THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

Keeping Forensic Tools Sharp

A Case Study of Updating bulk_extractor 1.6 to 2.0

Thursday, Feb 24

1:40 p.m. - 2:00 p.m. C14

Simson Garfinkel, PhD*

These slides can be downloaded from https://simson.net/ref/2022/AAFS_Keeping_Tools_Sharp.pdf

Hello!

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

Overview for this talk (15 minutes)

Why digital forensics tools require continual updating

Case study: bulk_extractor

What this means for open-source digital forensics

Digital forensics tools require constant maintenance

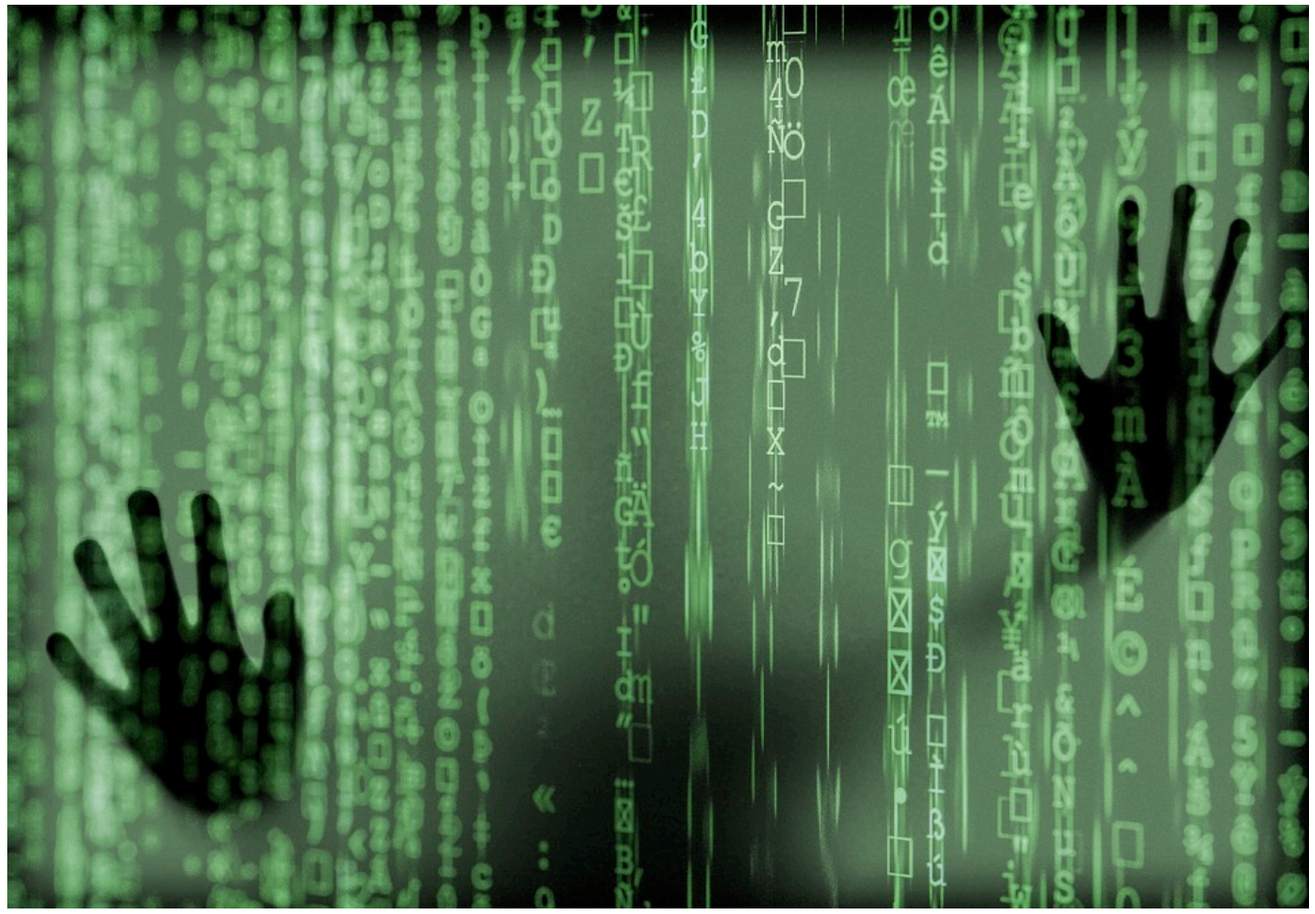
THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

OS Creep

Language Creep

Forensic Science Creep

O&M (operations & maintenance) “tail”



