Graphtage: A New Semantic Diffing Tool

POST AUGUST 28, 2020 1 COMMENT

Graphtage is a command line utility and underlying library for semantically comparing and merging tree-like structures such as JSON, JSON5, XML, HTML, YAML, and TOML files. Its name is a portmanteau of "graph" and "graftage" (i.e., the horticultural practice of joining two trees together so they grow as one). Read on for:

- What Graphtage does differently and better
- Why we developed it
- How it works
- Directions for using it as a library

All sorts of good

Graphtage lets you see what's different between two files quickly and easily, but it *isn't* a standard line-oriented comparison tool like `diff`. Graphtage is semantically aware, which allows it to map differences across unordered structures like JSON dicts and XML element tags. You can even compare files that are in two different formats! And when paired with our PolyFile tool, you can semantically diff arbitrary file formats.

Tree-like file formats are becoming increasingly common as a means for transmitting and storing data. If you've ever wrangled a gnarly REST API, disentangled the output of a template-generated webpage, or confabulated a config file (and subsequently needed to figure out which specific change was the one that made things work), you've probably fought with—and been disappointed by—the current state of open-source semantic diffing tools.

Graphtage solves these problems. It's available today. To install the utility, run:

```
pip3 install graphtage
```

Grab the source code here.

How are existing diff tools insufficient?

Ordered nodes in the tree (e.g., JSON lists) and, in particular, mappings (e.g., JSON dicts) are challenging. Most extant diffing algorithms and utilities assume that the structures are ordered. Take this JSON as an example:

```
# original.json
{
```

Trail of Bits Blog

Home

Search …

ABOUT US

Since 2012, Trail of Bits has helped secure some of the world's most targeted organizations and products. We combine high-end security research with a real world attacker mentality to reduce risk and fortify code.

Read more at www.trailofbits.com

SUBSCRIBE VIA RSS

RSS - Posts

RECENT POSTS

- Graphtage: A New Semantic Diffing Tool
- Using Echidna to test a smart contract library
- Sinter: New user-mode security enforcement for macOS
- Accidentally stepping on a DeFi lego
- Contract verification made easier
- Advocating for change
- Upgradeable contracts made safer with Crytic
- ECDSA: Handle with Care
- How to check if a mutex is locked in Go
- Breaking the Solidity Compiler with a Fuzzer
- Detecting Bad OpenSSL Usage
- Verifying Windows binaries, without Windows
- Emerging Talent: Winternship 2020 Highlights
- Reinventing Vulnerability Disclosure using Zero-knowledge Proofs
- Bug Hunting with Crytic

YEARLY ARCHIVE

- 2020
- 2019
- 2018
- 2017
- 2016
- 2015
- 2014

```
    "foo": [1, 2, 3, 4],
    "bar": "testing"
}
```

```
# modified.json
{
    "foo": [2, 3, 4, 5],
    "zab": "testing",
    "woo": ["foobar"]
}
```

Existing tools effectively canonicalize the JSON (e.g., sort dictionary elements by key and format lists with one item per line), and then perform a traditional diff. We don't need no fancy tools for that! Here's effectively what they do:

```
$ cat original.json | jq -M --sort-keys > original.canonica
$ cat modified.json | jq -M --sort-keys > modified.canonica
$ diff -u original.canonical.json modified.canonical.json
```

```
 1  {
 2  -   "bar": "testing",
 3      "foo": [
 4  -     1,
 5        2,
 6        3,
 7  -     4
 8  -   ]
 9  +     4,
10  +     5
11  +   ],
12  +   "woo": [
13  +     "foobar"
14  +   ],
15  +   "zab": "testing"
16  }
```

That result is not very useful, particularly if the input files are large. The problem is that changing dict keys breaks the diff: Since "bar" was changed to "zab," the canonical representation changed, and the traditional diff algorithm considered them separate edits (lines 2 and 15 of the diff).

