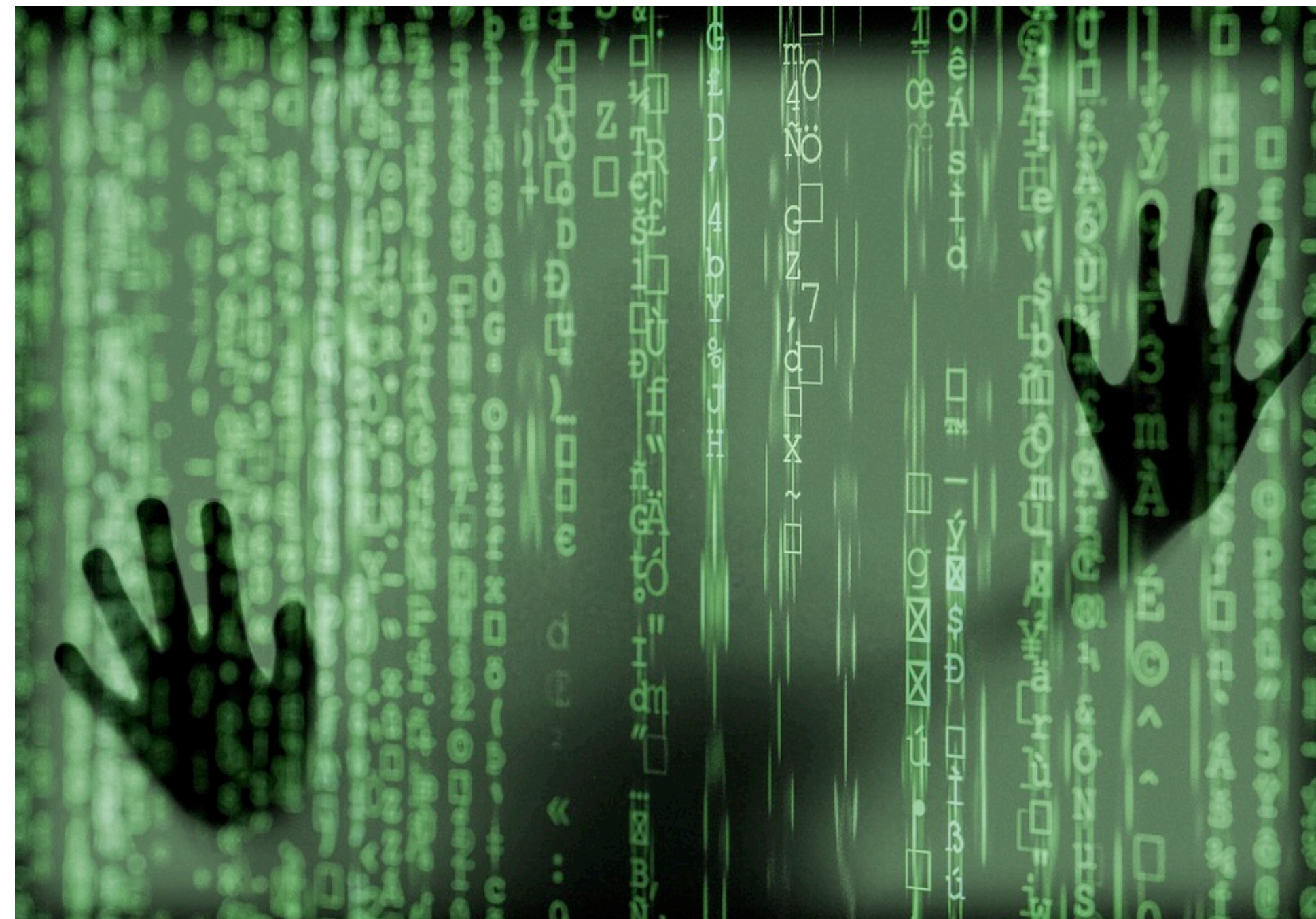
Digital forensics tools require constant maintenance

OS Creep

Language Creep

Forensic Science Creep

O&M (operations & maintenance) “tail”



<https://pixabay.com/illustrations/hacker-computer-ghost-cyber-code-4031973/>

Digital forensics tools require constant maintenance: OS Creep

Platforms being analyzed change over time

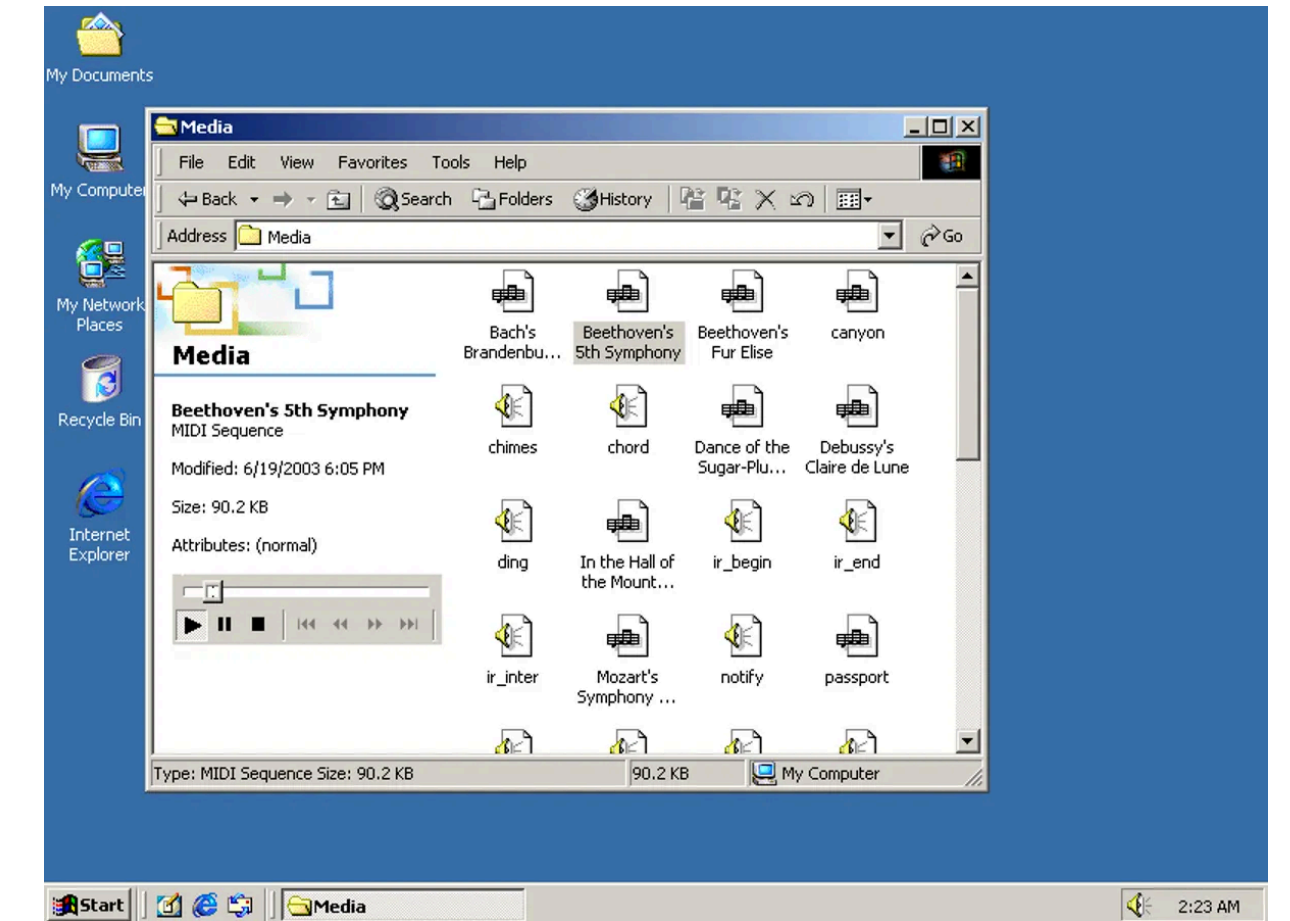
- Windows 7 → Windows 95 → Windows NT → Windows XP → Windows 2000 → Windows 7 → Windows 10 → Windows 11
- Feature Phones → iPhone & Android
- Tablets

Forensics practitioners favor different operating systems over time.

