ZLogging

Release 0.1.2

Jarry Shaw

Jul 16, 2022

CONTENTS

The `ZLogging` module provides an easy-to-use bridge between the logging framework of the well-known Bro/Zeek Network Security Monitor (IDS).

> As of version 3.0, the `Bro` project has been officially renamed to `Zeek`.[1]

It was originally developed and derived from the `BroAPT` project, which is an APT detection framework based on the Bro/Zeek IDS and extended with highly customised and customisable Python wrappers.

[1] https://blog.zeek.org/2018/10/renaming-bro-project_11.html

CHAPTER

ONE

LOG LOADERS

1.1 Functional Interfaces

1.1.1 General APIs

zlogging.loader.**parse**(*filename*, **args*, ***kwargs*)

Parse Bro/Zeek log file.

Parameters

- **filename** (*PathLike[str]*) – Log file name.
- ***args** (*Any*) – See *parse_json()* and *parse_ascii()* for more information.
- ****kwargs** (*Any*) – See *parse_json()* and *parse_ascii()* for more information.

Returns
The parsed JSON log data.

Raises
ParserError – If the format of the log file is unknown.

Return type
Union[*JSONInfo*, *ASCIIInfo*]

zlogging.loader.**loads**(*data*, **args*, ***kwargs*)

Parse Bro/Zeek log string.

Parameters

- **data** (Union[str, bytes]) – Log string as binary or encoded string.
- ***args** – See *loads_json()* and *loads_ascii()* for more information.
- ****kwargs** – See *loads_json()* and *loads_ascii()* for more information.

Return type
Union[*JSONInfo*, *ASCIIInfo*]

Returns
The parsed JSON log data.

Raises
ParserError – If the format of the log file is unknown.

zlogging.loader.**load**(*file*, **args*, ***kwargs*)

Parse Bro/Zeek log file.

Parameters

- **file** (BufferedReader) – Log file object opened in binary mode.
- ***args** – See *load_json()* and *load_ascii()* for more information.
- ****kwargs** – See *load_json()* and *load_ascii()* for more information.

Return type
Union[*JSONInfo*, *ASCIIInfo*]

Returns
The parsed JSON log data.

Raises
ParserError – If the format of the log file is unknown.

1.1.2 ASCII Format

zlogging.loader.**parse_ascii**(*filename*, *parser=None*, *type_hook=None*, *enum_namespaces=None*, *bare=False*, **args*, ***kwargs*)

Parse ASCII log file.

Parameters

- **filename** (*PathLike[str]*) – Log file name.
- **parser** (*Optional[Type[ASCIIParser]]*) – Parser class.
- **type_hook** (*Optional[dict[str, Type[BaseType]]]*) – Bro/Zeek type parser hooks. User may customise subclasses of *BaseType* to modify parsing behaviours.
- **enum_namespaces** (*Optional[list[str]]*) – Namespaces to be loaded.
- **bare** (*bool*) – If True, do not load zeek namespace by default.
- ***args** (*Any*) – Arbitrary positional arguments.
- ****kwargs** (*Any*) – Arbitrary keyword arguments.

Returns
The parsed ASCII log data.

Return type
ASCIIInfo

zlogging.loader.**loads_ascii**(*data*, *parser=None*, *type_hook=None*, *enum_namespaces=None*, *bare=False*, **args*, ***kwargs*)

Parse ASCII log string.

Parameters

- **data** (*AnyStr*) – Log string as binary or encoded string.
- **parser** (*Optional[Type[ASCIIParser]]*) – Parser class.
- **type_hook** (*Optional[dict[str, Type[BaseType]]]*) – Bro/Zeek type parser hooks. User may customise subclasses of *BaseType* to modify parsing behaviours.
- **enum_namespaces** (*Optional[list[str]]*) – Namespaces to be loaded.
- **bare** (*bool*) – If True, do not load zeek namespace by default.
- ***args** (*Any*) – Arbitrary positional arguments.

- ****kwargs** (*Any*) – Arbitrary keyword arguments.

Returns
The parsed ASCII log data.

Return type
ASCIIInfo

zlogging.loader.**load_ascii**(*file*, *parser=None*, *type_hook=None*, *enum_namespaces=None*, *bare=False*, **args*, ***kwargs*)

Parse ASCII log file.

Parameters

- **file** (*BinaryFile*) – Log file object opened in binary mode.
- **parser** (*Optional[Type[ASCIIParser]]*) – Parser class.
- **type_hook** (*Optional[dict[str, Type[BaseType]]]*) – Bro/Zeek type parser hooks. User may customise subclasses of *BaseType* to modify parsing behaviours.
- **enum_namespaces** (*Optional[list[str]]*) – Namespaces to be loaded.
- **bare** (*bool*) – If `True`, do not load `zeek` namespace by default.
- ***args** (*Any*) – Arbitrary positional arguments.
- ****kwargs** (*Any*) – Arbitrary keyword arguments.

Returns
The parsed ASCII log data.

Return type
ASCIIInfo

1.1.3 JSON Format

zlogging.loader.**parse_json**(*filename*, *parser=None*, *model=None*, **args*, ***kwargs*)

Parse JSON log file.

Parameters

- **filename** (*PathLike[str]*) – Log file name.
- **parser** (*Optional[Type[JSONParser]]*) – Parser class.
- **model** (*Optional[Type[Model]]*) – Field declarations for *JSONParser*, as in JSON logs the field typing information are omitted by the Bro/Zeek logging framework.
- ***args** (*Any*) – Arbitrary positional arguments.
- ****kwargs** (*Any*) – Arbitrary keyword arguments.

Returns
The parsed JSON log data.

Return type
JSONInfo

zlogging.loader.**loads_json**(*data*, *parser=None*, *model=None*, **args*, ***kwargs*)

Parse JSON log string.

Parameters

- **data** (Union[str, bytes]) – Log string as binary or encoded string.
- **parser** (Optional[Type[*JSONParser*]]) – Parser class.
- **model** (Optional[Type[*Model*]]) – Field declarations for *JSONParser*, as in JSON logs the field typing information are omitted by the Bro/Zeek logging framework.
- ***args** – Arbitrary positional arguments.
- ****kwargs** – Arbitrary keyword arguments.

Return type
JSONInfo

Returns
The parsed JSON log data.

zlogging.loader.**load_json**(*file*, *parser=None*, *model=None*, **args*, ***kwargs*)

Parse JSON log file.

Parameters

- **file** (BufferedReader) – Log file object opened in binary mode.
- **parser** (Optional[Type[*JSONParser*]]) – Parser class.
- **model** (Optional[Type[*Model*]]) – Field declarations for *JSONParser*, as in JSON logs the field typing information are omitted by the Bro/Zeek logging framework.
- ***args** – Arbitrary positional arguments.
- ****kwargs** – Arbitrary keyword arguments.

Return type
JSONInfo

Returns
The parsed JSON log data.

1.2 Predefined Loaders

class zlogging.loader.**ASCIIParser**(*type_hook=None*, *enum_namespaces=None*, *bare=False*)

Bases: *BaseParser*

ASCII log parser.

Parameters

- **type_hook** (*Optional[dict[str, Type[BaseType]]]*) – Bro/Zeek type parser hooks. User may customise subclasses of *BaseType* to modify parsing behaviours.
- **enum_namespaces** (*Optional[list[str]]*) – Namespaces to be loaded.
- **bare** (*bool*) – If True, do not load zeek namespace by default.

property **format**: Literal['ascii']

Log file format.

Return type
Literal[‘ascii’]

enum_namespaces: list[str]

Namespaces to be loaded.

bare: bool

If True, do not load zeek namespace by default.

parse_file(*file*, *model=None*)

Parse log file.

Parameters

- **file** (BufferedReader) – Log file object opened in binary mode.
- **model** (Optional[Type[*Model*]]) – Field declrations of current log. This parameter is only kept for API compatibility with its base class BaseLoader, and will **NOT** be used at runtime.

Return type
ASCIIInfo

Returns
The parsed log as a *Model* per line.

Warns
ASCIIParserWarning – If the ASCII log file exited with error, see *ASCIIInfo.exit_with_error* for more information.

parse_line(*line*, *lineno=0*, *model=None*, *separator=b'\\t'*, *parser=None*)

Parse log line as one-line record.

Parameters

- **line** (*bytes*) – A simple line of log.
- **lineno** (*Optional[int]*) – Line number of current line.
- **model** (*Optional[Type[Model]]*) – Field declrations of current log.
- **separator** (*Optional[bytes]*) – Data separator.
- **parser** (*Optional[list[tuple[str, BaseType]]]*) – Field data type parsers.

Returns
The parsed log as a plain dict.

Raises
ASCIIParserError – If parser is not provided; or failed to serialise line as ASCII.

Return type
Model

class zlogging.loader.**JSONParser**(*model=None*)

Bases: *BaseParser*

JSON log parser.

Parameters
model (Optional[Type[*Model*]]) – Field declrations for *JSONParser*, as in JSON logs the field typing information are omitted by the Bro/Zeek logging framework.

Warns
JSONParserWarning – If model is not specified.

property format: Literal['json']

Log file format.

Return type
Literal['json']

model: Optional[Type[Model]]

~zlogging.loader.JSONParser, as in JSON logs the field typing information are omitted by the Bro/Zeek logging framework.

Type
Field declrations for

Type
class

parse_file(*file*, *model=None*)

Parse log file.

Parameters

- **file** (BufferedReader) – Log file object opened in binary mode.
- **model** (Optional[Type[*Model*]]) – Field declrations of current log.

Return type
JSONInfo

Returns
The parsed log as a *Model* per line.

parse_line(*line*, *lineno=0*, *model=None*)

Parse log line as one-line record.

Parameters

- **line** (bytes) – A simple line of log.
- **lineno** (Optional[int]) – Line number of current line.
- **model** (Optional[Type[*Model*]]) – Field declrations of current log.

Return type
Model

Returns
The parsed log as a plain *Model*.

Raises
JSONParserError – If failed to serialise the line from JSON.

1.3 Abstract Base Loader

class zlogging.loader.BaseParser

Bases: object

Basic log parser.

abstract property format: str

Log file format.

Return type
str

parse(*filename*, *model=None*)

Parse log file.

Parameters

- **filename** (*PathLike[str]*) – Log file name.
- **model** (*Optional[Type[Model]]*) – Field declrations of current log.

