Firepit Documentation

Release 2.3.26

IBM Security

Aug 21, 2023

CONTENTS:

CHAPTER

ONE

FIREPIT - STIX COLUMNAR STORAGE

Columnar storage for STIX 2.0 observations.

- Free software: Apache Software License 2.0
- Documentation: https://firepit.readthedocs.io.

1.1 Features

- Transforms STIX Observation SDOs to a columnar format
- Inserts those transformed observations into SQL (currently sqlite3 and PostgreSQL)

1.2 Motivation

STIX 2.0 JSON is a graph-like data format. There aren't many popular tools for working with graph-like data, but there are numerous tools for working with data from SQL databases. Firepit attempts to make those tools usable with STIX data obtained from stix-shifter.

Firepit also supports STIX 2.1

Firepit is primarily designed for use with the Kestrel Threat Hunting Language.

1.3 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER

TWO

INSTALLATION

2.1 Stable release

To install firepit, run this command in your terminal:

```
$ pip install firepit
```

This is the preferred method to install firepit, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for firepit can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/opencybersecurityalliance/firepit
```

Or download the tarball:

```
$ curl -OJL https://github.com/opencybersecurityalliance/firepit/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER

THREE

USAGE

3.1 As a package

To use firepit in a project:

```
from firepit import get_storage

db = get_storage('observations.db', session_id)
db.cache('my_query_id', ['bundle1.json', 'bundle2.json'])
```

After caching your STIX bundles, your database will contain the data.

Passing a file path to *get_storage* will use sqlite3. Passing in a PostgreSQL connection URI (e.g. postgresql://...) will instead attempt to connect to the PostgreSQL instance specified.

3.2 As a tool

You can use the *firepit* command line tool to ingest and query your data.

To make things easier, you can set a pair of environment variables:

```
export FIREPITDB=my_dbname
export FIREPITID=my_session_id
```

FIREPITDB is your DB filename (sqlite3) or connection URI (PostgreSQL). *FIREPITID* is a "session" ID you can use to keep your data organized.

```
$ firepit --help
Usage: firepit [OPTIONS] COMMAND [ARGS]...

  Columnar storage for STIX observations

Options:
  --dbname TEXT   Path/name of database
  --session TEXT  Session ID to data separation  [default: test-id]
  --help          Show this message and exit.

Commands:
  assign           Perform an operation on a column and name the result
  cache            Cache STIX observation data in SQL
```

(continues on next page)

(continued from previous page)

```
  columns          Get the columns names of a view/table
  count            Get the count of rows in a view/table
  delete           Delete STIX observation data in SQL
  extract          Create a view of a subset of cached data
  filter           Create a filtered view of a subset of cached data
  get-appdata      Get the app-specific data for a view
  join             Join two views
  load             Cache STIX observation data in SQL
  lookup           Retrieve a view
  merge            Merge 2 or more views into a new view
  number-observed  Retrieve the count of values of a column from a view
  reassign         Update/replace STIX observation data in SQL
  remove           Remove a view
  rename           Rename a view
  schema           Get the schema of a view/table
  set-appdata      Set the app-specific data for a view
  sql              Run a SQL statement on the database [DANGEROUS!]
  summary          Retrieve timeframe and count from a view
  tables           Get all view/table names
  timestamped      Retrieve the timestamped values of a column from a view
  type             Get the SCO type of a view/table
  value-counts     Retrieve the value counts of a column from a view
  values           Retrieve the values of a STIX object path from a view
  viewdata         Get view data for views [default is all views]
  views            Get all view names

$ firepit cache --help
Usage: firepit cache [OPTIONS] QUERY_ID FILENAMES...

  Cache STIX observation data in SQL

Arguments:
  QUERY_ID      An identifier for this set of data  [required]
  FILENAMES...  STIX bundle files of query results  [required]

Options:
  --help  Show this message and exit.
```

3.3 splint

Firepit also includes a utility called *splint*. This tool supports multiple commands for manipulating STIX 2.0 bundles (2.1 is not yet supported), including randomzing IDs, shifting timestamps, and converting other log formats to STIX observations.