In contrast, here is Graphtage's output for the same pair of files:

- 2013
- 2012

CATEGORIES

- Apple (13)
- Attacks (7)
- Authentication (5)
- Binary Ninja (11)
- Blockchain (41)
- Capture the Flag (10)
- Compilers (20)
- Conferences (27)
- Containers (2)
- Cryptography (28)
- Crytic (2)
- Cyber Grand Challenge (7)
- DARPA (19)
- Dynamic Analysis (12)
- Education (13)
- Empire Hacking (7)
- Engineering Practice (11)
- Events (5)
- Exploits (23)
- Fuzzing (25)
- Go (3)
- Guides (8)
- Internship Projects (18)
- iVerify (4)
- Kubernetes (2)
- Linux (1)
- Malware (7)
- Manticore (14)
- McSema (11)
- Meta (10)
- Mitigations (9)
- osquery (20)
- Paper Review (11)
- Press Release (22)
- Privacy (7)
- Products (5)
- Program Analysis (15)
- Research Practice (10)
- Reversing (12)
- Rust (3)
- SafeDocs (1)
- Sinter (1)
- Sponsorships (12)
- Static Analysis (20)
- Symbolic Execution (15)
- Training (1)
- Year in Review (4)

My Tweets

Why hasn’t this been done before?

In general, optimally mapping one graph to another cannot be executed in polynomial time, and is therefore not tractable for graphs of any useful size (unless P=NP). This is true even for restricted classes of graphs like DAGs. However, trees and forests are special cases that can be mapped in polynomial time, with reasonable constraints on the types of edits possible. Graphtage exploits this.

How do it know?

Graphtage’s diffing algorithms operate on an intermediate representation rather than on the data structures of the original file format. This allows Graphtage to have generic comparison algorithms that can work on any input file type. Therefore, to add support for a new file type, all one needs to do is “lift” it to the intermediate representation. Likewise, one only needs to implement support for a new type of edit once, and it will immediately be available to apply against all supported filetypes. Using an intermediate representation has the added benefit of allowing cross-format comparisons and formatting translations: Graphtage will happily diff a JSON file against a YAML file, formatting the diff output in TOML syntax.

Graphtage matches ordered sequences like lists using an “online” “constructive” implementation of the Levenshtein distance metric, similar to the Wagner–Fischer algorithm. The algorithm starts with an unbounded mapping and iteratively improves it until the bounds converge, at which point the optimal edit sequence is discovered.

Dicts are matched by solving the minimum weight matching problem on the complete bipartite graph from key/value pairs in the source dict to key/value pairs in the destination dict.

Graphtage is a command line utility, but it can just as easily be used as a library. One can interact with Graphtage directly from Python, and extend it to support new file formats and edit types.

Next up for Graphtage

We think Graphtage is pretty nifty. You can also use Graphtage in conjunction with our PolyFile tool to semantically diff arbitrary file formats, even if they aren't naturally tree-based. Try it, and let us know how you use it.

We also plan to extend Graphtage to work on abstract syntax trees, which will allow your source code diffs to tell you things like which variables were changed and whether code blocks were reordered. If you have a similarly nifty idea for a new feature, please share it with us!

Note: This tool was partially developed with funding from the Defense Advanced Research Projects Agency (DARPA) on the SafeDocs project. The views, opinions, and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

Share this:

Twitter LinkedIn Reddit Telegram Facebook Pocket Email Print

Like this:

Loading...

By Evan Sultanik Posted in DARPA, Research Practice, SafeDocs

← Using Echidna to test a smart contract library

One thought on "Graphtage: A New Semantic Diffing Tool"

Daniel Bigham

August 28, 2020 at 6:40 pm Reply

An interesting area. I feel like this is an important chunk of software that could power a lot of useful things. I use this myself for things like showing how some expected test output is different from actual test output, etc.

Loading...

Leave a Reply

https://jetpack.wordpress.com/jetpack-comment/?blogid=3681601&postid=100164&comment_registration=0&require_name_e