Returns
The parsed log as an *ASCIIInfo* or *JSONInfo*.

Return type
Info

abstract parse_file(*file*, *model=None*)

Parse log file.

Parameters

- **file** (BufferedReader) – Log file object opened in binary mode.
- **model** (Optional[Type[*Model*]]) – Field declrations of current log.

Returns
The parsed log as a *Model* per line.

Return type
Info

abstract parse_line(*line*, *lineno=0*, *model=None*)

Parse log line as one-line record.

Parameters

- **line** (bytes) – A simple line of log.
- **lineno** (Optional[int]) – Line number of current line.
- **model** (Optional[Type[*Model*]]) – Field declrations of current log.

Return type
Model

Returns
The parsed log as a plain *Model*.

load(*file*)

Parse log file.

Parameters
file (BufferedReader) – Log file object opened in binary mode.

Returns
The parsed log as a *Model* per line.

Return type
Info

loads(*line*, *lineno=0*)

Parse log line as one-line record.

Parameters

- **line** (bytes) – A simple line of log.
- **lineno** (Optional[int]) – Line number of current line.

Return type
Model

Returns
The parsed log as a plain *Model*.

CHAPTER

TWO

LOG DUMPERS

2.1 Functional Interfaces

2.1.1 General APIs

zlogging.dumper.**write**(*data*, *filename*, *format*, **args*, ***kwargs*)

Write Bro/Zeek log file.

Parameters

- **data** (*Iterable[Model]*) – Log records as an Iterable of *Model* per line.
- **filename** (*PathLike[str]*) – Log file name.
- **format** (*str*) – Log format.
- ***args** (*Any*) – See *write_json()* and *write_ascii()* for more information.
- ****kwargs** (*Any*) – See *write_json()* and *write_ascii()* for more information.

Raises

WriterFormatError – If `format` is not supported.

Return type

None

zlogging.dumper.**dumps**(*data*, *format*, **args*, ***kwargs*)

Write Bro/Zeek log string.

Parameters

- **data** (Iterable[*Model*]) – Log records as an Iterable of *Model* per line.
- **format** (str) – Log format.
- ***args** – See *dumps_json()* and *dumps_ascii()* for more information.
- ****kwargs** – See *dumps_json()* and *dumps_ascii()* for more information.

Raises

WriterFormatError – If `format` is not supported.

Return type

str

zlogging.dumper.**dump**(*data*, *file*, *format*, **args*, ***kwargs*)

Write Bro/Zeek log file.

Parameters

- **data** (Iterable[*Model*]) – Log records as an Iterable of *Model* per line.
- **format** (str) – Log format.
- **file** (TextIOWrapper) – Log file object opened in text mode.
- ***args** – See *dump_json()* and *dump_ascii()* for more information.
- ****kwargs** – See *dump_json()* and *dump_ascii()* for more information.

Raises
WriterFormatError – If format is not supported.

Return type
None

2.1.2 ASCII Format

zlogging.dumper.**write_ascii**(*data*, *filename*, *writer=None*, *separator=None*, *empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***kwargs*)

Write ASCII log file.

Parameters

- **data** (*Iterable[Model]*) – Log records as an Iterable of *Model* per line.
- **filename** (*PathLike[str]*) – Log file name.
- **writer** (*Optional[Type[ASCIIWriter]]*) – Writer class.
- **separator** (*Optional[AnyStr]*) – Field separator when writing log lines.
- **empty_field** (*Optional[AnyStr]*) – Placeholder for empty field.
- **unset_field** (*Optional[AnyStr]*) – Placeholder for unset field.
- **set_separator** (*Optional[AnyStr]*) – Separator for set/vector fields.
- ***args** (*Any*) – Arbitrary positional arguments.
- ****kwargs** (*Any*) – Arbitrary keyword arguments.

Return type
None

zlogging.dumper.**dumps_ascii**(*data=None*, *writer=None*, *separator=None*, *empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***kwargs*)

Write ASCII log string.

Parameters

- **data** (Optional[Iterable[*Model*]]) – Log records as an Iterable of *Model* per line.
- **writer** (Optional[Type[*ASCIIWriter*]]) – Writer class.
- **separator** (Union[str, bytes, None]) – Field separator when writing log lines.
- **empty_field** (Union[str, bytes, None]) – Placeholder for empty field.
- **unset_field** (Union[str, bytes, None]) – Placeholder for unset field.
- **set_separator** (Union[str, bytes, None]) – Separator for set/vector fields.
- ***args** – Arbitrary positional arguments.
- ****kwargs** – Arbitrary keyword arguments.

Return type
str

Returns
The JSON log string.

zlogging.dumper.**dump_ascii**(*data*, *file*, *writer=None*, *separator=None*, *empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***kwargs*)

Write ASCII log file.

Parameters

- **data** (Iterable[*Model*]) – Log records as an Iterable of *Model* per line.
- **file** (TextIOWrapper) – Log file object opened in text mode.
- **writer** (Optional[Type[*ASCIIWriter*]]) – Writer class.
- **separator** (Union[str, bytes, None]) – Field separator when writing log lines.
- **empty_field** (Union[str, bytes, None]) – Placeholder for empty field.
- **unset_field** (Union[str, bytes, None]) – Placeholder for unset field.
- **set_separator** (Union[str, bytes, None]) – Separator for set/vector fields.
- ***args** – Arbitrary positional arguments.
- ****kwargs** – Arbitrary keyword arguments.

Return type
None

2.1.3 JSON Format

zlogging.dumper.**write_json**(*data*, *filename*, *writer=None*, *encoder=None*, **args*, ***kwargs*)

Write JSON log file.

Parameters

- **data** (*Iterable[Model]*) – Log records as an Iterable of *Model* per line.
- **filename** (*PathLike[str]*) – Log file name.
- **writer** (*Optional[Type[JSONWriter]]*) – Writer class.
- **encoder** (*Optional[Type[JSONEncoder]]*) – JSON encoder class.
- ***args** (*Any*) – Arbitrary positional arguments.
- ****kwargs** (*Any*) – Arbitrary keyword arguments.

Return type
None

zlogging.dumper.**dumps_json**(*data=None*, *writer=None*, *encoder=None*, **args*, ***kwargs*)

Write JSON log string.

Parameters

- **data** (Optional[Iterable[*Model*]]) – Log records as an Iterable of *Model* per line.
- **writer** (Optional[Type[*JSONWriter*]]) – Writer class.
- **encoder** (Optional[Type[JSONEncoder]]) – JSON encoder class.

- ***args** – Arbitrary positional arguments.
- ****kwargs** – Arbitrary keyword arguments.

Return type
str

Returns
The JSON log string.

zlogging.dumper.**dump_json**(*data*, *file*, *writer=None*, *encoder=None*, **args*, ***kwargs*)

Write JSON log file.

Parameters

- **data** (Iterable[*Model*]) – Log records as an Iterable of *Model* per line.
- **file** (TextIOWrapper) – Log file object opened in text mode.
- **writer** (Optional[Type[*JSONWriter*]]) – Writer class.
- **encoder** (Optional[Type[JSONEncoder]]) – JSON encoder class.
- ***args** – Arbitrary positional arguments.
- ****kwargs** – Arbitrary keyword arguments.

Return type
None

2.2 Predefined Dumpers

class zlogging.dumper.**ASCIIWriter**(*separator=None*, *empty_field=None*, *unset_field=None*, *set_separator=None*)

Bases: *BaseWriter*

ASCII log writer.

Parameters

- **separator** (Union[str, bytes, None]) – Field separator when writing log lines.
- **empty_field** (Union[str, bytes, None]) – Placeholder for empty field.
- **unset_field** (Union[str, bytes, None]) – Placeholder for unset field.
- **set_separator** (Union[str, bytes, None]) – Separator for set/vector fields.

property format: str

Log file format.

Return type
str

separator: bytes

Field separator when writing log lines.

empty_field: bytes

Placeholder for empty field.

unset_field: bytes

Placeholder for unset field.

set_separator: *bytes*

Separator for `set`/`vector` fields.

write_file(*file*, *data*)

Write log file.

Parameters

- **file** (`TextIOWrapper`) – Log file object opened in text mode.
- **data** (`Iterable`[*Model*]) – Log records as an `Iterable` of *Model* per line.

Return type
`int`

Returns
The file offset after writing.

write_line(*file*, *data*, *lineno=0*)

Write log line as one-line record.

Parameters

- **file** (`TextIOWrapper`) – Log file object opened in text mode.
- **data** (*Model*) – Log record.
- **lineno** (`Optional`[`int`]) – Line number of current line.

Return type
`int`

Returns
The file offset after writing.

Raises
ASCIIWriterError – If failed to serialise `data` as ASCII.

write_head(*file*, *data=None*)

Write header fields of ASCII log file.

Parameters

- **file** (`TextIOWrapper`) – Log file object opened in text mode.
- **data** (`Optional`[*Model*]) – Log record.

Return type
`int`

Returns
The file offset after writing.

write_tail(*file*)

Write trailing fields of ASCII log file.

Parameters
file (`TextIOWrapper`) – Log file object opened in text mode.

Return type
`int`

Returns
The file offset after writing.

dump_file(*data=None*, *name=None*)

Serialise records to a log line.

Parameters

- **data** (Optional[Iterable[*Model*]]) – Log records as an Iterable of *Model* per line.
- **name** (Optional[str]) – Log file name.

Return type
str

Returns
The converted log string.

dump_line(*data*, *lineno=0*)

Serialise one-line record to a log line.

Parameters

- **data** (*Model*) – Log record.
- **lineno** (Optional[int]) – Line number of current line.

Return type
str

Returns
The converted log string.

Raises
ASCIIWriterError – If failed to serialise data as ASCII.

dump_head(*data=None*, *name=None*)

Serialise header fields of ASCII log file.

Parameters

- **data** (Optional[*Model*]) – Log record.
- **name** (Optional[str]) – Log file name.

Return type
str

Returns
The converted log string.

dump_tail()

Serialise trailing fields of ASCII log file.

Return type
str

Returns
The converted log string.

class zlogging.dumper.**JSONWriter**(*encoder=None*)

Bases: *BaseWriter*

JSON log writer.

Parameters
encoder (Optional[Type[JSONEncoder]]) – JSON encoder class.

property format: Literal['json']

Log file format.

Return type
Literal['json']

encoder: Type[JSONEncoder]

JSON encoder class.

write_file(*file*, *data*)

Write log file.