The *convert* command currently supports the JSON format at https://securitydatasets.com as well as Zeek *conn.log* and *dns.log*.

```
$ splint
Usage: splint [OPTIONS] COMMAND [ARGS]...
```

(continues on next page)

(continued from previous page)

```
  STIX processing and linting

Options:
  --help  Show this message and exit.

Commands:
  convert        Convert various log files to STIX 2.0 bundles
  dedup-ids      Replace duplicate IDs with random IDs
  limit          Truncate STIX bundle
  randomize-ids  Randomize STIX observation IDs in a bundle
  timeshift      Timeshift STIX observations in a bundle
  upgrade        Upgrade a STIX 2.0 bundle to 2.1
```

CHAPTER

FOUR

DATABASE

4.1 Supported Databases

Firepit supports sqlite3 and PostgreSQL.

4.2 Database Tables

STIX observation data is inserted into multiple tables within a "session" (a database file for sqlite3 and a "schema" in PostgreSQL). A table is created for each object type. Since STIX data is a graph (i.e. nodes and edges), Firepit also creates some special "edge" tables:

- *__contains*: tracks which SCOs were contained in which *observed-data* SDOs
- *__reflist*: models 1:N reference lists like *process:opened_connection_refs*
- *__queries*: records which objects were inserted in which *cache* operations
- *__symtable*: records the name and type of "views" created by firepit calls

These tables are prefixed with __ and considered "private" by firepit.

The STIX *id* property is used as the unique key for each table.

4.2.1 The *observed-data* Table

This tables contains the actual STIX Observed Data SDO that "conveys information about cyber security related entities such as files, systems, and networks using the STIX Cyber-observable Objects (SCOs)." [STIX-v2_1]

This SDO (and therefore table) holds the timestamps and count of actual observations, whereas SCOs (and their firepit tables) only contain the properties (columns) of their respective object types.

The examples below show how to link *observed-data* with SCOs via the "private" *__contains* table.

4.2.2 SCO Tables

Each SCO table (*ipv4-addr*, *network-traffic*, *file*, etc.) contains the properties present from the cached bundles. Firepit does not require any specific properties (though STIX does). Columns are only created for properties found.

For example, the *network-traffic* table should have properties *src_ref* (a reference to an object in either the *ipv4-addr* or *ipv6-addr* table) which represents the connection's source address, *dst_ref*, *src_port*, *dst_port*, and *protocols*. The port properties are simple integers, and stored in integer columns. The *protocols* column is a list of strings; it's stored as a JSON-encoded string.

4.3 STIX Object Paths

STIX object paths (e.g. *network-traffic:src_ref.value*) are a key part of STIX patterning, which (from Firepit's perspective) is equivalent to a WHERE clause. They can contain implicit JOINs: *network-traffic* is a table, *src_ref* is the *id* property for an *ipv4-addr* (or *ipv6-addr*) which is the unique key for that table. *value* is a column in that referenced table.

Firepit operations will (in most cases) accept STIX object paths and create the required JOIN.

4.4 Example SQL queries

4.4.1 Full Network Traffic Information

The *network-traffic* SCO only contains references to the source and destination addresses. To see the actual addresses, you need to join the *ipv4-addr* table:

```
sqlite> select
   ...>   src.value as "src_ref.value",
   ...>   nt.src_port as "src_port",
   ...>   dst.value as "dst_ref.value",
   ...>   nt.dst_port as "dst_port",
   ...>   nt.protocols
   ...> from
   ...>   "network-traffic" as nt
   ...>   join "ipv4-addr" as src on nt.src_ref = src.id
   ...>   join "ipv4-addr" as dst on nt.dst_ref = dst.id
   ...> ;
src_ref.value  src_port    dst_ref.value  dst_port    protocols
-------------  ----------  -------------  ----------  ----------
192.168.1.156  60842       192.168.1.1    47413       ["tcp"]
127.0.0.1      60843       127.0.0.1      5357        ["tcp"]
```

The *firepit* CLI makes this easier; for example, using the *lookup* command:

```
$ firepit lookup network-traffic --columns src_ref.value,src_port,dst_ref.value,dst_port,
↪protocols
src_ref.value      src_port  dst_ref.value      dst_port  protocols
---------------  ----------  ---------------  ----------  -----------
192.168.1.156         60842  192.168.1.1           47413  ["tcp"]
127.0.0.1             60843  127.0.0.1              5357  ["tcp"]
```

Most CLI commands have an API function of the same name in the SqlStorage class.