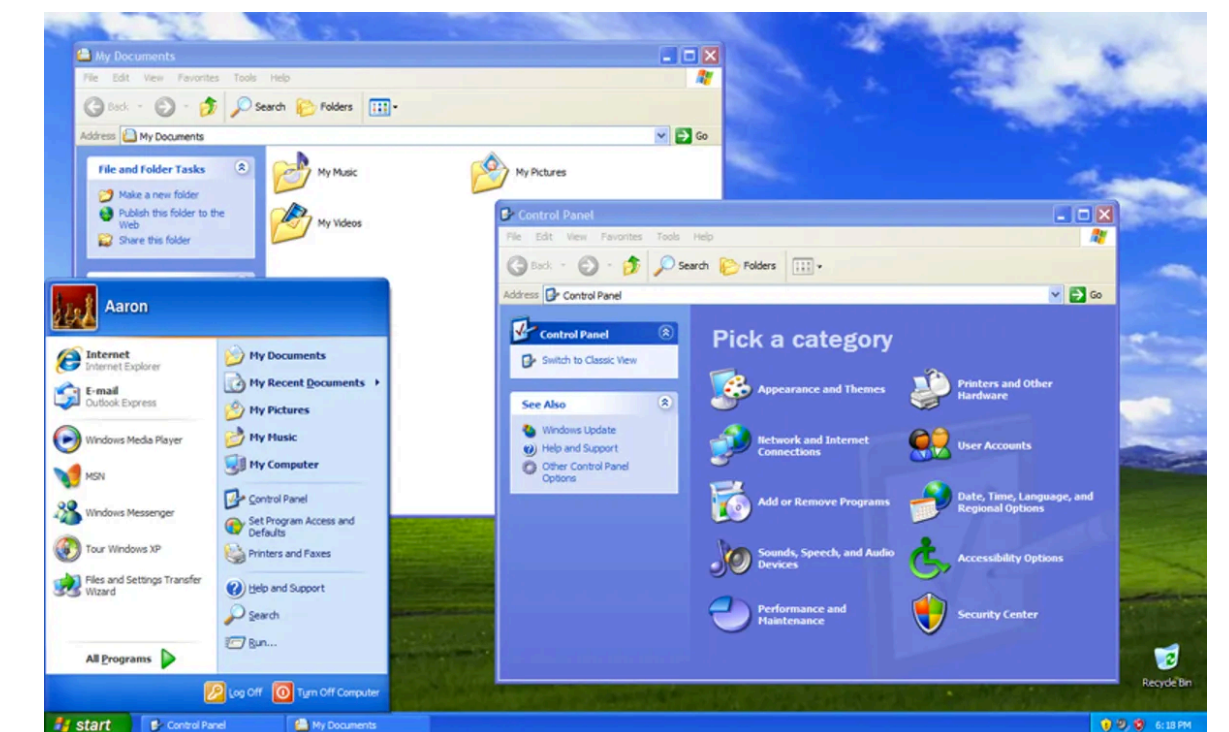
- Linux / Windows / MacOS

OS used for analysis must be upgraded

- Old apps may have bugs or security vulnerabilities
- Old apps may not run on new OS
- New versions of apps may not run on old operating systems



Windows 2000



Windows XP

Digital forensics tools require constant maintenance: Language Creep

Mostly a concern for open-source software

- Open-source software is typically distributed in source-code form
- Operating systems are better at preserving binary compatibility than source-code compatibility
 - ABI (Application Binary Interface) is very stable.
 - High-level languages change – file names change, features are deprecated, etc.
- Example:
 - Java source code from the early 2000s will not compile with a modern Java compiler
 - Java bytecode from the early 2000s will frequently run on a modern JVM
 - Java bytecode & JVM from the early 2000s will almost always run on a modern OS



1996 - 2003



2003 - NOW

Digital forensics tools require constant maintenance: Cybersecurity is constantly changing and improving

DF keeps getting better!

- More complete implementations of today's undocumented data structures
- More reliable, efficient implementations of today's documented data structures.

DF is struggling to keep up!

- Compression standards (e.g. Snappy)
- New memory structures (e.g. Windows 10 memory structures)
- New image formats (e.g. HEIC)

DF software keeps improving

- Usability improvements, support for running in cloud, etc.

Cybersecurity standards keep improving

- New standards for evidence preservation, chain of custody, presentation

Digital forensics tools require constant maintenance: The O&M (operations and maintenance) tail

All software needs to be maintained

DF software is not any different

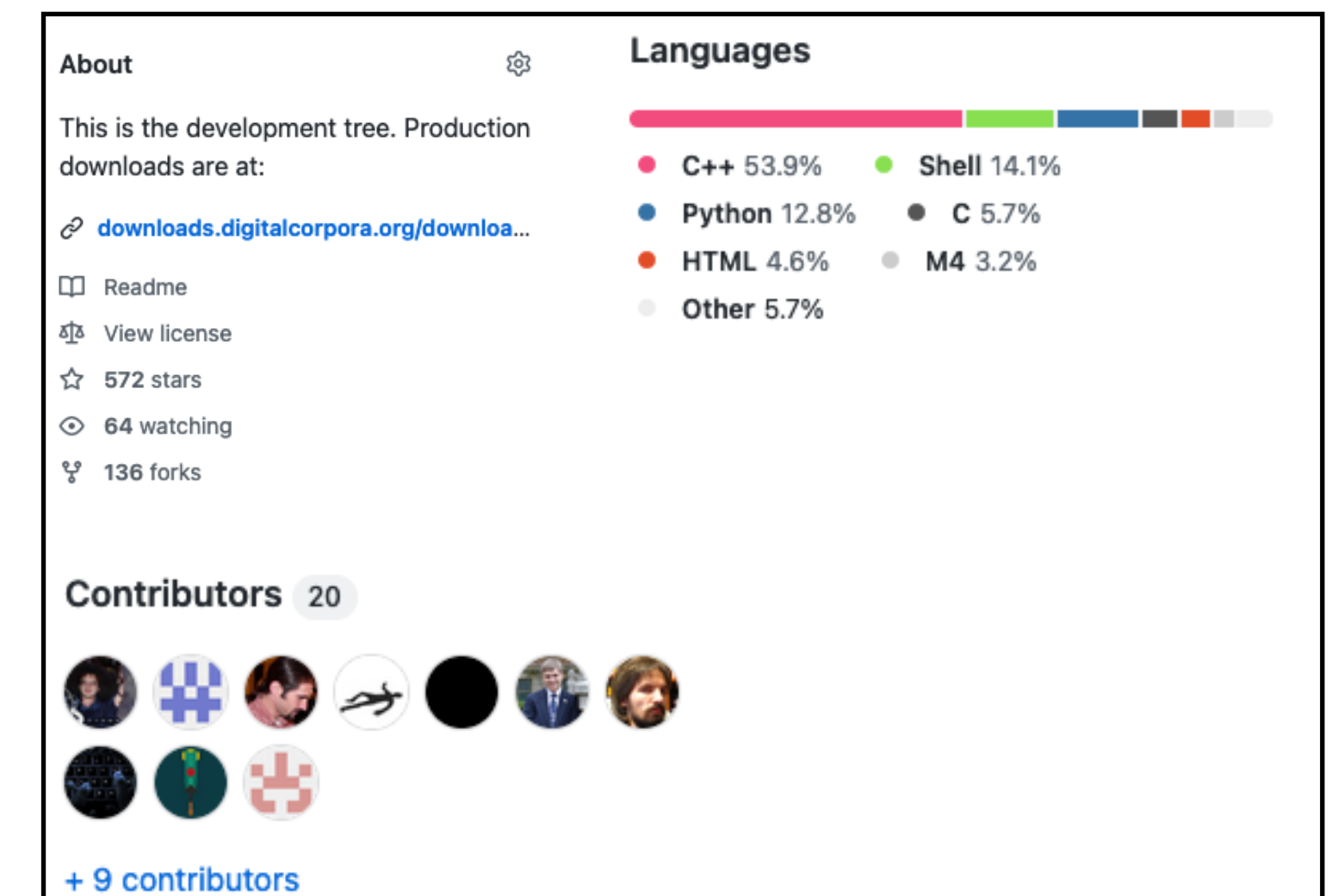
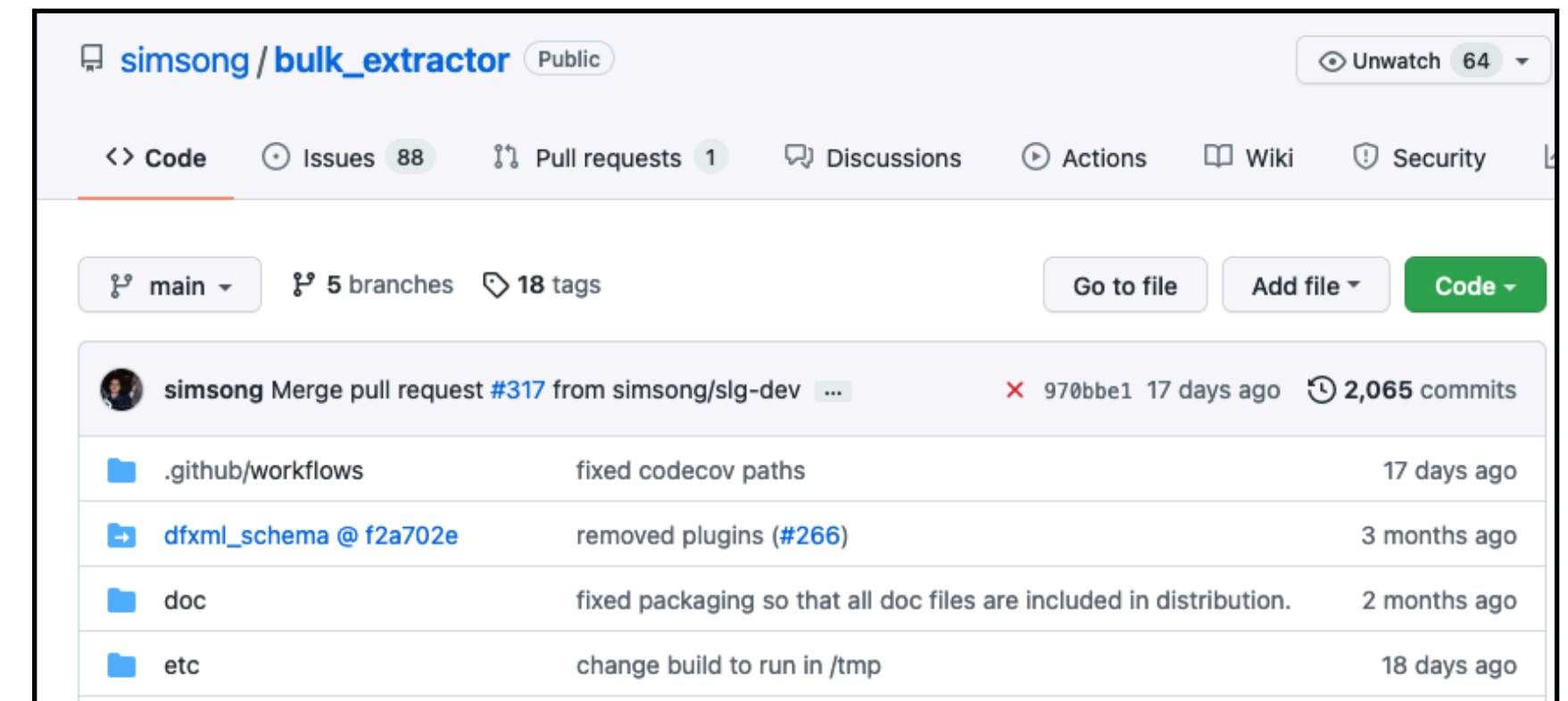
- Bugs reported in software
- Updates to secure hash algorithms (MD5 ✗; SHA-1 ✗; SHA-256 ✓)
- Updates to encryption algorithms (DES3 ✗; AES128 ✗; AES256 ✓)