Parameters

- **file** (TextIOWrapper) – Log file object opened in text mode.
- **data** (Iterable[*Model*]) – Log records as an Iterable of *Model* per line.

Return type
int

Returns
The file offset after writing.

write_line(*file*, *data*, *lineno=0*)

Write log line as one-line record.

Parameters

- **file** (TextIOWrapper) – Log file object opened in text mode.
- **data** (*Model*) – Log record.
- **lineno** (Optional[int]) – Line number of current line.

Return type
int

Returns
The file offset after writing.

Raises
JSONWriterError – If failed to serialise data as JSON.

dump_file(*data=None*)

Serialise records to a log line.

Parameters
data (Optional[Iterable[*Model*]]) – Log records as an Iterable of *Model* per line.

Return type
str

Returns
The converted log string.

dump_line(*data*, *lineno=0*)

Serialise one-line record to a log line.

Parameters

- **data** (*Model*) – Log record.
- **lineno** (Optional[int]) – Line number of current line.

Return type
str

Returns
The converted log string.

Raises
JSONWriterError – If failed to serialise `data` as JSON.

2.3 Abstract Base Dumper

class zlogging.dumper.**BaseWriter**

Bases: object

Basic log writer.

abstract property format: str

Log file format.

Return type
str

write(*filename*, *data*)

Write log file.

Parameters

- **filename** (*PathLike[str]*) – Log file name.
- **data** (*Iterable[Model]*) – Log records as an Iterable of *Model* per line.

Returns
The file offset after writing.

Return type
int

abstract write_file(*file*, *data*)

Write log file.

Parameters

- **file** (TextIOWrapper) – Log file object opened in text mode.
- **data** (Iterable[*Model*]) – Log records as an Iterable of *Model* per line.

Return type
int

Returns
The file offset after writing.

abstract write_line(*file*, *data*, *lineno=0*)

Write log line as one-line record.

Parameters

- **file** (TextIOWrapper) – Log file object opened in text mode.
- **data** (*Model*) – Log record.
- **lineno** (Optional[int]) – Line number of current line.

Return type
int

Returns
The file offset after writing.

abstract dump_file(*data*)

Serialise records to a log line.

Parameters
data (Iterable[*Model*]) – Log records as an Iterable of *Model* per line.

Return type
str

Returns
The converted log string.

abstract dump_line(*data*, *lineno=0*)

Serialise one-line record to a log line.

Parameters

- **data** (*Model*) – Log record.
- **lineno** (Optional[int]) – Line number of current line.

Return type
str

Returns
The converted log string.

dump(*data*, *file*)

Write log file.

Parameters

- **data** (Iterable[*Model*]) – Log records as an Iterable of *Model* per line.
- **file** (TextIOWrapper) – Log file object opened in text mode.

Return type
int

Returns
The file offset after writing.

dumps(*data*)

Serialise records to a log line.

Parameters
data (Iterable[*Model*]) – Log records as an Iterable of *Model* per line.

Return type
str

Returns
The converted log string.

CHAPTER

THREE

DATA MODEL

class zlogging.model.**Model**(**args*, ***kwargs*)

Bases: `object`

Log data model.

Parameters

- ***args** (*Any*) – Arbitrary positional arguments.
- ****kwargs** (*Any*) – Arbitrary keyword arguments.

Warns

BroDeprecationWarning – Use of `bro_*` type annotations.

Raises

- **ModelValueError** – In case of inconsistency between field data types, or values of `unset_field`, `empty_field` and `set_separator`.
- **ModelTypeError** – Wrong parameters when initialisation.

Return type

Model

Note: Customise the *Model.__post_init__* method in your subclassed data model to implement your own ideas.

Example

Define a custom log data model using the prefines Bro/Zeek data types, or subclasses of *BaseType*:

```
class MyLog(Model):
    field_one = StringType()
    field_two = SetType(element_type=PortType)
```

Or you may use type annotations as **PEP 484** introduced when declaring data models. All available type hints can be found in `zlogging.typing`:

```
class MyLog(Model):
    field_one: zeek_string
    field_two: zeek_set[zeek_port]
```

However, when mixing annotations and direct assignments, annotations will take proceedings, i.e. the *Model* class shall process first annotations then assignments. Should there be any conflicts, `ModelError` will be raised.

See also:

See *expand_typing()* for more information about processing the fields.

property fields: OrderedDict[str, Union[*_SimpleType*, *_GenericType*]]

Fields of the data model.

property unset_field: bytes

Placeholder for empty field.

Return type
bytes

property empty_field: bytes

Placeholder for unset field.

Return type
bytes

property set_separator: bytes

Separator for set/vector fields.

Return type
bytes

__post_init__()

Post-processing customisation.

Return type
None

__call__(*format*)

Serialise data model with given format.

Parameters
format (str) – Serialisation format.

Return type
Any

Returns
The serialised data.

Raises
ModelFormatError – If `format` is not supproted, i.e. `Mode.to{format}()` does not exist.

tojson()

Serialise data model as JSON log format.

Returns
An `OrderedDict` mapping each field and serialised JSON serialisable data.

Return type
OrderedDict[str, Any]

toascii()

Serialise data model as ASCII log format.

Returns
An `OrderedDict` mapping each field and serialised text data.

Return type
OrderedDict[str, str]

asdict(*dict_factory=None*)

Convert data model as a dictionary mapping field names to field values.

Parameters
dict_factory (*Optional[Type[dict]]*) – If given, `dict_factory` will be used instead of built-in `dict`.

Returns
A dictionary mapping field names to field values.

Return type
dict[str, Any]

astuple(*tuple_factory=None*)

Convert data model as a tuple of field values.

Parameters
tuple_factory (*Optional[Type[tuple]]*) – If given, `tuple_factory` will be used instead of built-in `namedtuple`.

Returns
A tuple of field values.

Return type
tuple[Any, ...]

zlogging.model.**new_model**(*name*, ***fields*)

Create a data model dynamically with the appropriate fields.

Parameters

- **name** (str) – data model name
- ****fields** – defined fields of the data model

Return type
Type[*Model*]

Returns
Created data model.

Examples

Typically, we define a data model by subclassing the *Model* class, as following:

```
class MyLog(Model):
    field_one = StringType()
    field_two = SetType(element_type=PortType)
```

when defining dynamically with *new_model()*, the definition above can be rewrote to:

```
MyLog = new_model('MyLog', field_one=StringType(), field_two=SetType(element_
↪type=PortType))
```

CHAPTER

FOUR

DATA TYPES

4.1 Bro/Zeek Types

4.1.1 Boolean

class zlogging.types.**BoolType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***kwargs*)

Bases: *_SimpleType*

Bro/Zeek `bool` data type.

Parameters

- **empty_field** (Union[str, bytes, None]) – Placeholder for empty field.
- **unset_field** (Union[str, bytes, None]) – Placeholder for unset field.
- **set_separator** (Union[str, bytes, None]) – Separator for `set`/`vector` fields.
- ***args** – Arbitrary positional arguments.
- ****kwargs** – Arbitrary keyword arguments.

property python_type: Type[bool]

Corresponding Python type annotation.

Return type
Type[bool]

property zeek_type: Literal['bool']

Corresponding Zeek type name.

Return type
Literal['bool']

parse(*data: Literal['T', b'T']*) → Literal[True]

parse(*data: Literal['F', b'F']*) → Literal[False]

parse(*data: AnyStr*) → Optional[bool]

Parse `data` from string.

Parameters
data (Union[str, bytes, bool]) – raw data

Return type
Optional[bool]

Returns
The parsed boolean data. If `data` is *unset*, `None` will be returned.

Raises
ZeekValueError – If `data` is NOT *unset* and NOT `T` (`True`) nor `F` (`False`) in Bro/Zeek script language.

tojson(*data: Literal[True]*) → Literal[True]

tojson(*data: Literal[False]*) → Literal[False]

tojson(*data: None*) → None

Serialize `data` as JSON log format.

Parameters
data (`Optional`[`bool`]) – raw data

Return type
`Optional`[`bool`]

Returns
The JSON serialisable boolean data.

toascii(*data: Literal[True]*) → Literal['T']

toascii(*data: Literal[False]*) → Literal['F']

toascii(*data: None*) → str

Serialize `data` as ASCII log format.

Parameters
data (`Optional`[`bool`]) – raw data

Returns
`T` if `True`, `F` if `False`.

Return type
str

empty_field: `bytes`

Placeholder for empty field.

unset_field: `bytes`

Placeholder for unset field.

set_separator: `bytes`

Separator for `set`/`vector` fields.

4.1.2 Numeric Types

class zlogging.types.**CountType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***kwargs*)

Bases: *_SimpleType*

Bro/Zeek `count` data type.

Parameters

- **empty_field** (`Union`[`str`, `bytes`, `None`]) – Placeholder for empty field.
- **unset_field** (`Union`[`str`, `bytes`, `None`]) – Placeholder for unset field.
- **set_separator** (`Union`[`str`, `bytes`, `None`]) – Separator for `set`/`vector` fields.
- ***args** – Arbitrary positional arguments.
- ****kwargs** – Arbitrary keyword arguments.

property python_type: Type[c_ulong]

Corresponding Python type annotation.

Return type
Type[c_ulong]

property zeek_type: Literal['count']

Corresponding Zeek type name.

Return type
Literal['count']

parse(*data: AnyStr*) → Optional[uint64]

parse(*data: int*) → uint64

parse(*data: uint64*) → uint64

Parse `data` from string.

Parameters
data (Union[str, bytes, int, c_ulong]) – raw data

Return type
Optional[c_ulong]

Returns
The parsed numeral data. If `data` is *unset*, `None` will be returned.

tojson(*data: uint64*) → int

tojson(*data: None*) → None

Serialize `data` as JSON log format.

Parameters
data (Optional[c_ulong]) – raw data

Returns
The JSON serialisable numeral data.

Return type
int

toascii(*data*)

Serialize `data` as ASCII log format.

Parameters
data (Optional[c_ulong]) – raw data

Returns
The ASCII representation of numeral data.

Return type
str

empty_field: bytes

Placeholder for empty field.

unset_field: bytes

Placeholder for unset field.

set_separator: bytes

Separator for `set`/`vector` fields.

class zlogging.types.**IntType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***kwargs*)

Bases: *_SimpleType*

Bro/Zeek `int` data type.