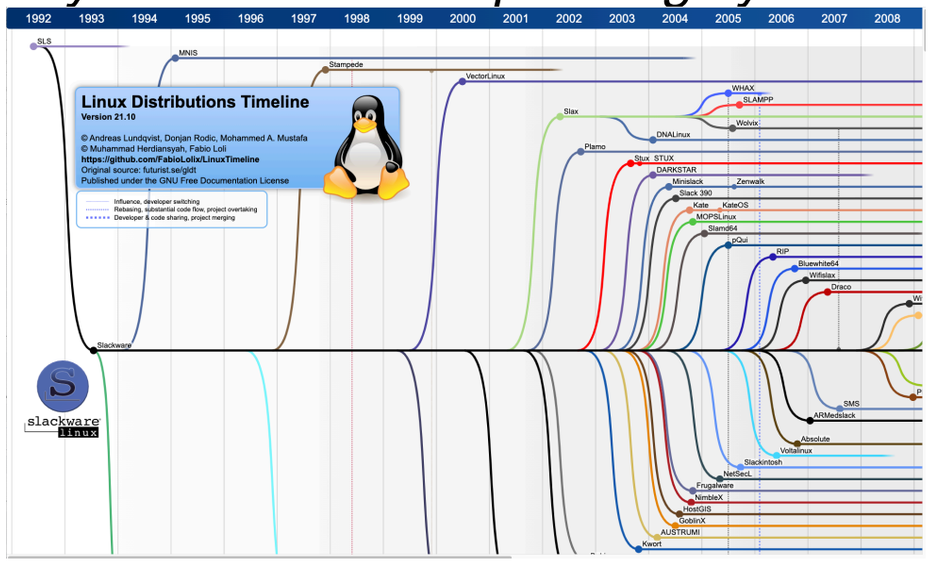
https://pixabay.com/illustrations/hacker-computer-ghost-cyber-code-4031973/

Digital forensics tools require constant maintenance

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

OS Creep

- Platforms being analyzed change over time
  - *Windows 95 → Windows NT → Windows XP → Windows 7 → Windows 10*
  - *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.*



Digital forensics tools require constant maintenance

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

OS Creep ✔

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*

Digital forensics tools require constant maintenance

OS Creep ✔

Language Creep ✔

Digital forensics creep — DF science is constantly 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.*

Digital forensics tools require constant maintenance

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

OS Creep ✔

Language Creep ✔

Forensic Science Creep ✔

O&M (operations & maintenance) “tail”

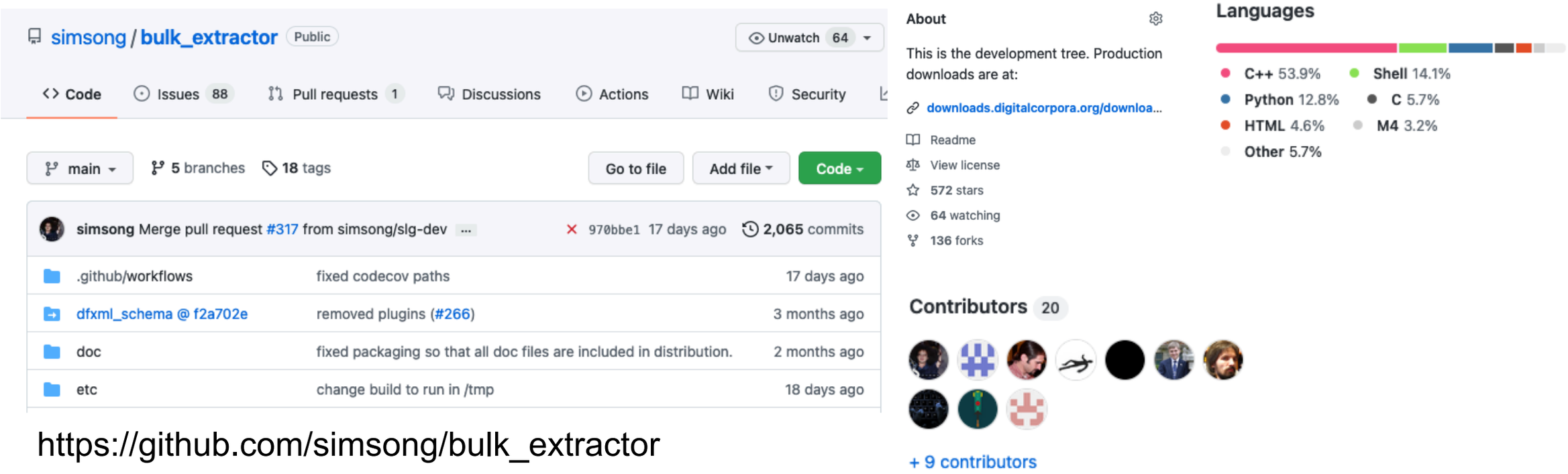
- All software needs to be maintained
- DF software is not any different
  - *Bugs reported in software*
  - *Updates to secure hash algorithms (MD5 ❌; SHA-1 ❌; SHA-256 ✔)*

Case Study: Updating bulk_extractor from 1.6 to 2.0

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

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

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

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% faster than in 2012
- Laptops and low-end workstations have 2x to 3x as many cores
- High-end servers: 64 cores in 2012; 96 cores in 2020
- Memory is 3x faster; SSDs are commonplace now → no seek time
- 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 end of page processing, and single-threaded.