Case Study: Updating bulk_extractor from 1.6 to 2.0

bulk_extractor:

- Open source DF tool developed between 2003 and 2014
 - *command-line tool: ~ 59K lines of C++98*
 - *GUI: ~ 18K lines java*
 - *Compiled with Autoconf toolchain*
- Runs on macOS, Linux and Windows
- Multi-threaded carving and identity “extraction” tool
- Embedded in at least one commercial product
- User base: research, education, law enforcement, defense

https://github.com/simsong/bulk_extractor



There were many reasons to update bulk_extractor

Maintenance Costs

- Autoconf-based system required modification for major OS releases
 - *BE uses threading, access file systems, etc.*
- bulk_extractor *support* of out-of-date Python versions
 - *caused it to be banned from a Linux release!*

Changes in CPU / IO / memory trade-off

- CPU cores are ~50-100% faster than in 2012
- Laptops and low-end workstations have 2x - 4x as many cores
- High-end servers: 64 cores in 2012; 96 cores in 2020; 224 in 2023
- Memory is 3x faster; everything is SSD → no seek time (seq. access still faster)
- Disk I/O and network drives are faster

Large parts of BE were single-threaded

- BE1 – 1 thread per 16MiB page. “Last page” could take 30-60 min to process
- Histogram processing: batch at the end of page processing, and single-threaded

The most important reason: Correctness

Most computer software implements specifications:

- Formal specifications – RFCs, end-user requirements, etc.
- Informal specifications – What’s in the programmer’s head
- Being able to *read data* written by the *same program*

Many digital forensics tools are based on reverse engineering.

- Read and decode data written by other programs.
- Authors of other programs may be *unknown* or *unwilling* to share technical details.

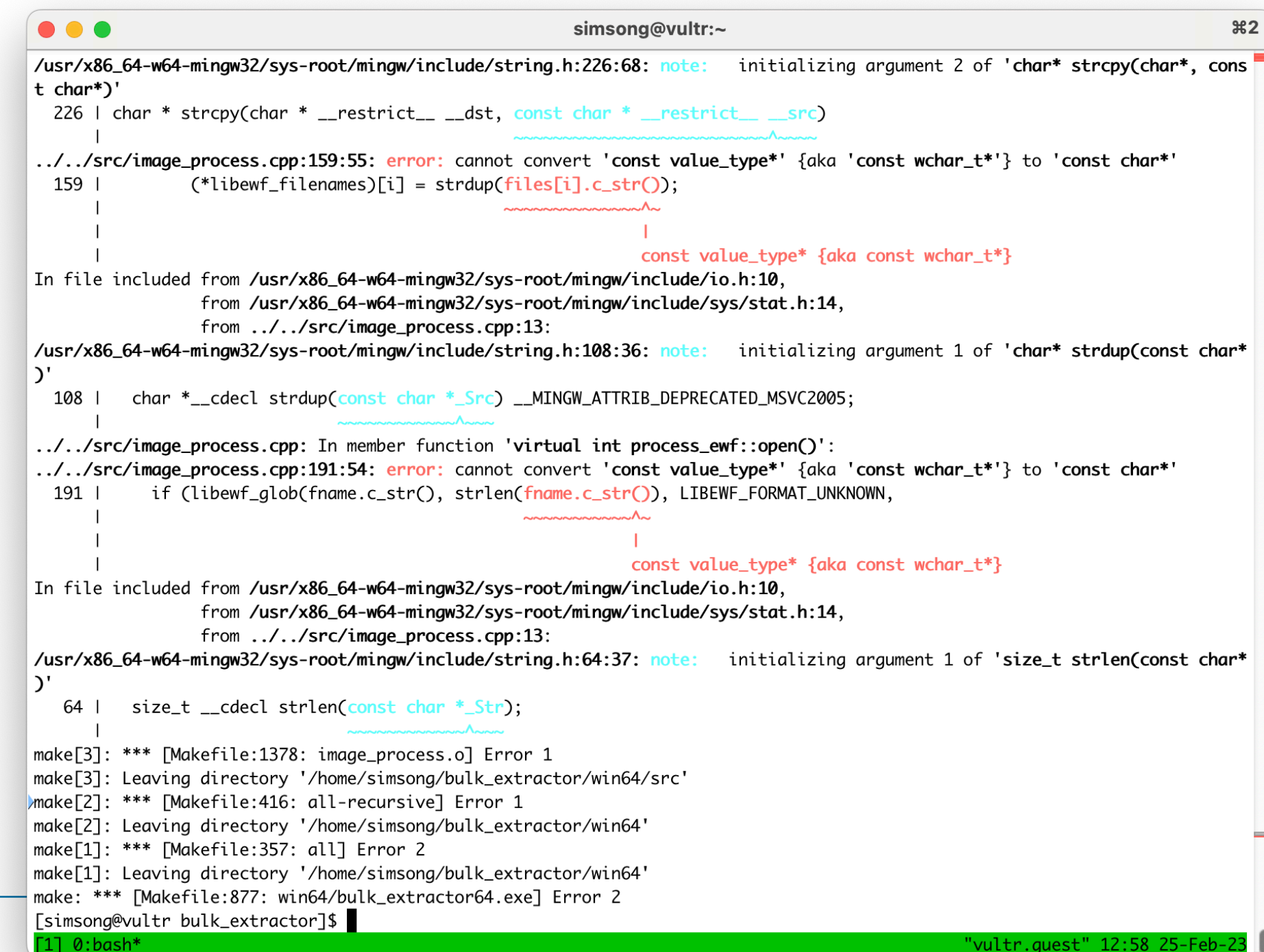
Many digital forensics tools crash or print warnings when they run.

- Bulk_extractor when processing *nps-2009-domexusers.E01*:

```
11:33:51 Offset 486MB (1.13%) Done in 1:57:57 at 13:31:48
11:34:08 Offset 570MB (1.33%) Done in 2:02:19 at 13:36:27
11:34:25 Offset 654MB (1.52%) Done in 2:03:45 at 13:38:10
std::exception Scanner: evtX Exception: Error: Read past end of sbuf sbuf.pos0: (661649934-HIBERFILE|84582400) bufsize=4096
std::exception Scanner: evtX Exception: Error: Read past end of sbuf sbuf.pos0: (721594368-HIBERFILE|44343296) bufsize=4096
std::exception Scanner: evtX Exception: Error: Read past end of sbuf sbuf.pos0: (721594368-HIBERFILE|44351488) bufsize=4096
std::exception Scanner: evtX Exception: Error: Read past end of sbuf sbuf.pos0: (721594368-HIBERFILE|44384256) bufsize=4096
```

Update plan: objectives

1. Make the program easier to compile and maintain
2. Make it easier for others to contribute code
3. Removal experimental code & simplify the codebase
4. Decrease program's runtime



```
simsong@vultr:~
/usr/x86_64-w64-mingw32/sys-root/mingw/include/string.h:226:68: note: initializing argument 2 of 'char* strcpy(char*, const char*)'
 226 | char * strcpy(char * __restrict__ __dst, const char * __restrict__ __src)
     |                                     ~~~~~
../src/image_process.cpp:159:55: error: cannot convert 'const value_type*' {aka 'const wchar_t*'} to 'const char*'
 159 |         (*libewf_filenames)[i] = strdup(files[i].c_str());
     |                                     ~~~~~
     |                                     |
     |                                     const value_type* {aka const wchar_t*}
In file included from /usr/x86_64-w64-mingw32/sys-root/mingw/include/io.h:10,
                 from /usr/x86_64-w64-mingw32/sys-root/mingw/include/sys/stat.h:14,
                 from ../src/image_process.cpp:13:
/usr/x86_64-w64-mingw32/sys-root/mingw/include/string.h:108:36: note: initializing argument 1 of 'char* strdup(const char*)'
 108 | char *__cdecl strdup(const char * __Src) __MINGW_ATTRIB_DEPRECATED_MSVC2005;
     |                             ~~~~~
../src/image_process.cpp: In member function 'virtual int process_ewf::open()':
../src/image_process.cpp:191:54: error: cannot convert 'const value_type*' {aka 'const wchar_t*'} to 'const char*'
 191 |         if (libewf_glob(fname.c_str(), strlen(fname.c_str()), LIBEWF_FORMAT_UNKNOWN,
     |                                     ~~~~~
     |                                     |
     |                                     const value_type* {aka const wchar_t*}
In file included from /usr/x86_64-w64-mingw32/sys-root/mingw/include/io.h:10,
                 from /usr/x86_64-w64-mingw32/sys-root/mingw/include/sys/stat.h:14,
                 from ../src/image_process.cpp:13:
/usr/x86_64-w64-mingw32/sys-root/mingw/include/string.h:64:37: note: initializing argument 1 of 'size_t strlen(const char*)'
   64 | size_t __cdecl strlen(const char * __Str);
     |                             ~~~~~
make[3]: *** [Makefile:1378: image_process.o] Error 1
make[3]: Leaving directory '/home/simsong/bulk_extractor/win64/src'
make[2]: *** [Makefile:416: all-recursive] Error 1
make[2]: Leaving directory '/home/simsong/bulk_extractor/win64'
make[1]: *** [Makefile:357: all] Error 2
make[1]: Leaving directory '/home/simsong/bulk_extractor/win64'
make: *** [Makefile:877: win64/bulk_extractor64.exe] Error 2
[simsong@vultr bulk_extractor]$
```


Goal: BE easier to compile and maintain

Approach: Adopting C++17

Autoconf checks for differences between OS.