Parameters

- **empty_field** (Union[str, bytes, None]) – Placeholder for empty field.
- **unset_field** (Union[str, bytes, None]) – Placeholder for unset field.
- **set_separator** (Union[str, bytes, None]) – Separator for `set`/`vector` fields.
- ***args** – Arbitrary positional arguments.
- ****kwargs** – Arbitrary keyword arguments.

property **python_type**: Type[c_long]

Corresponding Python type annotation.

Return type
Type[c_long]

property **zeek_type**: Literal['int']

Corresponding Zeek type name.

Return type
Literal['int']

parse(*data: AnyStr*) → Optional[int64]

parse(*data: int*) → int64

parse(*data: int64*) → int64

Parse `data` from string.

Parameters
data (Union[str, bytes, int, c_long]) – raw data

Return type
Optional[c_long]

Returns
The parsed numeral data. If `data` is *unset*, `None` will be returned.

tojson(*data: int64*) → int

tojson(*data: None*) → None

Serialize `data` as JSON log format.

Parameters
data (Optional[c_long]) – raw data

Returns
The JSON serialisable numeral data.

Return type
int

toascii(*data*)

Serialize `data` as ASCII log format.

Parameters
data (Optional[c_long]) – raw data

Returns
The ASCII representation of numeral data.

Return type
str

empty_field: bytes
Placeholder for empty field.

unset_field: bytes
Placeholder for unset field.

set_separator: bytes
Separator for set/vector fields.

class zlogging.types.**DoubleType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***kwargs*)

Bases: *_SimpleType*

Bro/Zeek double data type.

Parameters
- **empty_field** (Union[str, bytes, None]) – Placeholder for empty field.
- **unset_field** (Union[str, bytes, None]) – Placeholder for unset field.
- **set_separator** (Union[str, bytes, None]) – Separator for set/vector fields.
- ***args** – Arbitrary positional arguments.
- ****kwargs** – Arbitrary keyword arguments.

property python_type: Type[Decimal]
Corresponding Python type annotation.

Return type
Type[Decimal]

property zeek_type: Literal['double']
Corresponding Zeek type name.

Return type
Literal['double']

parse(*data: AnyStr*) → Optional[Decimal]

parse(*data: Union[int, float]*) → Decimal

parse(*data: Decimal*) → Decimal
Parse data from string.

Parameters
data (Union[str, bytes, int, float, Decimal]) – raw data

Return type
Optional[Decimal]

Returns
The parsed numeral data. If data is *unset*, None will be returned.

tojson(*data: Decimal*) → float

tojson(*data: None*) → None

Serialize `data` as JSON log format.

Parameters
data (`Optional[Decimal]`) – raw data

Returns
The JSON serialisable numeral data.

Return type
float

toascii(*data*)

Serialize `data` as ASCII log format.

Parameters
data (`Optional[Decimal]`) – raw data

Returns
The ASCII representation of numeral data.

Return type
str

empty_field: `bytes`

Placeholder for empty field.

unset_field: `bytes`

Placeholder for unset field.

set_separator: `bytes`

Separator for `set`/`vector` fields.

4.1.3 Time Types

class zlogging.types.**TimeType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***kwargs*)

Bases: *_SimpleType*

Bro/Zeek `time` data type.

Parameters

- **empty_field** (`Union[str, bytes, None]`) – Placeholder for empty field.
- **unset_field** (`Union[str, bytes, None]`) – Placeholder for unset field.
- **set_separator** (`Union[str, bytes, None]`) – Separator for `set`/`vector` fields.
- ***args** – Arbitrary positional arguments.
- ****kwargs** – Arbitrary keyword arguments.

property python_type: `Type[datetime]`

Corresponding Python type annotation.

Type
Any

Return type
`Type[datetime]`

property zeek_type: Literal['time']

Corresponding Zeek type name.

Type
str

Return type
Literal['time']

parse(*data: AnyStr*) → Optional[DateTimeType]

parse(*data: float*) → DateTimeType

parse(*data: DateTimeType*) → DateTimeType

Parse `data` from string.

Parameters
data (Union[str, bytes, float, datetime]) – raw data

Return type
Optional[datetime]

Returns
The parsed numeral data. If `data` is *unset*, `None` will be returned.

tojson(*data: DateTimeType*) → float

tojson(*data: None*) → None

Serialize `data` as JSON log format.

Parameters
data (Optional[datetime]) – raw data

Returns
The JSON serialisable numeral data.

Return type
float

toascii(*data*)

Serialize `data` as ASCII log format.

Parameters
data (Optional[datetime]) – raw data

Returns
The ASCII representation of numeral data.

Return type
str

empty_field: bytes

Placeholder for empty field.

unset_field: bytes

Placeholder for unset field.

set_separator: bytes

Separator for `set`/`vector` fields.

class zlogging.types.**IntervalType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***kwargs*)

Bases: *_SimpleType*

Bro/Zeek `interval` data type.

Parameters

- **empty_field** (Union[str, bytes, None]) – Placeholder for empty field.
- **unset_field** (Union[str, bytes, None]) – Placeholder for unset field.
- **set_separator** (Union[str, bytes, None]) – Separator for `set`/`vector` fields.
- ***args** – Arbitrary positional arguments.
- ****kwargs** – Arbitrary keyword arguments.

Variables

- **empty_field** (*bytes*) – Placeholder for empty field.
- **unset_field** (*bytes*) – Placeholder for unset field.
- **set_separator** (*bytes*) – Separator for `set`/`vector` fields.

property python_type: Type[timedelta]

Corresponding Python type annotation.

Type
Any

Return type
Type[timedelta]

property zeek_type: Literal['interval']

Corresponding Zeek type name.

Type
str

Return type
Literal['interval']

parse(*data: AnyStr*) → Optional[TimeDeltaType]

parse(*data: float*) → TimeDeltaType

parse(*data: TimeDeltaType*) → TimeDeltaType

Parse `data` from string.

Parameters
data (Union[str, bytes, float, timedelta]) – raw data

Return type
Optional[timedelta]

Returns
The parsed numeral data. If `data` is *unset*, `None` will be returned.

tojson(*data: TimeDeltaType*) → float

tojson(*data: None*) → None

Serialize `data` as JSON log format.

Parameters
data (Optional[timedelta]) – raw data

Returns
The JSON serialisable numeral data.

Return type
int

`toascii`(*data*)

Serialize `data` as ASCII log format.

Parameters
data (Optional[timedelta]) – raw data

Returns
The ASCII representation of numeral data.

Return type
str

empty_field: bytes

Placeholder for empty field.

unset_field: bytes

Placeholder for unset field.

set_separator: bytes

Separator for `set`/`vector` fields.

4.1.4 String

class zlogging.types.**StringType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***kwargs*)

Bases: *_SimpleType*

Bro/Zeek `string` data type.

Parameters

- **empty_field** (Union[str, bytes, None]) – Placeholder for empty field.
- **unset_field** (Union[str, bytes, None]) – Placeholder for unset field.
- **set_separator** (Union[str, bytes, None]) – Separator for `set`/`vector` fields.
- ***args** – Arbitrary positional arguments.
- ****kwargs** – Arbitrary keyword arguments.

property python_type: Any

Corresponding Python type annotation.

Type
Any

Return type
Any

property zeek_type: Literal['string']

Corresponding Zeek type name.

Type
str

Return type
Literal['string']

parse(*data*)

Parse `data` from string.

Parameters
data (Union[str, bytes, bytearray, memoryview]) – raw data

Return type
Optional[bytes]

Returns
The parsed string data. If `data` is *unset*, `None` will be returned.

tojson(*data: ByteString*) → str

tojson(*data: None*) → None

Serialize `data` as JSON log format.

Parameters
data (Union[bytes, bytearray, memoryview, None]) – raw data

Returns
The JSON serialisable string data encoded in ASCII.

Return type
str

toascii(*data*)

Serialize `data` as ASCII log format.

Parameters
data (Union[bytes, bytearray, memoryview, None]) – raw data

Returns
The ASCII encoded string data.

Return type
str

empty_field: bytes

Placeholder for empty field.

unset_field: bytes

Placeholder for unset field.

set_separator: bytes

Separator for `set`/`vector` fields.

4.1.5 Network Types

class zlogging.types.**PortType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***kwargs*)

Bases: *_SimpleType*

Bro/Zeek `port` data type.

Parameters

- **empty_field** (Union[str, bytes, None]) – Placeholder for empty field.
- **unset_field** (Union[str, bytes, None]) – Placeholder for unset field.

- **set_separator** (Union[str, bytes, None]) – Separator for set/vector fields.
- ***args** – Arbitrary positional arguments.
- ****kwargs** – Arbitrary keyword arguments.

empty_field: bytes

Placeholder for empty field.

unset_field: bytes

Placeholder for unset field.

set_separator: bytes

Separator for set/vector fields.

property python_type: Type[c_ushort]

Corresponding Python type annotation.

Type
Any

Return type
Type[c_ushort]

property zeek_type: Literal['port']

Corresponding Zeek type name.

Type
str

Return type
Literal['port']

parse(*data: AnyStr*) → Optional[uint16]

parse(*data: int*) → uint16

parse(*data: uint16*) → uint16

Parse data from string.

Parameters
data (Union[str, bytes, int, c_ushort]) – raw data

Return type
Optional[c_ushort]

Returns
The parsed port number. If data is *unset*, None will be returned.

tojson(*data: uint16*) → int

tojson(*data: None*) → None

Serialize data as JSON log format.

Parameters
data (Optional[c_ushort]) – raw data

Returns
The JSON serialisable port number string.

Return type
int

toascii(*data*)

Serialize `data` as ASCII log format.

Parameters
data (`Optional[c_ushort]`) – raw data

Returns
The ASCII representation of the port number.

Return type
str

class zlogging.types.**AddrType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***kwargs*)

Bases: *_SimpleType*

Bro/Zeek `addr` data type.

Parameters

- **empty_field** (`Union[str, bytes, None]`) – Placeholder for empty field.
- **unset_field** (`Union[str, bytes, None]`) – Placeholder for unset field.
- **set_separator** (`Union[str, bytes, None]`) – Separator for `set`/`vector` fields.
- ***args** – Arbitrary positional arguments.
- ****kwargs** – Arbitrary keyword arguments.

property python_type: `Any`

Corresponding Python type annotation.