4.4.2 Timestamped SCOs

To see the first 3 IP addresses observed, join the special *__contains* and *observed-data* tables:

```
sqlite> select obs.first_observed as time, sco.value as 'IP'
   ...>  from "ipv4-addr" as sco
   ...>  join "__contains" as c on sco.id = c.target_ref
   ...>  join "observed-data" as obs on c.source_ref = obs.id
   ...>  order by time limit 3;
time                    IP
----------------------  -------------
2019-11-16T12:55:28.101Z  192.168.1.156
2019-11-16T12:55:28.101Z  192.168.1.1
2019-11-16T12:55:28.883Z  127.0.0.1
```

This is effectively equivalent to the CLI's *timestamped* command or the API's *timestamped* function:

```
$ firepit timestamped ipv4-addr value | head -5
first_observed            value
------------------------  ---------------
2019-11-16T12:55:28.101Z  192.168.1.156
2019-11-16T12:55:28.101Z  192.168.1.1
2019-11-16T12:55:28.883Z  127.0.0.1
```

4.4.3 Value counts

To get a count of observations of each IP address (the *sqlite3* CLI truncates the *value* column):

```
sqlite> select sco.value, count(*) from "ipv4-addr" as sco
   ...>  join "__contains" as c on sco.id = c.target_ref
   ...>  join "observed-data" as obs on c.source_ref = obs.id
   ...>  group by sco.value;
value       count(*)
----------  ----------
127.0.0.1   413
172.16.0.1  33
172.16.0.1  7
172.16.0.1  8
172.16.0.1  24
172.16.0.2  13
192.168.1.  166
192.168.1.  138
192.168.1.  1
192.168.1.  3
192.168.1.  4
192.168.17  8
192.168.17  1
192.168.17  4
192.168.23  10
192.168.23  2
192.168.23  1
192.168.23  4
```

Again, this operation is provided by the CLI's *value-counts* command or the API's *value_counts* function:

```
$ firepit value-counts ipv4-addr value
value              count
---------------  -------
127.0.0.1            413
172.16.0.100          33
172.16.0.101           7
172.16.0.104           8
172.16.0.112          24
172.16.0.255          13
192.168.1.1          166
192.168.1.156        138
192.168.1.163          1
192.168.1.169          3
192.168.1.255          4
192.168.175.1          8
192.168.175.254        1
192.168.175.255        4
192.168.232.1         10
192.168.232.2          2
192.168.232.254        1
192.168.232.255        4
```

CHAPTER

FIVE

FIREPIT

5.1 firepit package

5.1.1 Subpackages

5.1.2 Submodules

5.1.3 firepit.deref module

firepit.deref.**auto_deref**(*store*, *view*, *ignore=None*, *paths=None*)

Automatically resolve refs for backward compatibility.

If *paths* is specified, only follow/deref those specific paths/properties.

Use auto_deref_cached if you already have col_dict in memory.

firepit.deref.**auto_deref_cached**(*view*, *cols*, *col_dict*, *ignore=None*, *paths=None*)

Automatically resolve refs for backward compatibility.

If *paths* is specified, only follow/deref those specific paths/properties.

firepit.deref.**unresolve**(*objects*)

Do the opposite of auto_deref: split out reference objects

5.1.4 firepit.exceptions module

exception firepit.exceptions.**DatabaseMismatch**(*dbversion*, *expected*)

Bases: Exception

exception firepit.exceptions.**DuplicateTable**

Bases: Exception

exception firepit.exceptions.**IncompatibleType**

Bases: Exception

exception firepit.exceptions.**InvalidAttr**(*msg*)

Bases: Exception

exception firepit.exceptions.**InvalidObject**(*msg*)

Bases: Exception

exception firepit.exceptions.**InvalidStixPath**

Bases: Exception

exception firepit.exceptions.**InvalidViewname**

Bases: Exception

exception firepit.exceptions.**SessionExists**

Bases: Exception

exception firepit.exceptions.**SessionNotFound**

Bases: Exception

exception firepit.exceptions.**StixPatternError**(*stix*)