- Can only check for what it knows!
- Creates `#define` statement that need to be handled in your code with `#ifdef`

C++11, C++14, C++17 standards

- Compiler flag to indicate which standard you want
- A standard set of `#include` files specified by the standard
- C++14 adds multi-threading → removed `#ifdefs` for POSIX and Windows threads!
- C++17 adds file system operations → removed `#ifdefs`, code for dir recursion, etc.

Be sure to check C++ compiler and library support!

- https://en.cppreference.com/w/cpp/compiler_support

C++17 core language features

C++17 feature	Paper(s)	GCC	Clang	MSVC	Apple Clang	EDG ecpp	Intel C++	IBM XL C++	Sun/Oracle C++	Embarcadero C++ Builder	Cray	Nvidia HPC C++ (ex Portland Group/PGI)	Nvidia nvcc	[Collapse]
New auto rules for direct-list-initialization	N3922	5	3.8	19.0 (2015)*	Yes	4.10.1	17.0			10.3		17.7	11.0	
<code>static_assert</code> with no message	N3928	6	2.5	19.10*	Yes	4.12	18.0			10.3		17.7	11.0	
typename in a template parameter	N4051	5	3.5	19.0 (2015)*	Yes	4.10.1	17.0			10.3		17.7	Yes*	
Removing trigraphs	N4086	5	3.5	16.0*	Yes	5.0				10.3		19.1	11.0	
Nested namespace				19.0										

C++20 core language features

C++20 feature	Paper(s)	GCC	Clang	MSVC	Apple Clang	EDG ecpp	Intel C++	IBM XL C++	Sun/Oracle C++	Embarcadero C++ Builder	Cray	Nvidia HPC C++ (ex Portland Group/PGI)	Nvidia nvcc	[Collapse]
Allow <code>lambda-capture [=, this]</code>	P0409R2	8	6	19.22*	10.0.0*	5.1							20.7	
<code>_VA_OPT_</code>	P0306R4 P1042R1	8 (partial)* 10 (partial)* 12	9	19.25*	11.0.3*	5.1							20.7	
Designated initializers	P0329R4	4.7 (partial)* 8	3.0 (partial)* 10	19.21*	(partial)*	5.1							20.7	
template-parameter-list for generic lambdas	P0428R2	8	9	19.22*	11.0.0*	5.1							20.7	
Default member initializers for <code>struct</code>	P0683R1	8	6	19.25*	10.0.0*	5.1							20.7	

Goal: Improve reliability and make it easier for others to contribute code; Approach: continuous integration

BE 1.6: No formal or ongoing testing; occasional End-to-End Testing

- Run the program and see if output looks right.
- (Common in digital forensics tools.)

BE 2.0: Systematic testing

- Unit tests & end-to-end regression tests.
- All automated as part of development and build process.
- Implemented with C++ test framework (Catch2)

Using C++ test framework

- Enable compiler instrumentation:
 - Record test coverage*
-fprofile-arcs -ftest-coverage
 - AddressSanitizer to catch invalid/illegal memory references*
-fsanitize=address -fsanitize-address-use-after-scope
 - ThreadSanitizer to address multithreading issues*
-fsanitize=thread

Automating Tests - Unit Tests

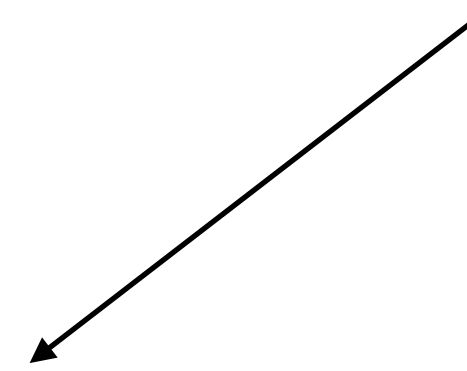
C++ instrumentation

- Unit tests for *every* forensic function
- Frequently required restructuring code

Example: Base64 identification

```
49  const std::string JSON2 {"[{\"1\": \"one@base64.com\"}, {\"2\": \"two@base64.com\"}, {\"3\": \"three@base64.com\"}]\n"};
182  TEST_CASE("scan_base64_functions", "[support]" ){
183      base64array_initialize();
184      sbuf_t sbuf1("W3siMSI6ICJvbmVAYmFzZTY0LmNvbSJ9LCB7IjIiOiAidHdvQGJhc2U2NC5jb20i");
185      bool found_equal = false;
186      REQUIRE(sbuf_line_is_base64(sbuf1, 0, sbuf1.bufsize, found_equal) == true);
187      REQUIRE(found_equal == false);
188
189      sbuf_t sbuf2("W3siMSI6ICJvbmVAYmFzZTY0LmNvbSJ9LCB7IjIiOiAidHdvQGJhc2U2NC5jb20i\n"
190                 "fSwgeyIzIjogInRocmVlQGJhc2U2NC5jb20ifV0K");
191      REQUIRE(sbuf_line_is_base64(sbuf2, 0, sbuf1.bufsize, found_equal) == true);
192      REQUIRE(found_equal == false);
193
194      sbuf_t *sbuf3 = decode_base64(sbuf2, 0, sbuf2.bufsize);
195      REQUIRE(sbuf3 != nullptr);
196      REQUIRE(sbuf3->bufsize == 78);
197      REQUIRE(sbuf3->asString() == JSON2);
198      delete sbuf3;
199  }
```

Expected result



1. Verify BASE64 recognition

2. Verify BASE64 recognition among other data.

3. Verify BASE64 properly decoded

Automating Tests - End-to-End tests

Uses the same C++ instrumentation!

- Refactored main(argv, argc) so that is now called bulk_extractor(argv, argc)
- main() calls bulk_extractor()
- Unit tests can repeatedly call bulk_extractor() with different arguments.

Advantages:

- Test program sets up runtime environment, calls bulk_extractor(), and validates results.
- Makes it easier to catch errors involving resource management (e.g. memory, file descriptors).
- Makes it possible to validate processing of command-line parameters.
- Makes it possible to validate program restart logic.

```
94 TEST_CASE("e2e-h", "[end-to-end]") {
95     /* Try the -h option */
96     const char *argv[] = {"bulk_extractor", "-h", nullptr};
97     std::stringstream ss;
98     int ret = run_be(ss, argv);
99     REQUIRE( ret==1 );           // -h now produces 1
100 }
---
```

Update plan: objectives

1. Make the program easier to compile and maintain ✓
2. Make it easier for others to contribute code
 - Use Git “modules” for increased separation between components
 - Use GitHub “Actions” for continuous integration tests on every commit & pull request
 - Display code coverage results of unit tests
3. Removal experimental code & simplify the codebase
4. Decrease program’s runtime

Split projects up into modules for improved maintainability.

bulk_extractor 1.0 consists of three git modules:

- [github.com://simsong/bulk_extractor.git](https://github.com/simsong/bulk_extractor.git) –CLI, GUI, data reader, scanners
- [github.com://simsong/be13_api.git](https://github.com/simsong/be13_api.git) – Framework for scanner set, feature recorders
- [github.com://simsong/dfxml.git](https://github.com/simsong/dfxml.git) – Digital Forensics XML writer.

For bulk_extractor 2.0:

- [github.com://simsong/bulk_extractor.git](https://github.com/simsong/bulk_extractor.git)
- [github.com://simsong/be13_api.git](https://github.com/simsong/be13_api.git)
- https://github.com/dfxml-working-group/dfxml_cpp
 - Created a GitHub “organization.”
 - Separated DFXML C++ tools from DFXML Python tools
- <https://github.com/simsong/BEViewer/>
 - Java GUI is now a separate module (simsong/bulk_extractor is a sub-module)
 - Allows significant updates to C++ application without impact on Java GUI

GitHub Actions to combine unit tests with code coverage tools

The screenshot shows the GitHub Actions interface for the repository 'simsong/be13_api'. The 'All workflows' section is active, displaying a list of workflow runs. The runs are filtered by 'All workflows' and show a mix of successful and failed runs. The failed runs are specifically for the workflow 'BE13_API CI (c++17) on FreeBSD' on the 'FreeBSD' branch. The successful runs are for the workflow 'BE13_API CI (c++17)' on the 'main' branch. The workflow runs are listed in a table with columns for Event, Status, Branch, and Actor.