Type
Any

Return type
`Any`

property zeek_type: `str`

Corresponding Zeek type name.

Type
str

Return type
`str`

empty_field: `bytes`

Placeholder for empty field.

unset_field: `bytes`

Placeholder for unset field.

set_separator: `bytes`

Separator for `set`/`vector` fields.

parse(*data: AnyStr*) → Optional[IPAddress]

parse(*data: IPAddress*) → IPAddress

Parse `data` from string.

Parameters
data (`Union[str, bytes, IPv4Address, IPv6Address]`) – raw data

Return type
Union[IPv4Address, IPv6Address, None]

Returns
The parsed IP address. If `data` is *unset*, None will be returned.

tojson(*data: IPAddress*) → str

tojson(*data: None*) → None

Serialize `data` as JSON log format.

Parameters
data (Union[IPv4Address, IPv6Address, None]) – raw data

Returns
The JSON serialisable IP address string.

Return type
str

toascii(*data*)

Serialize `data` as ASCII log format.

Parameters
data (Union[IPv4Address, IPv6Address, None]) – raw data

Returns
The ASCII representation of the IP address.

Return type
str

class zlogging.types.**SubnetType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***kwargs*)

Bases: *_SimpleType*

Bro/Zeek `subnet` data type.

Parameters

- **empty_field** (Union[str, bytes, None]) – Placeholder for empty field.
- **unset_field** (Union[str, bytes, None]) – Placeholder for unset field.
- **set_separator** (Union[str, bytes, None]) – Separator for `set`/`vector` fields.
- ***args** – Arbitrary positional arguments.
- ****kwargs** – Arbitrary keyword arguments.

empty_field: bytes

Placeholder for empty field.

unset_field: bytes

Placeholder for unset field.

set_separator: bytes

Separator for `set`/`vector` fields.

property python_type: Any

Corresponding Python type annotation.

Type
Any

Return type
Any

property zeek_type: Literal['subnet']

Corresponding Zeek type name.

Type
str

Return type
Literal['subnet']

parse(*data: AnyStr*) → Optional[IPNetwork]

parse(*data: IPNetwork*) → IPNetwork

Parse `data` from string.

Parameters
data (Union[str, bytes, IPv4Network, IPv6Network]) – raw data

Return type
Union[IPv4Network, IPv6Network, None]

Returns
The parsed IP network. If `data` is *unset*, None will be returned.

tojson(*data: IPNetwork*) → str

tojson(*data: None*) → None

Serialize `data` as JSON log format.

Parameters
data (Union[IPv4Network, IPv6Network, None]) – raw data

Returns
The JSON serialisable IP network string.

Return type
str

toascii(*data*)

Serialize `data` as ASCII log format.

Parameters
data (Union[IPv4Network, IPv6Network, None]) – raw data

Returns
The ASCII representation of the IP network.

Return type
str

4.1.6 Enumeration

class zlogging.types.**EnumType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, *namespaces=None*, *bare=False*, *enum_hook=None*, **args*, ***kwargs*)

Bases: *_SimpleType*

Bro/Zeek enum data type.

Parameters

- **empty_field** (*Optional[AnyStr]*) – Placeholder for empty field.
- **unset_field** (*Optional[AnyStr]*) – Placeholder for unset field.
- **set_separator** (*Optional[AnyStr]*) – Separator for set/vector fields.
- **namespaces** (*Optional[list[str]]*) – Namespaces to be loaded.
- **bare** (*bool*) – If True, do not load zeek namespace by default.
- **enum_hook** (*Optional[dict[str, enum.Enum]]*) – Additional enum to be included in the namespace.
- ***args** (*Any*) – Arbitrary positional arguments.
- ****kwargs** (*Any*) – Arbitrary keyword arguments.

property **python_type**: Any

Corresponding Python type annotation.

Type
Any

Return type
Any

property **zeek_type**: str

Corresponding Zeek type name.

Type
str

Return type
str

enum_namespaces: dict[str, enum.Enum]

Namespaces to be loaded.

parse(*data: AnyStr*) → Optional[enum.Enum]

parse(*data: enum.Enum*) → enum.Enum

Parse data from string.

Parameters
data (Union[str, bytes, Enum]) – raw data

Return type
Optional[Enum]

Returns
The parsed enum data. If data is *unset*, None will be returned.

Warns
ZeekValueWarning – If date is not defined in the enum namespace.

tojson(*data: enum.Enum*) → str

tojson(*data: None*) → None

Serialize `data` as JSON log format.

Parameters
data (Optional[Enum]) – raw data

Returns
The JSON serialisable enum data.

Return type
str

toascii(*data*)

Serialize `data` as ASCII log format.

Parameters
data (Optional[Enum]) – raw data

Returns
The ASCII representation of the enum data.

Return type
str

4.1.7 Container Types

class zlogging.types.**SetType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, *element_type=None*, **args*, ***kwargs*)

Bases: *_GenericType*, Generic[_S]

Bro/Zeek `set` data type.

Parameters

- **empty_field** (Union[str, bytes, None]) – Placeholder for empty field.
- **unset_field** (Union[str, bytes, None]) – Placeholder for unset field.
- **set_separator** (Union[str, bytes, None]) – Separator for `set`/`vector` fields.
- **element_type** (Union[TypeVar(_S, bound= *_SimpleType*), Type[TypeVar(_S, bound= *_SimpleType*)], None]) – Data type of container's elements.
- ***args** – Arbitrary positional arguments.
- ****kwargs** – Arbitrary keyword arguments.

Raises

- **ZeekTypeError** – If `element_type` is not supplied.
- **ZeekValueError** – If `element_type` is not a valid Bro/Zeek data type.

Example

As a *generic* data type, the class supports the typing proxy as introduced **PEP 484**:

```
>>> SetType[StringType]
```

which is the same **at runtime** as following:

```
>>> SetType(element_type=StringType())
```

Note: A valid `element_type` should be a *simple* data type, i.e. a subclass of *_SimpleType*.

property python_type: Any

Corresponding Python type annotation.

Type
Any

Return type
Any

property zeek_type: str

Corresponding Zeek type name.

Type
str

Return type
str

element_type: _S

Data type of container's elements.

parse(*data: AnyStr*) → Optional[set[_S]]

parse(*data: set[_S]*) → set[_S]

Parse `data` from string.

Parameters
data – raw data

Returns
The parsed set data. If `data` is *unset*, `None` will be returned.

tojson(*data: set[_S]*) → list[Optional[_T]]

tojson(*data: None*) → None

Serialize `data` as JSON log format.

Parameters
data – raw data

Returns
The JSON serialisable set data.

toascii(*data*)

Serialize `data` as ASCII log format.

Parameters
data (*Optional[set[_S]]*) – raw data

Returns
The ASCII representation of the set data.

Return type
str

class zlogging.types.**VectorType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, *element_type=None*, **args*, ***kwargs*)

Bases: *_GenericType*, Generic[_S]

Bro/Zeek `vector` data type.

Parameters

- **empty_field** (Union[str, bytes, None]) – Placeholder for empty field.
- **unset_field** (Union[str, bytes, None]) – Placeholder for unset field.
- **set_separator** (Union[str, bytes, None]) – Separator for `set`/`vector` fields.
- **element_type** (Union[TypeVar(_S, bound= *_SimpleType*), Type[TypeVar(_S, bound= *_SimpleType*)], None]) – Data type of container's elements.
- ***args** – Arbitrary positional arguments.
- ****kwargs** – Arbitrary keyword arguments.

Raises

- **ZeekTypeError** – If `element_type` is not supplied.
- **ZeekValueError** – If `element_type` is not a valid Bro/Zeek data type.

Example

As a *generic* data type, the class supports the typing proxy as introduced **PEP 484**:

```
>>> VectorType[StringType]
```

which is the same **at runtime** as following:

```
>>> VectorType(element_type=StringType())
```

Note: A valid `element_type` should be a *simple* data type, i.e. a subclass of *_SimpleType*.

---

property **python_type**: Any

Corresponding Python type annotation.

Type
Any

Return type
Any

property **zeek_type**: str

Corresponding Zeek type name.

Type
str

Return type
str

element_type: _S

Data type of container's elements.

parse(*data: AnyStr*) → Optional[list[_S]]

parse(*data: list[_S]*) → list[_S]

Parse `data` from string.

Parameters
data – raw data

Returns
The parsed list data. If `data` is *unset*, `None` will be returned.

tojson(*data: list[_S]*) → list[Optional[_T]]

tojson(*data: None*) → None

Serialize `data` as JSON log format.

Parameters
data – raw data

Returns
The JSON serialisable list data.

Return type
list

toascii(*data*)

Serialize `data` as ASCII log format.

Parameters
data (*Optional[list[_S]]*) – raw data

Returns
The ASCII representation of the list data.

Return type
str

class zlogging.types.**RecordType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***element_mapping*)

Bases: *_VariadicType*

Bro/Zeek `record` data type.

Parameters

- **empty_field** (Union[str, bytes, None]) – Placeholder for empty field.
- **unset_field** (Union[str, bytes, None]) – Placeholder for unset field.
- **set_separator** (Union[str, bytes, None]) – Separator for `set`/`vector` fields.
- **element_mapping** (Union[Type[*_SimpleType*], *_SimpleType*, *_GenericType*]) – Data type of container's elements.
- ***args** (*Any*) – Arbitrary positional arguments.
- ****kwargs** (*Any*) – Arbitrary keyword arguments.

Raises

- **ZeekTypeError** – If `element_mapping` is not supplied.

- **ZeekValueError** – If `element_mapping` is not a valid Bro/Zeek data type; or in case of inconsistency from `empty_field`, `unset_field` and `set_separator` of each field.

Return type
RecordType

Note: A valid `element_mapping` should be a *simple* or *generic* data type, i.e. a subclass of *_SimpleType* or *_GenericType*.

See also:

See `_aux_expand_typing()` for more information about processing the fields.

property python_type: `Any`

Corresponding Python type annotation.

Return type
`Any`

property zeek_type: `Literal['record']`

Corresponding Zeek type name.

Return type
`Literal`['record']

element_mapping: `OrderedDict[str, Union[_SimpleType, _GenericType]]`

Data type of container's elements.

4.1.8 Any type

class zlogging.types.**AnyType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, *json_encoder=None*, **args*, ***kwargs*)

Bases: *_SimpleType*

Bro/Zeek any data type.