The most important reason: Correctness

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

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

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

Make the program easier to compile and maintain

Make it easier for others to contribute code

Removal experimental code & simplify the codebase

Decrease program’s runtime

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 eccp	Intel C++	IBM XLC++	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	
static_assert with no message	N3928	6	2.5	19.10*	Yes	4.12	18.0			10.3		17.7	11.0	
typename in a template 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 definition	N4230	6	3.6	19.0 (Update 3)*	Yes	4.12	17.0			10.3		17.7	11.0	
Attributes for		4.9												

C++20 core language features

C++20 feature	Paper(s)	GCC	Clang	MSVC	Apple Clang	EDG eccp	Intel C++	IBM XLC++	Sun/Oracle C++	Embarcadero C++ Builder	Cray	Nvidia HPC C++ (ex Portland Group/PGI)	Nvidia nvcc	[Collapse]
Allow lambda-capture [=, this]	P0409R2	8	6	19.22*	10.0.0*	5.1						20.7		
__VA_OPT__	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 bit-fields	P0683R1	8	6	19.25*	10.0.0*	5.1						20.7		
Initializer list constructors in														

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

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

C++ instrumentation

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

Example: Base64 identification

Expected result

```
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  }
```

1. Verify BASE64 recognition
2. Verify BASE64 recognition among other data.
3. Verify BASE64 properly decoded

Automating Tests - End-to-End tests

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

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.

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

Update plan: objectives

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

Make the program easier to compile and maintain ✔

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

Removal experimental code & simplify the codebase

Decrease program’s runtime

Split projects up into modules for improved maintainability.

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

bulk_extractor 1.0 consists of three git modules:

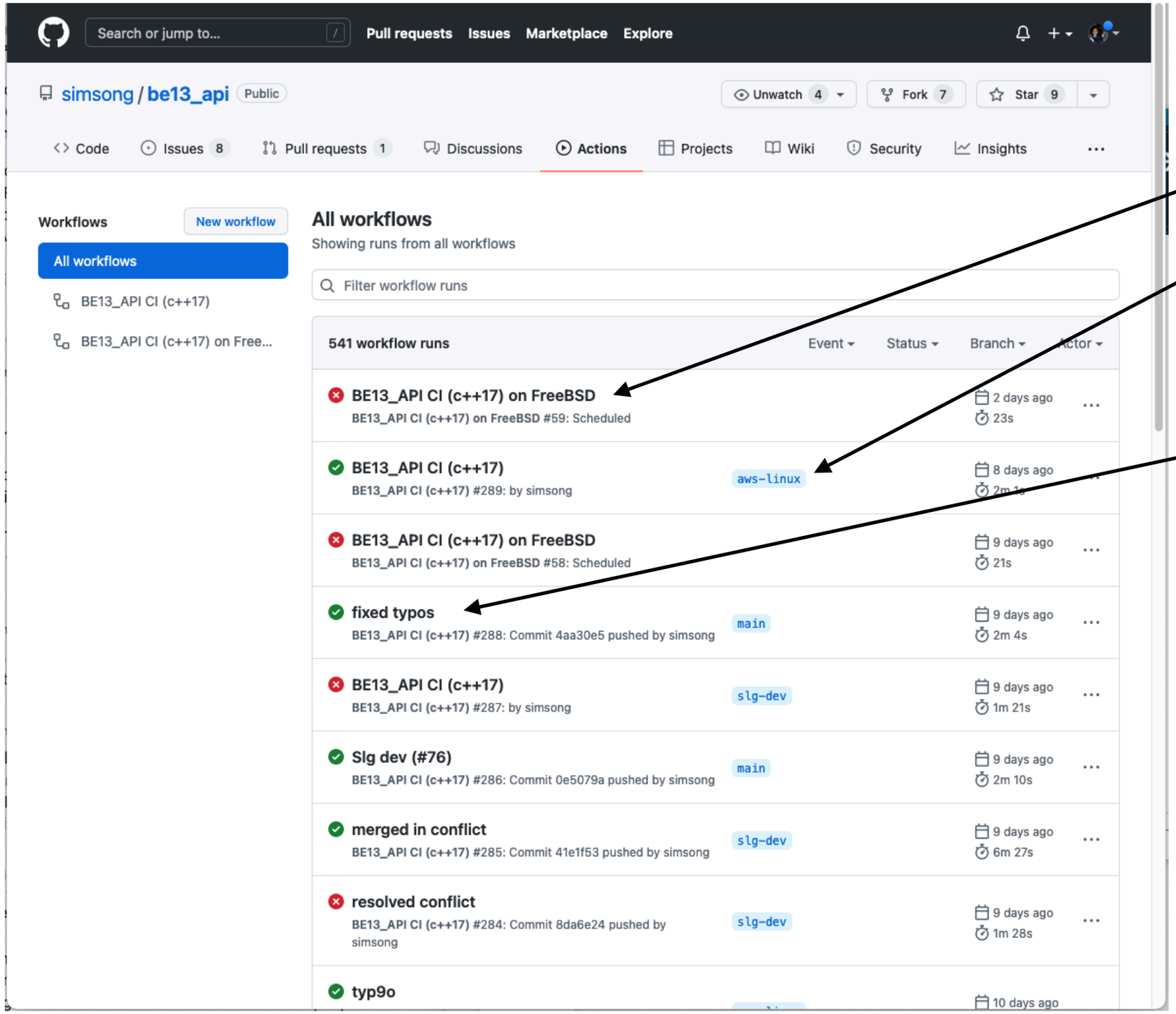
- github.com://simsong/bulk_extractor.git —CLI, GUI, data reader, scanners
- github.com://simsong/be13_api.git — Framework for scanner set, feature recorders
- github.com://simsong/dfxml.git — Digital Forensics XML writer.