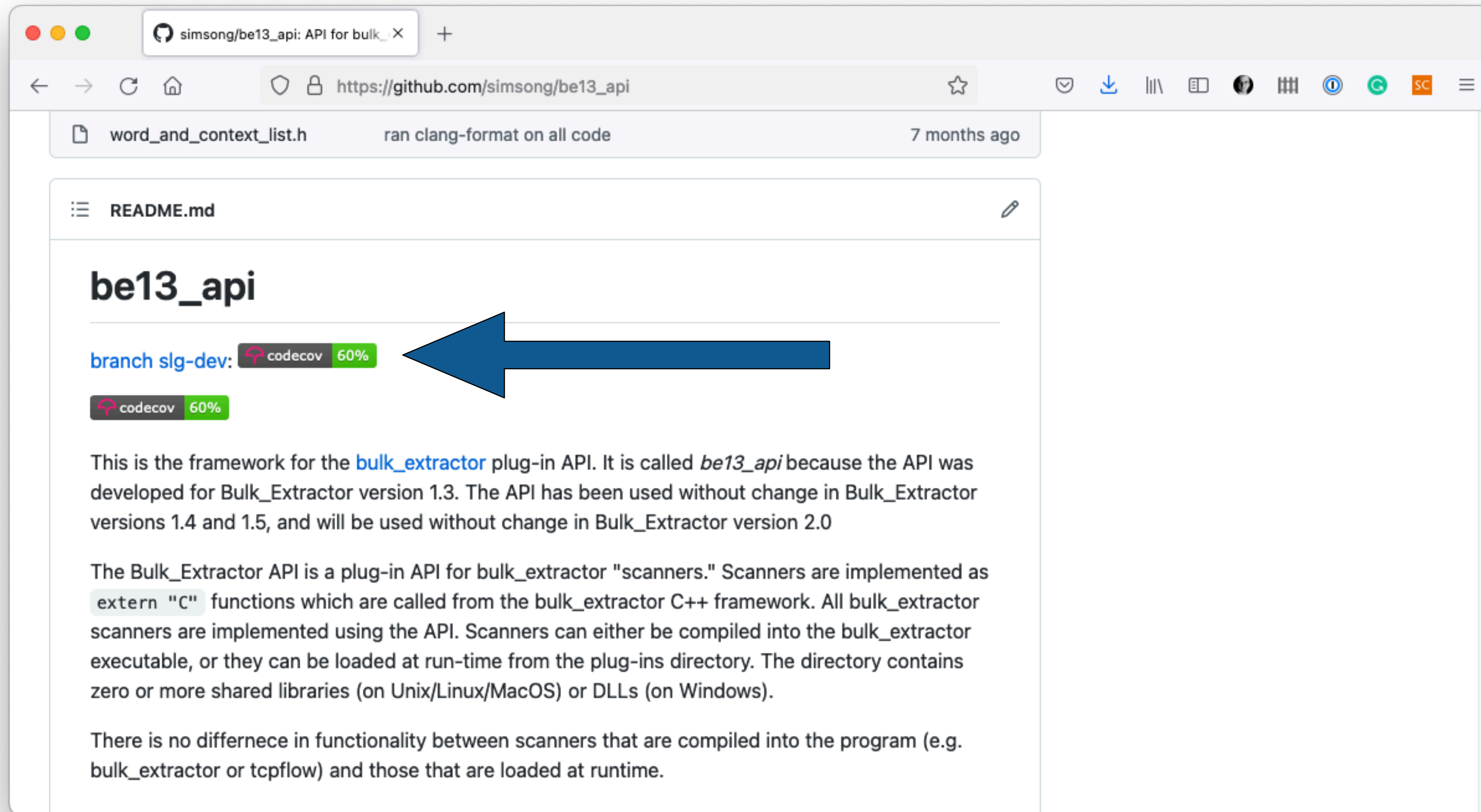
Event	Status	Branch	Actor
BE13_API CI (c++17) on FreeBSD #59: Scheduled	Failed	FreeBSD	
BE13_API CI (c++17) #289: by simsong	Success	aws-linux	
BE13_API CI (c++17) on FreeBSD #58: Scheduled	Failed	FreeBSD	
BE13_API CI (c++17) #288: Commit 4aa30e5 pushed by simsong	Success	main	
BE13_API CI (c++17) #287: by simsong	Failed	slg-dev	
BE13_API CI (c++17) #286: Commit 0e5079a pushed by simsong	Success	main	
BE13_API CI (c++17) #285: Commit 41e1f53 pushed by simsong	Success	slg-dev	
BE13_API CI (c++17) #284: Commit 8da6e24 pushed by simsong	Failed	slg-dev	
typ9o	Success		

Tests on FreeBSD failing

Branch being tested

Commit to "main" did not cause any problems!

“codecov” tool integrates with GitHub Actions





The screenshot shows a web browser window displaying the GitHub repository page for 'simsong/be13_api'. The browser's address bar shows the URL 'https://github.com/simsong/be13_api'. The repository name 'simsong/be13_api: API for bulk_...' is visible in the top left. The main content area shows the README file for 'be13_api'. A blue arrow points to the Codecov coverage badge for the 'branch slg-dev: codecov 60%'. Below the badge, there is another Codecov badge showing '60%'. The README text describes the framework for the 'bulk_extractor' plug-in API, its development history, and its integration with the Bulk_Extractor API.

word_and_context_list.h ran clang-format on all code 7 months ago

README.md

be13_api

branch slg-dev:  ←



This is the framework for the [bulk_extractor](#) plug-in API. It is called *be13_api* because the API was developed for Bulk_Extractor version 1.3. The API has been used without change in Bulk_Extractor versions 1.4 and 1.5, and will be used without change in Bulk_Extractor version 2.0

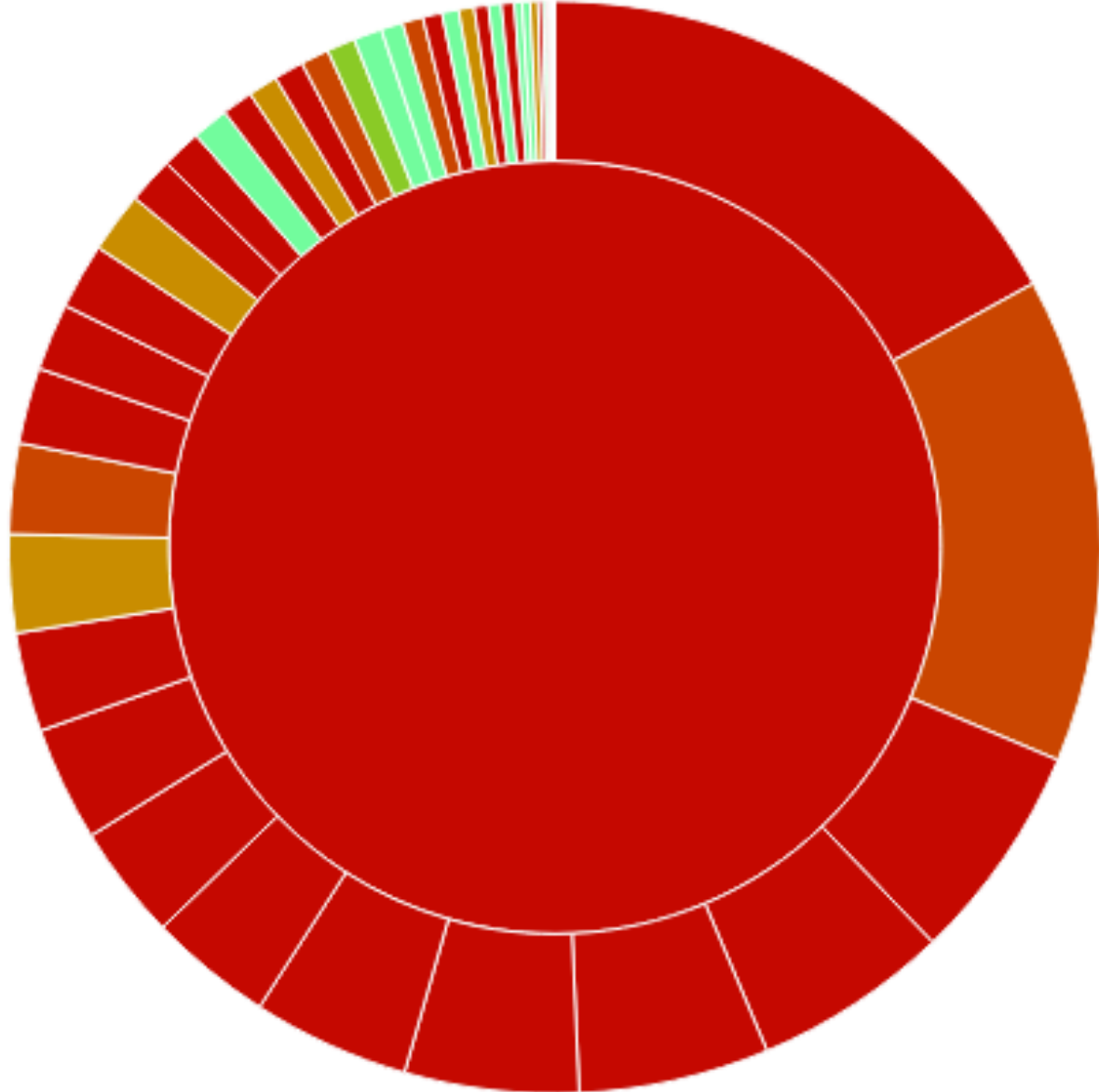
The Bulk_Extractor API is a plug-in API for bulk_extractor "scanners." Scanners are implemented as extern "C" functions which are called from the bulk_extractor C++ framework. All bulk_extractor scanners are implemented using the API. Scanners can either be compiled into the bulk_extractor executable, or they can be loaded at run-time from the plug-ins directory. The directory contains zero or more shared libraries (on Unix/Linux/MacOS) or DLLs (on Windows).

There is no difference in functionality between scanners that are compiled into the program (e.g. bulk_extractor or tcpflow) and those that are loaded at runtime.