Parameters

- **empty_field** (`Union[str, bytes, None]`) – Placeholder for empty field.
- **unset_field** (`Union[str, bytes, None]`) – Placeholder for unset field.
- **set_separator** (`Union[str, bytes, None]`) – Separator for `set`/`vector` fields.
- **json_encoder** (`Optional[Type[JSONEncoder]]`) – JSON encoder class for *tojson()* method calls.
- ***args** – Arbitrary positional arguments.
- ****kwargs** – Arbitrary keyword arguments.

Note: The *AnyType* is only used for arbitrary typing as required in *JSONParser*. It is **NOT** a valid type of Bro/Zeek logging framework.

---

property python_type: `Any`

Corresponding Python type annotation.

Return type
Any

property zeek_type: Literal['any']

Corresponding Zeek type name.

Return type
Literal['any']

json_encoder: Type[JSONEncoder]

JSON encoder class for *tojson()* method calls.

parse(*data*)

Parse data from string.

Parameters
data (TypeVar(_T)) – raw data

Return type
Optional[TypeVar(_T)]

Returns
The parsed data. If data is *unset*, None will be returned.

tojson(*data*)

Serialize data as JSON log format.

Parameters
data (Any) – raw data

Return type
Any

Returns
The JSON representation of data.

Notes

If the data is not JSON serialisable, i.e. json.dumps() raises TypeError, the method will return a dict object with data representing str sanitised raw data and error representing the error message.

---

toascii(*data*)

Serialize data as ASCII log format.

Parameters
data (Any) – raw data

Return type
str

Returns
The ASCII representation of data.

4.2 Abstract Base Types

class zlogging.types.**BaseType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***kwargs*)

Bases: object

Base Bro/Zeek data type.

Parameters

- **empty_field** (Union[str, bytes, None]) – Placeholder for empty field.
- **unset_field** (Union[str, bytes, None]) – Placeholder for unset field.
- **set_separator** (Union[str, bytes, None]) – Separator for set/vector fields.
- ***args** – Arbitrary positional arguments.
- ****kwargs** – Arbitrary keyword arguments.

abstract property python_type: Any

Corresponding Python type annotation.

Return type
Any

abstract property zeek_type: str

Corresponding Zeek type name.

Return type
str

property bro_type: str

Corresponding Bro type name.

Return type
str

empty_field: bytes

Placeholder for empty field.

str_empty_field: str

unset_field: bytes

Placeholder for unset field.

str_unset_field: str

set_separator: bytes

Separator for set/vector fields.

str_set_separator: str

__call__(*data*)

Parse data from string.

This is a proxy method which calls to *parse()* of the type implementation.

Return type
Any

Parameters
data (*Any*) –

__str__()

Returns the corresponding Zeek type name.

Return type
str

abstract parse(*data*)

Parse `data` from string.

Return type
Any

Parameters
data (*Any*) –

abstract tojson(*data*)

Serialize `data` as JSON log format.

Return type
Any

Parameters
data (*Any*) –

abstract toascii(*data*)

Serialize `data` as ASCII log format.

Return type
str

Parameters
data (*Any*) –

class zlogging.types.**_SimpleType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***kwargs*)

Bases: *BaseType*

Simple data type.

In Bro/Zeek script language, such simple type includes `bool`, `count`, `int`, `double`, `time`, `interval`, `string`, `addr`, `port`, `subnet` and `enum`.

To support arbitrary typing as required in *JSONParser*, `any`, the arbitrary date type is also included.

Parameters

- **empty_field** (*bytes*) –
- **unset_field** (*bytes*) –
- **set_separator** (*bytes*) –
- **args** (*Any*) –
- **kwargs** (*Any*) –

empty_field: bytes

Placeholder for empty field.

str_empty_field: str

unset_field: bytes

Placeholder for unset field.

str_unset_field: *str*

set_separator: *bytes*

Separator for `set`/`vector` fields.

str_set_separator: *str*

class zlogging.types.**_GenericType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***kwargs*)

Bases: *BaseType*, Generic[_S]

Generic data type.

In Bro/Zeek script language, such generic type includes `set` and `vector`, which are also known as *container* types.

element_type: *_S*

Data type of container's elements.

class zlogging.types.**_VariadicType**(*empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***kwargs*)

Bases: *BaseType*

Variadic data type.

In Bro/Zeek script language, such variadic type refers to `record`, which is also a *container* type.

Parameters

- **empty_field** (*Optional[AnyStr]*) –
- **unset_field** (*Optional[AnyStr]*) –
- **set_separator** (*Optional[AnyStr]*) –
- **args** (*Any*) –
- **kwargs** (*Any*) –

element_mapping: *OrderedDict[str, Union[_SimpleType, _GenericType]]*

Data type of container's elements.

parse(*data*)

Not supported for a variadic data type.

Parameters
data (Any) – data to process

Raises
ZeekNotImplemented – If try to call such method.

Return type
NoReturn

tojson(*data*)

Not supported for a variadic data type.

Parameters
data (Any) – data to process

Raises
ZeekNotImplemented – If try to call such method.

Return type
NoReturn

`toascii`(*data*)

Not supported for a variadic data type.

Parameters
data (Any) – data to process

Raises
ZeekNotImplemented – If try to call such method.

Return type
NoReturn

CHAPTER

FIVE

TYPING ANNOTATIONS

5.1 Zeek Data Types

5.1.1 Boolean

zlogging.typing.**zeek_bool**: *BoolType*

Zeek `bool` data type.

5.1.2 Numeric Types

zlogging.typing.**zeek_count**: *CountType*

Zeek `count` data type.

zlogging.typing.**zeek_double**: *DoubleType*

Zeek `count` data type.

zlogging.typing.**zeek_int**: *IntType*

Zeek `int` data type.

5.1.3 Time Types

zlogging.typing.**zeek_time**: *TimeType*

Zeek `time` data type.

zlogging.typing.**zeek_interval**: *IntervalType*

Zeek `interval` data type.

5.1.4 String

zlogging.typing.**zeek_string**: *StringType*

Zeek `string` data type.

5.1.5 Network Types

zlogging.typing.**zeek_port**: *PortType*

Zeek `port` data type.

zlogging.typing.**zeek_addr**: *AddrType*

Zeek `addr` data type.

zlogging.typing.**zeek_subnet**: *SubnetType*

Zeek `subnet` data type.

5.1.6 Enumeration

zlogging.typing.**zeek_enum**: *EnumType*

Zeek `enum` data type.

5.1.7 Container Types

class zlogging.typing.**zeek_set**(*empty_field=None*, *unset_field=None*, *set_separator=None*, *element_type=None*, **args*, ***kwargs*)

Bases: *SetType*, `Generic`[_S]

Zeek `set` data type.

Notes

As a *generic* data type, the class supports the typing proxy as introduced **PEP 484**:

```
class MyLog(zeek_record):
    field_one: zeek_set[zeek_str]
```

which is the same **at runtime** as following:

```
class MyLog(zeek_record):
    field_one = SetType(element_type=StringType())
```

Parameters

- **empty_field** – Placeholder for empty field.
- **unset_field** – Placeholder for unset field.
- **set_separator** – Separator for `set`/`vector` fields.
- **element_type** – Data type of container's elements.
- ***args** – Arbitrary positional arguments.
- ****kwargs** – Arbitrary keyword arguments.

class zlogging.typing.**zeek_vector**(*empty_field=None*, *unset_field=None*, *set_separator=None*, *element_type=None*, **args*, ***kwargs*)

Bases: *VectorType*, Generic[_S]

Zeek `vector` data type.

Notes

As a *generic* data type, the class supports the typing proxy as introduced **PEP 484**:

```
class MyLog(zeek_record):
    field_one: zeek_vector[zeek_str]
```

which is the same **at runtime** as following:

```
class MyLog(zeek_record):
    field_one = VectorType(element_type=StringType())
```

Parameters

- **empty_field** – Placeholder for empty field.
- **unset_field** – Placeholder for unset field.
- **set_separator** – Separator for `set`/`vector` fields.
- **element_type** – Data type of container's elements.
- ***args** – Arbitrary positional arguments.
- ****kwargs** – Arbitrary keyword arguments.

class zlogging.typing.**zeek_record**(*empty_field=None*, *unset_field=None*, *set_separator=None*, **args*, ***element_mapping*)

Bases: *RecordType*

Zeek `record` data type.

Notes

As a *variadic* data type, it supports the typing proxy as `TypedDict`, introduced in **PEP 589**:

```
class MyLog(zeek_record):
    field_one: zeek_int
    field_two: zeek_set[zeek_port]
```

which is the same **at runtime** as following:

```
RecordType(field_one=IntType,
           field_two=SetType(element_type=PortType))
```

See also:

See *expand_typing()* for more information about the processing of typing proxy.

Parameters

- ***args** (*Any*) – Arbitrary positional arguments.
- ****kwargs** (*Any*) – Arbitrary keyword arguments.

Return type
RecordType

5.2 Bro Data Types

Warning: Use of `bro` is deprecated. Please use `zeek` instead.

5.2.1 Boolean

`zlogging.typing.`**bro_bool:** *BoolType*
Bro `bool` data type.

5.2.2 Numeric Types

`zlogging.typing.`**bro_count:** *CountType*
Bro `count` data type.

`zlogging.typing.`**bro_double:** *CountType*
Bro `count` data type.

`zlogging.typing.`**bro_int:** *IntType*
Bro `int` data type.

5.2.3 Time Types

`zlogging.typing.`**bro_time:** *TimeType*
Bro `time` data type.

`zlogging.typing.`**bro_interval:** *IntervalType*
Bro `interval` data type.

5.2.4 String

`zlogging.typing.`**bro_string:** *StringType*
Bro `string` data type.

5.2.5 Network Types

zlogging.typing.**bro_port**: *PortType*

Bro `port` data type.

zlogging.typing.**bro_addr**: *AddrType*

Bro `addr` data type.

zlogging.typing.**bro_subnet**: *SubnetType*

Bro `subnet` data type.

5.2.6 Enumeration

zlogging.typing.**bro_enum**: *EnumType*

Bro `enum` data type.

5.2.7 Container Types

class zlogging.typing.**bro_set**(**args*, ***kwargs*)

Bases: *SetType*, `Generic`[_S]

Bro `set` data type.

See also:

See *zeek_set* for more information.