Bases: Exception

exception firepit.exceptions.**UnexpectedError**

Bases: Exception

exception firepit.exceptions.**UnknownViewname**

Bases: Exception

5.1.5 firepit.generator module

5.1.6 firepit.pgcommon module

Some common PostgreSQL stuff used by both pgstorage.py (the normal interface) and aio.asyncpgstorage.py (the async interface).

firepit.pgcommon.**pg_shorten**(*key*)

5.1.7 firepit.pgstorage module

5.1.8 firepit.props module

Utility functions for STIX properties

firepit.props.**auto_agg**(*sco_type*, *prop*, *col_type*)

Infer an aggregation function based on column name and type

firepit.props.**auto_agg_tuple**(*sco_type*, *prop*, *col_type*)

Infer an aggregation function based on column name and type

firepit.props.**get_last**(*prop*)

firepit.props.**is_ref**(*name*)

firepit.props.**parse_path**(*path*)

firepit.props.**parse_prop**(*sco_type*, *prop*)

firepit.props.**path_metadata**(*path*)

Get metadata for a STIX object path

firepit.props.**primary_prop**(*sco_type*)

Returns the "primary" property name for each SCO type

firepit.props.**prop_metadata**(*sco_type*, *prop*)

Get metadata for a STIX object property

firepit.props.**ref_type**(*sco_type*, *part*)

Get STIX SCO type for reference prop *part*

5.1.9 firepit.query module

Utilities for generating SQL while avoiding SQL injection vulns

class firepit.query.**Aggregation**(*aggs*)

Bases: object

Aggregate rows

render(*_placeholder*, *_dialect=None*)

class firepit.query.**BinnedColumn**(*prop: str*, *n: int*, *unit: Optional[str] = None*, *table: Optional[str] = None*, *alias: Optional[str] = None*)

Bases: *Column*

Bin (or "bucket") column values, persumably for easier grouping

render(*_placeholder*, *dialect=None*)

class firepit.query.**CoalescedColumn**(*names*, *alias*)

Bases: object

First non-null column from a list - used after a JOIN

class firepit.query.**Column**(*name*, *table=None*, *alias=None*)

Bases: object

SQL Column name

endswith(*s*)

class firepit.query.**Count**

Bases: object

Count the rows in a result set

render(*_placeholder*, *_dialect=None*)

class firepit.query.**CountUnique**(*cols=None*)

Bases: object

Unique count of the rows in a result set

render(*_placeholder*, *_dialect=None*)

class firepit.query.**Filter**(*preds*, *op=' AND '*)

Bases: object

Alternative SQL WHERE clause

AND = ' AND '

OR = ' OR '

render(*placeholder*, *_dialect=None*)

set_table(*table*)

Specify table for ALL Predicates in Filter

class firepit.query.**Group**(*cols*)

Bases: object

SQL GROUP clause

render(*_placeholder*, *_dialect=None*)

exception firepit.query.**InvalidAggregateFunction**

Bases: Exception

exception firepit.query.**InvalidComparisonOperator**

Bases: Exception

exception firepit.query.**InvalidJoinOperator**

Bases: Exception

exception firepit.query.**InvalidPredicateOperand**

Bases: Exception

exception firepit.query.**InvalidPredicateOperator**

Bases: Exception

exception firepit.query.**InvalidQuery**

Bases: Exception

class firepit.query.**Join**(*name*, *left_col=None*, *op=None*, *right_col=None*, *preds=None*, *how='INNER'*, *alias=None*, *lhs=None*)

Bases: object

Join 2 tables

render(*placeholder*, *_dialect=None*)

class firepit.query.**Limit**(*num*)

Bases: object

SQL row count

render(*_placeholder*, *_dialect=None*)

class firepit.query.**Offset**(*num*)

Bases: object

SQL row offset

render(*_placeholder*, *_dialect=None*)

class firepit.query.**Order**(*cols*)

Bases: object

SQL ORDER BY clause

ASC = 'ASC'

DESC = 'DESC'

render(*_placeholder*, *_dialect=None*)

class firepit.query.**Predicate**(*lhs*, *op*, *rhs*)

Bases: object

Row value predicate

render(*placeholder*, *_dialect=None*)

set_table(*table*)