Codecov

https://app.codecov.io/gh/simsong/be13_api

COVERAGE SUNBURST



The sunburst chart displays code coverage for the file `regex_vector.cpp`. The inner ring is almost entirely red, indicating high coverage. The outer ring shows some variation in colors, including yellow, orange, and green, representing different levels of coverage across various code blocks. A small portion of the chart is colored light green, indicating 100% coverage.

[/ regex_vector.cpp](#)

RECENT COMMITS

- Merge remote-tracking branch 'origin/aws-linux' into HE**
simsong 8 days ago `aws-linux` `fb622d1`
✓ CI Passed
- fixed typos**
simsong 9 days ago `main` `4aa30e5`
✓ CI Passed
- Slg dev (#76)**
simsong 9 days ago `main` `0e5079a`
✓ CI Passed
- merged in conflict**
simsong 9 days ago `slg-dev` `41e1f53`
✓ CI Passed
- typ9o**
simsong 10 days ago `aws-linux` `827643e`

Codecov

https://app.codecov.io/gh/simsong/be13_api

Files	≡	●	●	●	Coverage
scanner_set.cpp	445	267	0	178	60.00%
pcap_fake.cpp	89	0	0	89	0.00%
path_printer.cpp	170	83	0	87	48.82%
sbuf.cpp	377	292	0	85	77.45%
sbuf_stream.cpp	122	48	0	74	39.34%
unicode_escape.cpp	135	63	0	72	46.67%
feature_recorder_file.cpp	152	92	0	60	60.53%
word_and_context_list.cpp	59	0	0	59	0.00%
feature_recorder.cpp	148	102	0	46	68.92%
sbuf.h	77	40	0	37	51.95%
threadpool.cpp	96	65	0	31	67.71%
feature_recorder_set.cpp	89	58	0	31	65.17%
utils.cpp	52	27	0	25	51.92%
histogram_def.h	51	30	0	21	58.82%
scanner_params.cpp	24	5	0	19	20.83%
regex_vector.cpp	31	14	0	17	45.16%
atomic_map.h	70	54	0	16	77.14%
histogram_def.cpp	36	20	0	16	55.56%
word_and_context_list.h	15	0	0	15	0.00%

Sorted by lines not covered

```
Codecov x +
https://app.codecov.io/gh/simsong/be13_api/blob/main/scanner_set.cpp
726 /**
727  * Records when each sbuf starts. Used for restarting and graphing CPU utilization during run.
728  */
729 void scanner_set::record_work_start(const sbuf_t *sbufp)
730 {
731     if (sbufp->depth()==0 && writer) {
732         writer->xmlout("debug:work_start","",
733             Formatter()
734                 << "threadid='" << std::this_thread::get_id() << "' "
735                 << " pos0='" << dFXML_writer::xmlescape(sbufp->pos0.str()) << "' "
736                 << " pagesize='" << sbufp->pagesize << "' "
737                 << " bufsize='" << sbufp->bufsize << "' "
738                 << aftimer::now_str(" t='", "'"), true);
739     }
740 }
741
742 void scanner_set::record_work_start_pos0str(const std::string pos0str)
743 {
744     if (writer) {
745         writer->xmlout("debug:work_start","",
746             Formatter() << "pos0='" << dFXML_writer::xmlescape(pos0str) << "'", true);
747     }
748 }
749
750
751 void scanner_set::record_work_end(const sbuf_t *sbufp)
752 {
753     if (debug_flags.debug_benchmark && sbufp->depth()==0 && writer) {
754         writer->xmlout("debug:work_end", "",
755             Formatter()
756                 << "threadid='" << std::this_thread::get_id() << "' "
757                 << "pos0='" << dFXML_writer::xmlescape(sbufp->pos0.str()) << "' "
758                 << "rc='" << sbufp->reference_count << "' "
759                 << aftimer::now_str(" t='", "'"), true);
760     }
761 }
762
763
764 /*****
765 ** sbuf processing
```

Completely Covered

Not covered

Partially Covered

Update plan: objectives

1. Make the program easier to compile and maintain ✓
2. Make it easier for others to contribute code ✓
3. Removal experimental code & simplify the codebase
 - Experimental code:
 - Removed bulk_extractor scanners written for specific research projects*
 - Simplify codebase:
 - Moved more functionality from bulk_extractor.git to be13_api.git*
 - Removed features that were not widely used (e.g. writing to SQLite3)*
 - Removed support for obsolete operating systems*
4. Decrease program's runtime

Update plan: objectives

Make the program easier to compile and maintain ✓

Make it easier for others to contribute code ✓

Remove experimental code & simplify the codebase ✓

Decrease program runtime – a difficult goal!

- Run time depends on what's being analyzed
 - Run time increases when more scanners are activated
 - Run time decreases when scanners decide not to analyze something
- Redesign internals to make it easier to measure:
 - CPU time spent in each scanner (vs. recursively called scanners)
 - CPU time spent at top-level analysis (vs. recursive analysis)
 - CPU time spent analyzing new data
- Better reporting of runtime:
 - Systematically capture runtime information in DFXML
- Refactoring measurement system led to more efficient analysis
 - Measuring “time spent analyzing new data” → “only analyze new data” scanner flag.
 - Moved speedups for individual scanners into architecture

Results: BE1 vs. BE2

Size

Compile-time (relevant for development)

Runtime

Analysis

BE1 vs. BE2: BE2 is a lot smaller

Size

	BE1 files	BE2 files	BE1 lines	BE2 lines
C++ Code	274	221	191,779	178,848
Java Code	88	0	17,933	0

Compile-time (relevant for development)

Runtime

Analysis

BE1 vs. BE2: BE2 compiles faster

Size ✓

Compile-time (relevant for development)

	BE1 Mac mini 2018	BE2 Mac mini 2018	Reason
configure	25 sec	16 sec	less probing
make -j1	115 sec	121 sec	Slightly harder C++ compiles
make -j12	32 sec	32 sec	parallelism!

Runtime
Analysis

BE1 vs. BE2: BE2 ran a lot slower ... but got getting faster

Built-in microbenchmarks allowed optimizing sections that mattered.

TABLE 1: **CLOCK TIME COMPARISON OF RUNNING BE1.6 AND BE2**

COMPUTER	DISK IMAGE (+ CONFIG)	SCANNERS				
		29			30 + AES192	
		BE1.6	BE2	THROUGHPUT	BE2	THROUGHPUT
MACBOOK PRO (RETINA, 13 INCH, LATE 2013)*	nps-2009-ubnist1	140s	109s	128%	120s	117%
	nps-2009-domexusers	1420s	837s	170%	1208s	118%
MAC MINI (2018)**	nps-2009-ubnist1	43s	35s	123%	33s	130%
	nps-2009-domexusers	428s	319s	134%	428s	100%
MACBOOK PRO (16-INCH, 2021)†	nps-2009-ubnist1	20s	16s	125%	17s	118%
	nps-2009-domexusers	221s	126s	175%	172s	128%
	nps-2013-2tb	20142s	10944s	184%	11184s	180%

* 2.8GHz Dual-core Intel Core i7; 16GB, 1600MHz DDR3; two physical cores (four with hyperthreading); macOS 11.6.3

** 3GHz 6-core i5; 2667 MHz DDR4; macOS 12.1

† Apple M1 Pro 10 core; 32GB RAM; macOS 12.1

BE1 vs. BE2: BE2 is finding a lot of stuff that BE1 missed

Size ✓

Compile-time (relevant for development) ✓

Runtime ✓

Analysis

file	BE16	BE2.0 Beta 4
alerts.txt	62	19
domain.txt	72,027	76,800
email.txt	8,757	8,751
ether.txt	5	1
ether_histogram_1.txt	n/a	0
exif.txt	232	235
facebook.txt	n/a	0
ip.txt	4	4,444
jpeg_carved.txt	43	1,767
json.txt	4	958
kml.txt	0	2
ntfsusn_carved.txt	2	1
rfc822.txt	4,240	4,219
tcp.txt	n/a	56
tcp_histogram.txt	n/a	0
telephone.txt	767	760
unzip_carved.txt	41	n/a
url.txt	108,352	112,754
winpe.txt	10,740	10,592
winpe_carved.txt	4	10,573
winprefetch.txt	124	0
zip.txt	5,196	10,193

1,724 additional JPEGs carved

10,569 windows executables carved!