Parameters

- ***args** – Arbitrary positional arguments.
- ****kwargs** – Arbitrary keyword arguments.

class zlogging.typing.**bro_vector**(**args*, ***kwargs*)

Bases: *VectorType*, `Generic`[_S]

Bro `vector` data type.

See also:

See *zeek_vector* for more information.

Parameters

- ***args** – Arbitrary positional arguments.
- ****kwargs** – Arbitrary keyword arguments.

class zlogging.typing.**bro_record**(**args*, ***kwargs*)

Bases: *RecordType*

Bro `record` data type.

See also:

See *zeek_record* for more information.

Parameters

- ***args** (*Any*) – Arbitrary positional arguments.
- ****kwargs** (*Any*) – Arbitrary keyword arguments.

Return type
RecordType

CHAPTER

SIX

EXCEPTIONS & WARNINGS

class zlogging._exc.**ZeekException**

Bases: Exception

Base exception.

class zlogging._exc.**ZeekWarning**

Bases: Warning

Base warning.

class zlogging._exc.**ParserError**(*msg*, *lineno=None*, *field=None*)

Bases: *ZeekException*, ValueError

Error when parsing logs.

Parameters

- **msg** (str) – The unformatted error message.
- **lineno** (Optional[int]) – The line corresponding to the failure.
- **field** (Optional[str]) – The field name where parsing failed.

msg: str

The unformatted error message.

field: Optional[str]

The field name where parsing failed.

lineno: Optional[int]

The line corresponding to the failure.

class zlogging._exc.**JSONParserError**(*msg*, *lineno=None*, *field=None*)

Bases: *ParserError*, JSONDecodeError

Error when parsing JSON log.

Parameters

- **msg** (str) – The unformatted error message.
- **lineno** (Optional[int]) – The line corresponding to the failure.
- **field** (Optional[str]) – The field name where parsing failed.

msg: str

The unformatted error message.

field: Optional[str]

The field name where parsing failed.

lineno: Optional[int]

The line corresponding to the failure.

class zlogging._exc.**ASCIIParserError**(*msg*, *lineno=None*, *field=None*)

Bases: *ParserError*

Error when parsing ASCII log.

Parameters

- **msg** (str) – The unformatted error message.
- **lineno** (Optional[int]) – The line corresponding to the failure.
- **field** (Optional[str]) – The field name where parsing failed.

msg: str

The unformatted error message.

field: Optional[str]

The field name where parsing failed.

lineno: Optional[int]

The line corresponding to the failure.

class zlogging._exc.**WriterError**(*msg*, *lineno=None*, *field=None*)

Bases: *ZeekException*, TypeError

Error when writing logs.

Parameters

- **msg** (str) – The unformatted error message.
- **lineno** (Optional[int]) – The line corresponding to the failure.
- **field** (Optional[str]) – The field name where parsing failed.

msg: str

The unformatted error message.

field: Optional[str]

The field name where parsing failed.

lineno: Optional[int]

The line corresponding to the failure.

class zlogging._exc.**JSONWriterError**(*msg*, *lineno=None*, *field=None*)

Bases: *WriterError*

Error when writing JSON logs.

Parameters

- **msg** (str) – The unformatted error message.
- **lineno** (Optional[int]) – The line corresponding to the failure.
- **field** (Optional[str]) – The field name where parsing failed.

msg: str

The unformatted error message.

field: Optional[str]

The field name where parsing failed.

lineno: Optional[int]

The line corresponding to the failure.

class zlogging._exc.**ASCIIWriterError**(*msg*, *lineno=None*, *field=None*)

Bases: *WriterError*

Error when writing ASCII logs.

Parameters

- **msg** (str) – The unformatted error message.
- **lineno** (Optional[int]) – The line corresponding to the failure.
- **field** (Optional[str]) – The field name where parsing failed.

msg: str

The unformatted error message.

field: Optional[str]

The field name where parsing failed.

lineno: Optional[int]

The line corresponding to the failure.

class zlogging._exc.**WriterFormatError**(*msg*, *lineno=None*, *field=None*)

Bases: *WriterError*, ValueError

Unsupported format.

Parameters

- **msg** (str) – The unformatted error message.
- **lineno** (Optional[int]) – The line corresponding to the failure.
- **field** (Optional[str]) – The field name where parsing failed.

msg: str

The unformatted error message.

field: Optional[str]

The field name where parsing failed.

lineno: Optional[int]

The line corresponding to the failure.

class zlogging._exc.**ParserWarning**

Bases: *ZeekWarning*, UserWarning

Warning when parsing logs.

class zlogging._exc.**JSONParserWarning**

Bases: *ParserWarning*

Warning when parsing logs in JSON format.

class zlogging._exc.**ASCIIParserWarning**

Bases: *ParserWarning*

Warning when parsing logs in ASCII format.

CHAPTER

SEVEN

INTERNAL AUXILIARY FUNCTIONS

7.1 I/O Utilities

zlogging._aux.**readline**(*file: BinaryFile*, *seperator: bytes = ...*, *maxsplit: int = - 1*, *decode: Literal[False] = False*) → list[bytes]

zlogging._aux.**readline**(*file: BinaryFile*, *seperator: bytes = ...*, *maxsplit: int = - 1*, *decode: Literal[True] = False*) → list[str]

Wrapper for `file.readline()` function.

Parameters

- **file** – Log file object opened in binary mode.
- **separator** – Data separator.
- **maxsplit** – Maximum number of splits to do; see `bytes.split()` and `str.split()` for more information.
- **decode** – If decide the buffered string with `ascii` encoding.

Returns

The splitted line as a `list` of `bytes`, or as `str` if `decode` if set to `True`.

7.2 Value Conversion

zlogging._aux.**decimal_toascii**(*data*, *infinite=None*)

Convert `decimal.Decimal` to ASCII.

Parameters

- **data** (`Decimal`) – A `decimal.Decimal` object.
- **infinite** (`Optional`[`str`]) – The ASCII representation of infinite numbers (`NaN` and infinity).

Return type

`str`

Returns

The converted ASCII string.

Example

When converting a `decimal.Decimal` object, for example:

```
>>> d = decimal.Decimal('-123.123456789')
```

the function will preserve only **6 digits** of its fractional part, i.e.:

```
>>> decimal_toascii(d)
'-123.123456'
```

Note: Infinite numbers, i.e. `NaN` and infinity (`inf`), will be converted as the value specified in `infinite`, in default the string representation of the number itself, i.e.:

- `NaN` -> `'NaN'`
- Infinity -> `'Infinity'`

---

zlogging._aux.**float_toascii**(*data*, *infinite=None*)

Convert `float` to ASCII.

Parameters

- **data** (`float`) – A `float` number.
- **infinite** (`Optional[str]`) – The ASCII representation of infinite numbers (`NaN` and infinity).

Return type
`str`

Returns
The converted ASCII string.

Example

When converting a `float` number, for example:

```
>>> f = -123.123456789
```

the function will preserve only **6 digits** of its fractional part, i.e.:

```
>>> float_toascii(f)
'-123.123456'
```

Note: Infinite numbers, i.e. `NaN` and infinity (`inf`), will be converted as the value specified in `infinite`, in default the string representation of the number itself, i.e.:

- `NaN` -> `'nan'`
- Infinity -> `'inf'`

---

zlogging._aux.**unicode_escape**(*string*)

Conterprocess of `bytes.decode('unicode_escape')`.

Parameters
string (`bytes`) – The bytestring to be escaped.

Return type
str

Returns
The escaped bytestring as an encoded string

Example

```
>>> b'\x09'.decode('unicode_escape')
'\\t'
>>> unicode_escape(b'\t')
'\\x09'
```

7.3 Typing Inspection

zlogging._aux.**expand_typing**(*cls*, *exc=None*)

Expand typing annotations.

Parameters

- **cls** (*Union[Model, Type[Model], _VariadicType, Type[_VariadicType]]*) – a variadic class which supports **PEP 484** style attribute typing annotations
- **exc** (*Optional[Type[ValueError]]*) – exception to be used in case of inconsistent values for `unset_field`, `empty_field` and `set_separator`

Returns

- `fields`: a mapping proxy of field names and their corresponding data types, i.e. an instance of a *BaseType* subclass
- `record_fields`: a mapping proxy for fields of `record` data type, i.e. an instance of *RecordType*
- `unset_fields`: placeholder for unset field
- `empty_fields`: placeholder for empty field
- `set_separator`: separator for `set`/`vector` fields

Return type
The returned dictionary contains the following directives

Warns
BroDeprecationWarning – Use of `bro_*` prefixed typing annotations.

Raises
ValueError – In case of inconsistent values for `unset_field`, `empty_field` and `set_separator`.

Example

Define a custom log data model from *Model* using the prefines Bro/Zeek data types, or subclasses of *BaseType*:

```
class MyLog(Model):
    field_one = StringType()
    field_two = SetType(element_type=PortType)
```

Or you may use type annotations as **PEP 484** introduced when declaring data models. All available type hints can be found in `zlogging.typing`:

```
class MyLog(Model):
    field_one: zeek_string
    field_two: zeek_set[zeek_port]
```

However, when mixing annotations and direct assignments, annotations will take proceedings, i.e. the function shall process first typing annotations then `cls` attribute assignments. Should there be any conflicts, the `exc` will be raised.

Note: Fields of *`zlogging.types.RecordType`* type will be expanded as plain fields of the `cls`, i.e. for the variadic class as below:

```
class MyLog(Model):
    record = RecrodType(one=StringType(),
                        two=VectorType(element_type=CountType()))
```

will have the following fields:

- `record.one` -> `string` data type
- `record.two` -> `vector[count]` data type

CHAPTER

EIGHT

DATA CLASSES

8.1 Predefined Data Classes

class zlogging._data.**ASCIIInfo**(*path*, *open*, *close*, *data*, *exit_with_error*)

Bases: *Info*

Parsed log info for ASCII logs.

The ASCII log will be stored as in this `dataclass`, as introduced in **PEP 557**.

Parameters

- **path** (*PathLike[str]*) – The value is specified in the ASCII log file under `# path` directive.
- **open** (*DateTimeType*) – The value is specified in the ASCII log file under `# open` directive.
- **close** (*DateTimeType*) – The value is specified in the ASCII log file under `# close` directive.
- **data** (*list[Model]*) – The log records parsed as a `list` of *Model* per line.
- **exit_with_error** (*bool*) – When exit with error, the ASCII log file doesn't has a `# close` directive.

property format: Literal['ascii']

Log file format.