Specify table for ALL columns in Predicate

class firepit.query.**Projection**(*cols*)

Bases: object

SQL SELECT (really projection - pick column subset) clause

render(*placeholder*, *dialect=None*)

class firepit.query.**Query**(*arg=None*)

Bases: object

SQL Query statement

SQL order of evaluations: FROM, including JOINs WHERE GROUP BY HAVING WINDOW functions SELECT (projection) DISTINCT UNION ORDER BY LIMIT and OFFSET

append(*stage*)

extend(*stages*)

render(*placeholder*, *dialect=None*)

class firepit.query.**Table**(*name*)

Bases: object

SQL Table selection

render(*_placeholder*, *_dialect=None*)

class firepit.query.**Unique**

Bases: object

Reduce the rows in a result set to unique tuples

render(*placeholder*, *dialect=None*)

5.1.10 firepit.schemas module

5.1.11 firepit.sqlitestorage module

class firepit.sqlitestorage.**SQLiteStorage**(*dbname*)

Bases: *SqlStorage*

columns(*viewname*)

Get the column names (properties) of *viewname*

delete()

Delete ALL data in this store

schema(*viewname=None*)

Get the schema (names and types) of table/view *viewname* or all tables if not specified

tables()

Get all table names

types(*private=False*)

Get all table names that correspond to SCO types

firepit.sqlitestorage.**get_storage**(*path*)

firepit.sqlitestorage.**row_factory**(*cursor*, *row*)

5.1.12 firepit.sqlstorage module

class firepit.sqlstorage.**SqlStorage**

Bases: object

assign(*viewname*, *on*, *op=None*, *by=None*, *ascending=True*, *limit=None*)

DEPRECATED: Perform (unary) operation *op* on *on* and store result as *viewname*

assign_query(*viewname*, *query*, *sco_type=None*)

Create a new view *viewname* defined by *query*

cache(*query_id*, *bundles*, *batchsize=2000*, ***kwargs*)

Cache the result of a query/dataset

Takes the *observed-data* SDOs from *bundles* and "flattens" them, splits out SCOs by type, and inserts into a database with 1 table per type.

Accepts some keyword args for runtime options, some of which may depend on what database type is in use (e.g. sqlite3, postgresql, ...)

Args:

query_id (str): a unique identifier for this set of bundles

bundles (list): STIX bundles (either in-memory Python objects or filename paths)

batchsize (int): number of objects to insert in 1 batch (defaults to 2000)

close()

columns(*viewname*)

Get the column names (properties) of *viewname*

count(*viewname*)

Get the count of objects (rows) in *viewname*

delete()

Delete ALL data in this store

extract(*viewname*, *sco_type*, *query_id*, *pattern*)

Extract all *sco_type* object from the results of *query_id* and store as *viewname*

extract_observeddata_attribute(*viewname*, *name_of_attribute*, *path=None*, *value=None*, *limit=None*, *run=True*)

Get the observations of *value* in *viewname*.`path` Returns list of dicts like {'name_of_attribute': '...', '{column}': '...'} *name_of_attribute* can be a str or list of str (to get multiple attrs)

filter(*viewname*, *sco_type*, *input_view*, *pattern*)

Extract all *sco_type* object from *input_view* and store as *viewname*

finish(*index=True*)

Do any DB-specific post-caching/insertion activity, such as indexing

get_appdata(*viewname*)

Retrieve app-specific data for a viewname

get_view_data(*viewnames=None*)

Retrieve information about one or more viewnames

group(*newname*, *viewname*, *by*, *aggs=None*)

Create new view *newname* defined by grouping *viewname* by *by*

join(*viewname*, *l_var*, *l_on*, *r_var*, *r_on*)

Join vars *l_var* and *r_var* and store result as *viewname*

load(*viewname*, *objects*, *sco_type=None*, *query_id=None*, *preserve_ids=True*)

Import *objects* as type *sco_type* and store as *viewname*

lookup(*viewname*, *cols='**, *limit=None*, *offset=None*, *col_dict=None*)

Get the value of *viewname*

merge(*viewname*, *input_views*)

number_observed(*viewname*, *path*, *value=None*)