Conclusion:

What this means for digital forensics tools

New releases:

- Should be validated against previous releases in a systemic manner
- Results should be published in a machine-readable form.
- Clearly document:
 - New data that is recovered from legacy datasets (compared to previous version)
 - Data recovered from new datasets that previous version would miss
 - Overcollection that has been eliminated

We need to set expectations for DF tools

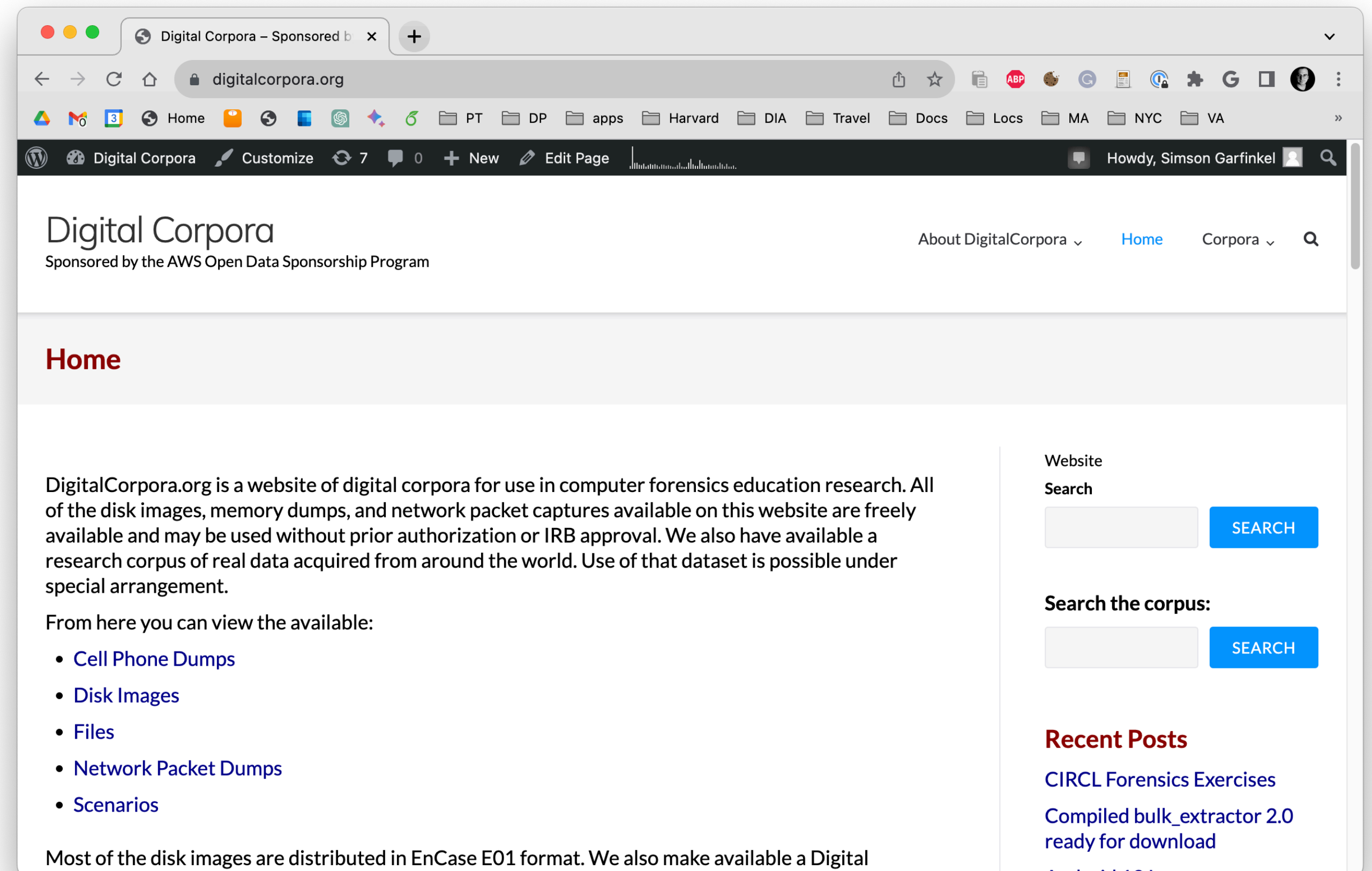
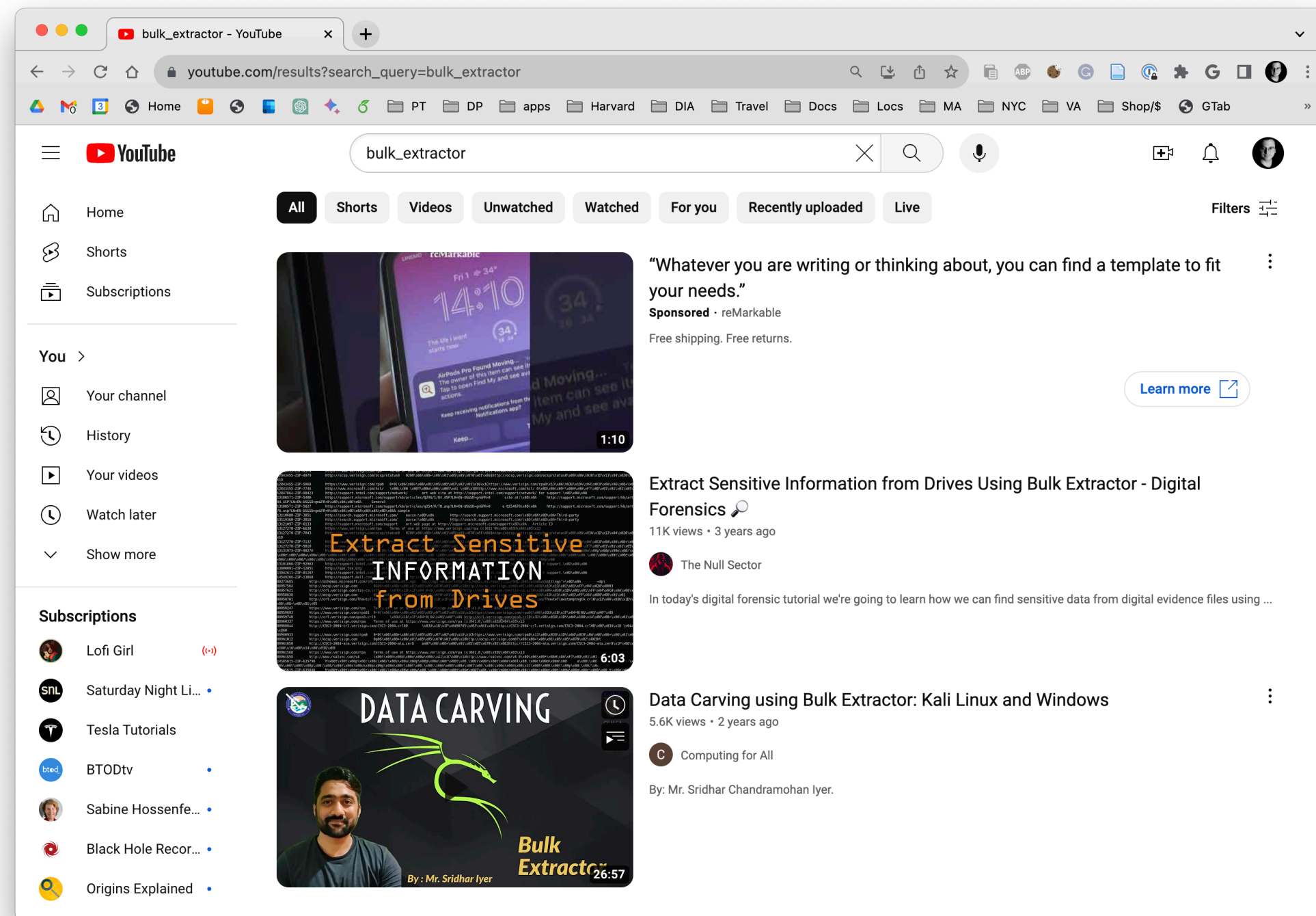
- Complete rewrites are slow
 - 10 years to get from “Ethereal” to Wireshark 1.0 in 2008, 2.0 in 2015
 - Volatility 2: 2.5 - October 2015; 2.6 - December 2016
 - Volatility 3: v1.0.0 - Feb 01, 2021; v 1.0.1 - Feb 1, 2021

Unclear how to measure proprietary tools

How you can help!

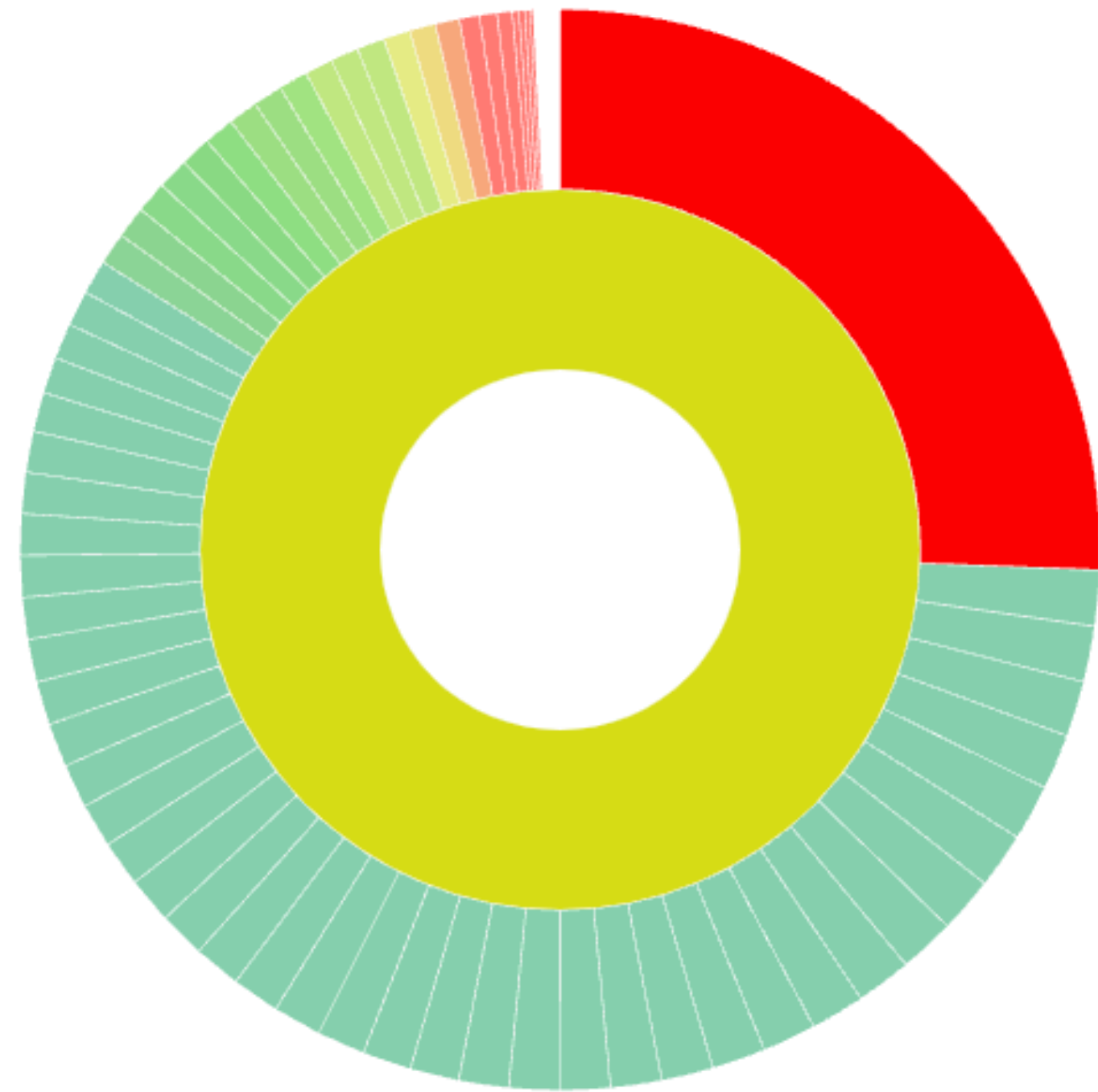
You can make bulk_extractor better!