For bulk_extractor 2.0:

- github.com://simsong/bulk_extractor.git
- 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 GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC



Tests on FreeBSD failing

Branch being tested

Commit to “main” did not cause any problems!

“codecov” tool integrates with GitHub Actions

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

simsong/be13_api: API for bulk_

https://github.com/simsong/be13_api

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

README.md

be13_api

branch slg-dev: codecov 60%

codecov 60%

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

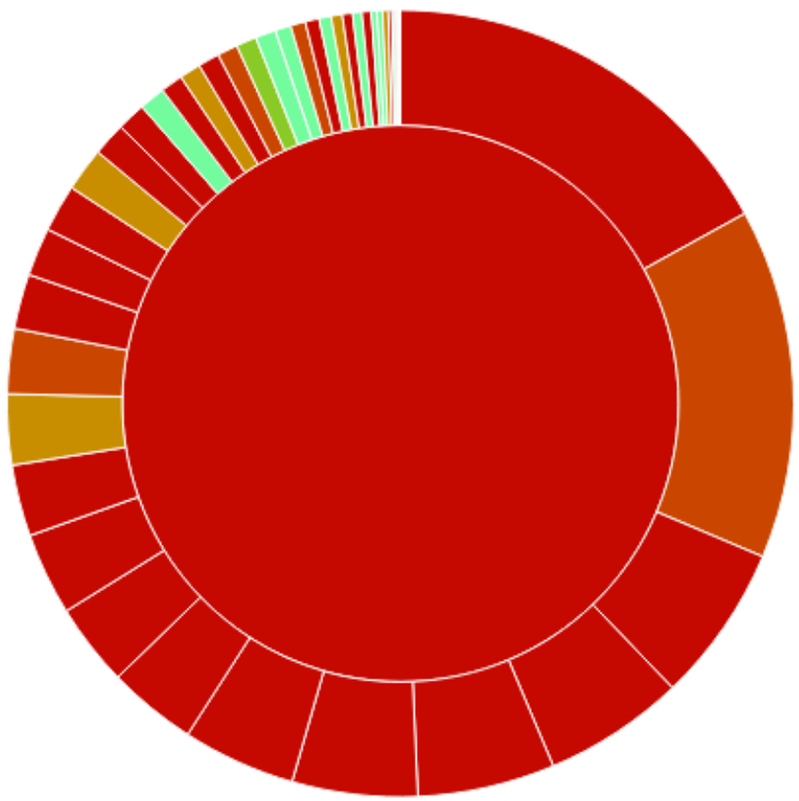
The Bulk_Extractor API is a plug-in API for bulk_extractor "scanners." Scanners are implemented as `extern "C"` functions which are called from the bulk_extractor C++ framework. All bulk_extractor scanners are implemented using the API. Scanners can either be compiled into the bulk_extractor executable, 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 differnece 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



/ 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

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

Make the program easier to compile and maintain ✔

Make it easier for others to contribute code ✔

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*

Decrease program's runtime

Update plan: objectives

Make the program easier to compile and maintain ✔

Make it easier for others to contribute code ✔

Removal 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

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

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 is finding a lot of stuff that BE1 missed

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

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

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

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