Get the count of observations of *value* in *viewname*.`path` Returns integer count

path_joins(*viewname*, *sco_type*, *column*)

Determine if *column* has implicit Joins and return them if so

reassign(*viewname*, *objects*)

Replace *objects* (or insert them if they're not there)

remove_view(*viewname*)

Remove view *viewname*

rename_view(*oldname*, *newname*)

Rename view *oldname* to *newname*

run_query(*query*)

schema(*viewname=None*)

Get the schema (names and types) of table/view *viewname* or all tables if not specified

set_appdata(*viewname*, *data*)

Attach app-specific data to a viewname

summary(*viewname*, *path=None*, *value=None*)

Get the first and last observed time and number observed for observations of *viewname*, optionally specifying *path* and *value*. Returns list of dicts like {'first_observed': '2021-10-…', 'last_observed': '2021-10-…', 'number_observed': N}

table_type(*viewname*)

Get the SCO type for table/view *viewname*

tables()

Get all table names

timestamped(*viewname*, *path=None*, *value=None*, *timestamp='first_observed'*, *limit=None*, *run=True*)

Get the timestamped observations of *value* in *viewname*.`path` Returns list of dicts like {'timestamp': '2021-10-...', '{column}': '...'}

types(*private=False*)

Get all table names that correspond to SCO types

upsert(*cursor*, *tablename*, *obj*, *query_id*, *schema*)

upsert_many(*cursor*, *tablename*, *objs*, *query_id*, *schema*)

value_counts(*viewname*, *path*)

Get the count of observations of each value in *viewname*.`path` Returns list of dicts like {'{column}': '...', 'count': 1}

values(*path*, *viewname*)

Get the values of STIX object path *path* (a column) from *viewname*

views()

Get all view names

firepit.sqlstorage.**get_path_joins**(*viewname*, *sco_type*, *column*)

Determine if *column* has implicit Joins and return them if so

firepit.sqlstorage.**infer_type**(*key*, *value*)

5.1.13 firepit.stix20 module

firepit.stix20.**comp2sql**(*sco_type*, *prop*, *op*, *value*, *dialect*)

firepit.stix20.**get_grammar**()

firepit.stix20.**path2sql**(*sco_type*, *path*)

firepit.stix20.**stix2sql**(*pattern*, *sco_type*, *dialect='sqlite3'*)

firepit.stix20.**summarize_pattern**(*pattern*)

5.1.14 firepit.stix21 module

firepit.stix21.**makeid**(*sco*, *obs=None*)

5.1.15 firepit.timestamp module

firepit.timestamp.**timefmt**(*t*, *prec=3*)

Format Python datetime *t* in RFC 3339-format

firepit.timestamp.**to_datetime**(*timestamp*)

Convert RFC 3339-format *timestamp* to Python datetime

5.1.16 firepit.validate module

STIX and SQL identifier validators

firepit.validate.**validate_name**(*name*)

Make sure *name* is a valid (SQL) identifier

firepit.validate.**validate_path**(*path*)

Make sure *path* is a valid STIX object path or property name

5.1.17 firepit.woodchipper module

class firepit.woodchipper.**FlatJsonMapper**

Bases: *Mapper*

convert(*event*)

detect(*event*)

class firepit.woodchipper.**IscHoneypotJsonMapper**

Bases: *Mapper*

convert(*event*)

detect(*event*)

mapping = {'dest': 'network-traffic:dst_ref.value', 'dport': 'network-traffic:dst_port', 'proto': 'network-traffic:protocols', 'source': 'network-traffic:src_ref.value', 'sport': 'network-traffic:src_port', 'ts': ['first_observed', 'last_observed'], 'url': 'url:value', 'user_agent': "network-traffic:extensions.'http-request-ext'.request_header.'User-Agent'"}

class firepit.woodchipper.**Mapper**

Bases: object

convert(*event*)

detect(*event*)

class firepit.woodchipper.**SdsMapper**

Bases: *Mapper*

common_mapping = {'@timestamp': ['first_observed', 'last_observed'], 'Application': <function split_image>, 'Category': <function to_cat_list>, 'Channel': 'x-oca-event:module', 'EventID': <function to_action_code>, 'Hostname': 'x-oca-asset:hostname', 'Message': <function SdsMapper.<lambda>>, 'ProcessGuid': 'process:x_unique_id', 'ProcessId': 'process:pid', 'ProcessName': <function split_image>, 'SourceName': 'x-oca-event:provider', 'TimeCreated': ['first_observed', 'last_observed']}