You can make a tutorial and put it on YouTube!

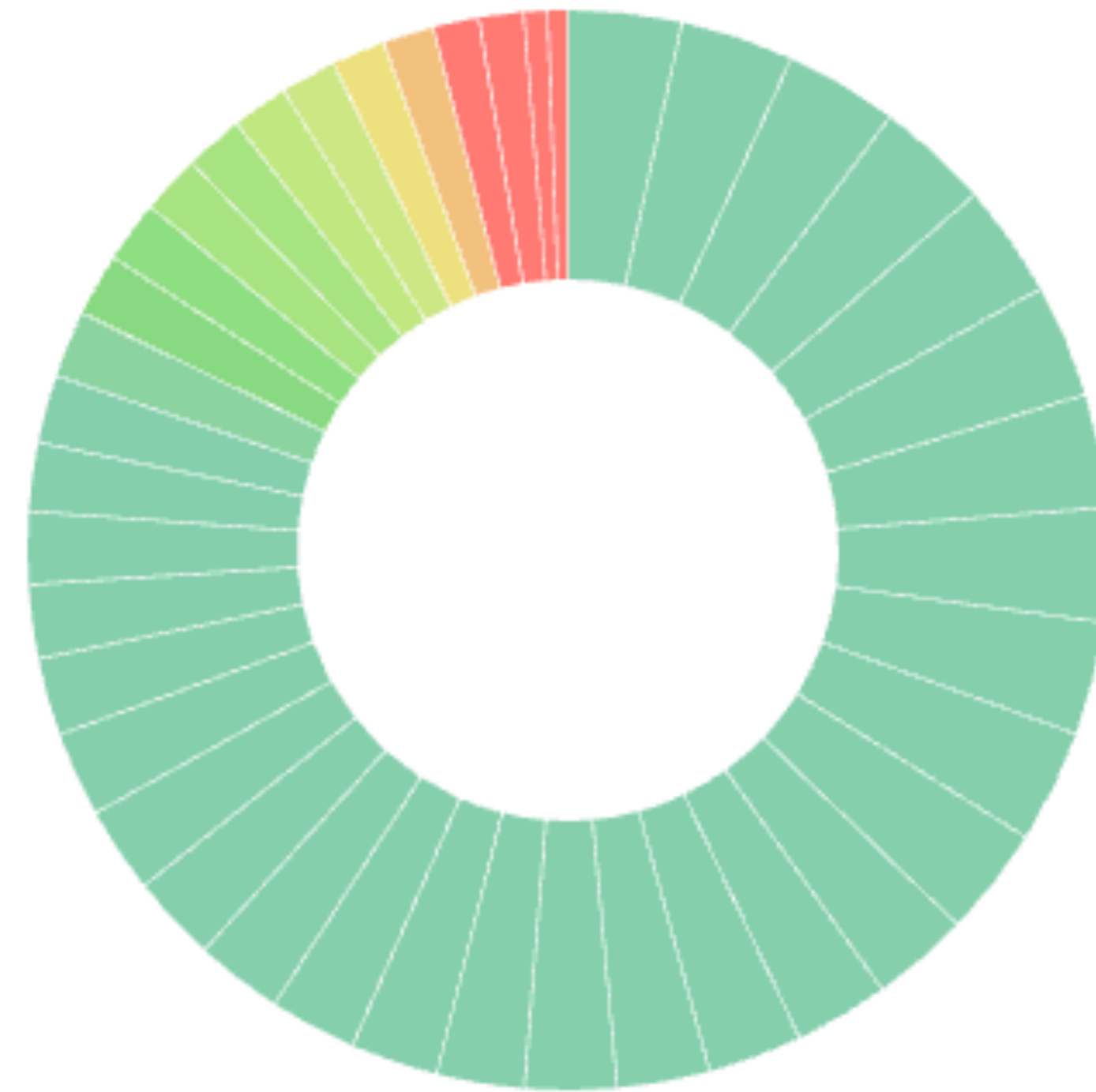


You can use the materials digitalcorpora.org!

You can improve the test coverage of `bulk_extractor` or `be2_api`!



bulk_extractor



be20_api

You can help us close some of the open issues on github!

The screenshot shows the GitHub interface for the repository 'simsong / bulk_extractor'. The 'Issues' tab is selected, showing 115 open issues. The issues are listed in a table with columns for checkboxes, issue titles, labels, and comment counts. The top 10 issues are:

Issue Title	Labels	Comments
Remove or rewrite INSTALL so that it can be understood by people who are not familiar with GNU autoconf	usability	2
GitHub actions not reporting warnings	SRE	0
Add actual validation to build-jo-work run	enhancement, Good student project, SRE	0
-x all -e outlook does not enable outlook	bug, option processing	0
-J doesn't put sbuf debug information into report.xml	0	2 tasks
bulk_extractor hangs with -F	hang, high priority, multithreading	7
Running bulk_extractor with --max_minute_wait flag has no effect	bug, high priority	0
Running bulk_extractor with debug options has not effect on run	bug	4
Running bulk_extractor with -J uses multiple threads	bug, hang, high priority	8
Hang: -R with 10,000 files and 20 threads on MacBook Pro	hang, high priority, multithreading	5 tasks

Several issues are identified as “good student project”

The screenshot shows the GitHub interface for the repository 'simsong / bulk_extractor'. The 'Issues' tab is active, displaying 115 issues. A search filter is applied: 'is:open label:"Good student project"'. The results show 6 open issues and 0 closed issues. The issues are listed in a table with columns for checkboxes, status, title, labels, and comments.

<input type="checkbox"/>	6 Open	0 Closed	Author	Label	Projects	Milestones	Assignee	Sort
<input type="checkbox"/>	🟢			enhancement	Good student project	SRE		
	#419 opened on Apr 29 by simsong							
<input type="checkbox"/>	🟢			BE2.0	enhancement	Good student project	SRE	4
	#394 opened on Mar 25 by simsong							
<input type="checkbox"/>	🟢			enhancement	Good student project			12
	#364 opened on Jun 29, 2022 by Donovoi							
<input type="checkbox"/>	🟢			BE2.0	enhancement	Good student project		
	#240 opened on Sep 10, 2021 by simsong 2 tasks							
<input type="checkbox"/>	🟢			enhancement	Good student project	performance		
	#232 opened on Sep 1, 2021 by simsong 2 tasks							
<input type="checkbox"/>	🟢				Good student project			1
	#212 opened on Aug 2, 2021 by simsong 1 of 3 tasks							

You can work on one of our “enhancements”

Filters Labels 30 Milestones 3 [New issue](#)

Clear current search query, filters, and sorts

<input type="checkbox"/>	<input type="radio"/> 37 Open <input checked="" type="checkbox"/> 10 Closed	Author	Label	Projects	Milestones	Assignee	Sort
<input type="checkbox"/>	<input checked="" type="radio"/> Allow scanners to be written in Rust. enhancement #428 opened now by simsong						
<input type="checkbox"/>	<input checked="" type="radio"/> Add actual validation to build-jo-work run enhancement Good student project SRE #419 opened on Apr 29 by simsong						
<input type="checkbox"/>	<input checked="" type="radio"/> integrate with a web scraper enhancement #399 opened on Apr 5 by simsong						2
<input type="checkbox"/>	<input checked="" type="radio"/> Improve code coverage with more RAR decodes BE2.0 enhancement Good student project SRE #394 opened on Mar 25 by simsong						4
<input type="checkbox"/>	<input checked="" type="radio"/> bulk_extractor needs a quote printable decoder enhancement Good student project #364 opened on Jun 29, 2022 by Donovoi						12
<input type="checkbox"/>	<input checked="" type="radio"/> Warn if compiled without -O3 enhancement #349 opened on Feb 18, 2022 by simsong						
<input type="checkbox"/>	<input checked="" type="radio"/> Add support for YARA enhancement #320 opened on Dec 22, 2021 by simsong						3
<input type="checkbox"/>	<input checked="" type="radio"/> add status information when restarting enhancement #319 opened on Dec 21, 2021 by simsong						
<input type="checkbox"/>	<input checked="" type="radio"/> More efficient handling of seen before enhancement #310 opened on Dec 15, 2021 by simsong <input type="checkbox"/> 2 tasks						
<input type="checkbox"/>	<input checked="" type="radio"/> parallelize imgc_iterator enhancement performance #294 opened on Nov 24, 2021 by simsong						4