convert(*event*)

detect(*event*)

static enhanced_action(*message*)

event_types = {'ConnectPipe': 18, 'CreateKey': 12, 'CreatePipe': 17, 'DeleteKey': 12, 'DeleteValue': 12, 'SetValue': 13}

class firepit.woodchipper.ZeekCsvMapper

Bases: *Mapper*

convert(*event*)

detect(*event*)

zeek_mapping = {'id.orig_h': 'network-traffic:src_ref.value', 'id.orig_p': 'network-traffic:src_port', 'id.resp_h': 'network-traffic:dst_ref.value', 'id.resp_p': 'network-traffic:dst_port', 'orig_ip_bytes': 'network-traffic:src_byte_count', 'orig_pkts': 'network-traffic:src_packets', 'proto': 'network-traffic:protocols', 'resp_ip_bytes': 'network-traffic:dst_byte_count', 'resp_pkts': 'network-traffic:dst_packets', 'ts': <function from_unix_time>}

class firepit.woodchipper.ZeekJsonMapper

Bases: *Mapper*

common_mapping = {'id_orig_h': 'network-traffic:src_ref.value', 'id_orig_p': 'network-traffic:src_port', 'id_resp_h': 'network-traffic:dst_ref.value', 'id_resp_p': 'network-traffic:dst_port', 'proto': 'network-traffic:protocols', 'ts': <function from_unix_time>}

convert(*event*)

detect(*event*)

static process_answers(*answers*)

zeek_mapping = {'conn': {'orig_ip_bytes': 'network-traffic:src_byte_count', 'orig_l2_addr': 'network-traffic:src_ref.resolves_to_refs[0].value', 'orig_pkts': 'network-traffic:src_packets', 'resp_ip_bytes': 'network-traffic:dst_byte_count', 'resp_l2_addr': 'network-traffic:dst_ref.resolves_to_refs[0].value', 'resp_pkts': 'network-traffic:dst_packets'}, 'dns': {'answers': <function ZeekJsonMapper.<lambda>>, 'query': 'domain-name:value'}}

firepit.woodchipper.convert(*input_file*, *output_file=None*)

firepit.woodchipper.convert_to_stix(*input_file*)

firepit.woodchipper.detect_filetype(*input_file*)

firepit.woodchipper.dict2observation(*creator*, *row*)

firepit.woodchipper.fixup_hashes(*hashes: dict*)

firepit.woodchipper.format_val(*sco_type*, *prop*, *val*)

firepit.woodchipper.from_unix_time(*ts*)

firepit.woodchipper.guess_ref_type(*sco_type*, *prop*, *val*)

Get data type for *sco_type*:*prop* reference

firepit.woodchipper.is_file_event(*event_id*)

firepit.woodchipper.**merge_mappings**(*common*, *specific*, *key=None*)

Merge common mapping into specific[key] mapping

firepit.woodchipper.**process_event**(*event*, *mapping*, *event_id=None*)

firepit.woodchipper.**process_events**(*events*, *mappers*, *ident*)

firepit.woodchipper.**process_mapping**(*event*, *mapping*)

firepit.woodchipper.**read_csv**(*fp*, *mappers*, *ident*)

firepit.woodchipper.**read_json**(*fp*, *mappers*, *ident*)

firepit.woodchipper.**read_log**(*fp*, *mappers*, *ident*)

firepit.woodchipper.**recreate_dict**(*obj*, *prop*, *rest*, *val*)

firepit.woodchipper.**set_obs_prop**(*observable*, *path*, *val*, *scos*, *key*)

firepit.woodchipper.**split_file_hash**(*hash_string: str*)

firepit.woodchipper.**split_file_path**(*abs_name: str*, *prefix='file:'*)

firepit.woodchipper.**split_hash**(*hash_string: str*, *prefix: str*, *tag: str = ''*)

firepit.woodchipper.**split_image**(*abs_name: str*, *prefix='process:'*)

firepit.woodchipper.**split_image_hash**(*hash_string: str*)

firepit.woodchipper.**split_image_loaded**(*abs_name: str*)

firepit.woodchipper.**split_loaded_hash**(*hash_string: str*)

firepit.woodchipper.**split_parent_image**(*abs_name: str*)

firepit.woodchipper.**split_reg_key_value**(*path: str*)

firepit.woodchipper.**to_action_code**(*event_id*)

Convert windows event ID to x-oca-event action and code

firepit.woodchipper.**to_cat_list**(*category*)

firepit.woodchipper.**to_payload_bin**(*value*)

firepit.woodchipper.**to_protocol**(*value*)

5.1.18 Module contents

Top-level package for STIX Columnar Storage.

firepit.**get_storage**(*url*, *session_id=None*)

Get a storage object for firepit. *url* will determine the type; a file path means sqlite3. *session_id* is used in the case of postgresql to partition your data.

CHAPTER

SIX

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

6.1 Types of Contributions

6.1.1 Report Bugs

Report bugs at https://github.com/opencybersecurityalliance/firepit/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

6.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

6.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

6.1.4 Write Documentation

firepit could always use more documentation, whether as part of the official firepit docs, in docstrings, or even on the web in blog posts, articles, and such.

6.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/opencybersecurityalliance/firepit/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

6.2 Get Started!

Ready to contribute? Here's how to set up *firepit* for local development.

1. Fork the *firepit* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/firepit.git
```

3. Install your local copy into a virtualenv. Assuming you have pyenv installed, this is how you set up your fork for local development:

```
$ cd firepit/
$ pyenv virtualenv python-3.9.2 firepit
$ pyenv local firepit
$ make setup
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass linting and the tests (this will be done automatically with a git pre-commit hook):

```
$ make lint
$ make test
```

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.
8. Request a review from one of the maintainers.

6.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.8, 3.9, 3.10, and 3.11. There is a GitHub workflow that will check for this automatically.

All contributions must be covered by a Contributor's License Agreement (CLA) and Entity CLA (if you are contributing on behalf of your employer). You will get a prompt to sign CLA when you submit your first PR.

6.4 Tips

To run a subset of tests:

```
$ pytest tests/test_storage.py
```

Or:

```
$ pytest -k test_something...
```

6.5 Releasing

A reminder for the maintainers on how to release. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```

CHAPTER

SEVEN

CREDITS

7.1 Development Lead

- Xiaokui Shu
- Paul Coccoli

7.2 Contributors

- Raymund Lin

CHAPTER

EIGHT

HISTORY

8.1 2.3.0 (2022-06-15)

- Added query.BinnedColumn so you can group by time buckets

8.2 2.2.0 (2022-06-08)

- Better STIX extension property support - Add a new *__columns* "private" table to store mapping from object path to column name - New path/prop metadata functions to supply metadata about STIX properties
- Improved STIX `process` "deterministic" `id` generation - Use a unique ID from extension properties, if found - Use related `x-oca-asset` hostname or ID if available

8.3 2.1.0 (2022-05-18)

- Add `splint convert` command to convert some logs files to STIX bundles

8.4 2.0.0 (2022-04-01)

- Use a "normalized" SQL database
- Initial STIX 2.1 support

8.5 1.3.0 (2021-10-04)

New assign_query API, minor query API improvements

- new way to create views via assign_query
- can now init a Query with a list instead of calling append
- Some SQL injection protection in query classes

8.6 1.2.0 (2021-08-18)

- Better support for grouped data

8.7 1.1.0 (2021-07-18)

- First stable release
- Concurrency fixes in `cache()`

8.8 1.0.0 (2021-05-18)

- First release on PyPI.

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[STIX-v2_1] STIX Version 2.1. Edited by Bret Jordan, Rich Piazza, and Trey Darley. 10 June 2021. OASIS Standard. https://docs.oasis-open.org/cti/stix/v2.1/os/stix-v2.1-os.html. Latest stage: https://docs.oasis-open.org/cti/stix/v2.1/stix-v2.1.html.

PYTHON MODULE INDEX

f

INDEX

G

I

J

L

M

N

O

P

Q

R

S

T

U

V

Z