Return type
Literal['ascii']

path: PathLike[str]

Log path. The value is specified in the ASCII log file under `# path` directive.

open: DateTimeType

Log open time. The value is specified in the ASCII log file under `# open` directive.

close: DateTimeType

Log close time. The value is specified in the ASCII log file under `# close` directive.

data: list[*Model*]

Log records. The log records parsed as a `list` of *Model* per line.

exit_with_error: bool

Log exit with error. When exit with error, the ASCII log file doesn't has a `# close` directive.

class zlogging._data.**JSONInfo**(*data*)

Bases: *Info*

Parsed log info for JSON logs.

The JSON log will be stored as in this dataclass, as introduced in **PEP 557**.

Parameters
data (*list[Model]*) – The log records parsed as a list of *Model* per line.

property format: Literal['json']

Log file format.

Return type
Literal[‘json’]

data: list[*Model*]

Log records. The log records parsed as a list of *Model* per line.

8.2 Abstract Base Data Class

class zlogging._data.**Info**

Bases: object

Parsed log info.

The parsed log will be stored as in this dataclass, as introduced in **PEP 557**.

abstract property format: str

Log file format.

Return type
str

CHAPTER

NINE

ENUM NAMESPACE

9.1 zeek Namespace

Namespace: zeek.

class zlogging.enum.zeek.**TableChange**(*value*)

Bases: IntFlag

Enum: TableChange.

See also:

base/bif/types.bif.zeek

TABLE_ELEMENT_NEW = 1

TABLE_ELEMENT_CHANGED = 2

TABLE_ELEMENT_REMOVED = 4

TABLE_ELEMENT_EXPIRED = 8

class zlogging.enum.zeek.**layer3_proto**(*value*)

Bases: IntFlag

Enum: layer3_proto.

See also:

base/bif/types.bif.zeek

L3_IPV4 = 1

L3_IPV6 = 2

L3_ARP = 4

L3_UNKNOWN = 8

class zlogging.enum.zeek.**link_encap**(*value*)

Bases: IntFlag

Enum: link_encap.

See also:

base/bif/types.bif.zeek

LINK_ETHERNET = 1

LINK_UNKNOWN = 2

class zlogging.enum.zeek.**rpc_status**(*value*)

Bases: IntFlag

Enum: rpc_status.

See also:

base/bif/types.bif.zeek

RPC_SUCCESS = 1

RPC_PROG_UNAVAIL = 2

RPC_PROG_MISMATCH = 4

RPC_PROC_UNAVAIL = 8

RPC_GARBAGE_ARGS = 16

RPC_SYSTEM_ERR = 32

RPC_TIMEOUT = 64

RPC_VERS_MISMATCH = 128

RPC_AUTH_ERROR = 256

RPC_UNKNOWN_ERROR = 512

class zlogging.enum.zeek.**IPAddrAnonymization**(*value*)

Bases: IntFlag

Enum: IPAddrAnonymization.

See also: anonymize_addr.

See also:

base/init-bare.zeek

KEEP_ORIG_ADDR = 1

SEQUENTIALLY_NUMBERED = 2

RANDOM_MD5 = 4

PREFIX_PRESERVING_A50 = 8

PREFIX_PRESERVING_MD5 = 16

class zlogging.enum.zeek.**IPAddrAnonymizationClass**(*value*)

Bases: IntFlag

Enum: IPAddrAnonymizationClass.

See also: anonymize_addr.

See also:

base/init-bare.zeek

ORIG_ADDR = 1

RESP_ADDR = 2

OTHER_ADDR = 4

class zlogging.enum.zeek.**PcapFilterID**(*value*)

Bases: IntFlag

Enum: PcapFilterID.

Enum type identifying dynamic BPF filters. These are used by Pcap::precompile_pcap_filter and Pcap::precompile_pcap_filter.

See also:

base/init-bare.zeek

PacketFilter_DefaultPcapFilter = 2

PacketFilter::DefaultPcapFilter (present if base/frameworks/packet-filter/main.zeek is loaded)

PacketFilter_FilterTester = 4

PacketFilter::FilterTester (present if base/frameworks/packet-filter/main.zeek is loaded)

None = 1

class zlogging.enum.zeek.**pkt_profile_modes**(*value*)

Bases: IntFlag

Enum: pkt_profile_modes.

Output modes for packet profiling information.

See also: pkt_profile_mode, pkt_profile_freq, pkt_profile_file.

See also:

base/init-bare.zeek

PKT_PROFILE_MODE_NONE = 1

No output.

PKT_PROFILE_MODE_SECS = 2

Output every pkt_profile_freq seconds.

PKT_PROFILE_MODE_PKTS = 4

Output every pkt_profile_freq packets.

PKT_PROFILE_MODE_BYTES = 8

Output every pkt_profile_freq bytes.

class zlogging.enum.zeek.**transport_proto**(*value*)

Bases: IntFlag

Enum: transport_proto.

A connection's transport-layer protocol. Note that Zeek uses the term "connection" broadly, using flow semantics for ICMP and UDP.

See also:

base/init-bare.zeek

unknown_transport = 1

An unknown transport-layer protocol.

tcp = 2

TCP.

udp = 4

UDP.

icmp = 8

ICMP.

class zlogging.enum.zeek.**Direction**(*value*)

Bases: IntFlag

Enum: Direction.

See also:

base/utils/directions-and-hosts.zeek

INBOUND = 1

The connection originator is not within the locally-monitored network, but the other endpoint is.

OUTBOUND = 2

The connection originator is within the locally-monitored network, but the other endpoint is not.

BIDIRECTIONAL = 4

Only one endpoint is within the locally-monitored network, meaning the connection is either outbound or inbound.

NO_DIRECTION = 8

This value doesn't match any connection.

class zlogging.enum.zeek.**Host**(*value*)

Bases: IntFlag

Enum: Host.

See also:

base/utils/directions-and-hosts.zeek

LOCAL_HOSTS = 1

A host within the locally-monitored network.

REMOTE_HOSTS = 2

A host not within the locally-monitored network.

ALL_HOSTS = 4

Any host.

NO_HOSTS = 8

This value doesn't match any host.

9.2 Broker Namespace

Namespace: Broker.

class zlogging.enum.Broker.**DataType**(*value*)

Bases: IntFlag

Enum: Broker::DataType.

Enumerates the possible types that Broker::Data may be in terms of Zeek data types.

See also:

base/bif/data.bif.zeek

NONE = 1

BOOL = 2

INT = 4

COUNT = 8

DOUBLE = 16

STRING = 32

ADDR = 64

SUBNET = 128

PORT = 256

TIME = 512

INTERVAL = 1024

ENUM = 2048

SET = 4096

TABLE = 8192

VECTOR = 16384

class zlogging.enum.Broker.**Type**(*value*)

Bases: IntFlag

Enum: Broker::Type.

The type of a Broker activity being logged.

See also:

base/frameworks/broker/log.zeek

STATUS = 1

An informational status update.

ERROR = 2

An error situation.

class zlogging.enum.Broker.**ErrorCode**(*value*)

Bases: IntFlag

Enum: Broker::ErrorCode.

Enumerates the possible error types.

See also:

base/frameworks/broker/main.zeek

NO_ERROR = 1

(present if base/bif/comm.bif.zeek is loaded)

UNSPECIFIED = 2

The unspecified default error code.

PEER_INCOMPATIBLE = 4

Version incompatibility.

PEER_INVALID = 8

Referenced peer does not exist.

PEER_UNAVAILABLE = 16

Remote peer not listening.

PEER_DISCONNECT_DURING_HANDSHAKE = 32

(present if base/bif/comm.bif.zeek is loaded)

PEER_TIMEOUT = 64

A peering request timed out.

MASTER_EXISTS = 128

Master with given name already exists.

NO_SUCH_MASTER = 256

Master with given name does not exist.

NO_SUCH_KEY = 512

The given data store key does not exist.

REQUEST_TIMEOUT = 1024

The store operation timed out.

TYPE_CLASH = 2048

The operation expected a different type than provided.

INVALID_DATA = 4096

The data value cannot be used to carry out the desired operation.

BACKEND_FAILURE = 8192

The storage backend failed to execute the operation.

STALE_DATA = 16384

The storage backend failed to execute the operation.

CANNOT_OPEN_FILE = 32768

(present if base/bif/comm.bif.zeek is loaded)

CANNOT_WRITE_FILE = 65536

(present if base/bif/comm.bif.zeek is loaded)

INVALID_TOPIC_KEY = 131072

(present if base/bif/comm.bif.zeek is loaded)

END_OF_FILE = 262144

(present if base/bif/comm.bif.zeek is loaded)

INVALID_TAG = 524288

(present if base/bif/comm.bif.zeek is loaded)

INVALID_STATUS = 1048576

(present if base/bif/comm.bif.zeek is loaded)

CAF_ERROR = 2097152

Catch-all for a CAF-level problem.

class zlogging.enum.Broker.**PeerStatus**(*value*)

Bases: IntFlag

Enum: Broker::PeerStatus.

The possible states of a peer endpoint.

See also:

base/frameworks/broker/main.zeek

INITIALIZING = 1

The peering process is initiated.

CONNECTING = 2

Connection establishment in process.

CONNECTED = 4

Connection established, peering pending.

PEERED = 8

Successfully peered.

DISCONNECTED = 16

Connection to remote peer lost.

RECONNECTING = 32

Reconnecting to peer after a lost connection.

class zlogging.enum.Broker.**BackendType**(*value*)

Bases: IntFlag

Enum: Broker::BackendType.

Enumerates the possible storage backends.

See also:

base/frameworks/broker/store.zeek

MEMORY = 1

SQLITE = 2

ROCKSDB = 4

class zlogging.enum.Broker.**QueryStatus**(*value*)

Bases: IntFlag

Enum: Broker::QueryStatus.

Whether a data store query could be completed or not.

See also:

base/frameworks/broker/store.zeek

SUCCESS = 1

FAILURE = 2

9.3 Cluster Namespace

Namespace: Cluster.

class zlogging.enum.Cluster.**NodeType**(*value*)

Bases: IntFlag

Enum: Cluster::NodeType.

Types of nodes that are allowed to participate in the cluster configuration.

See also:

base/frameworks/cluster/main.zeek

NONE = 1

A dummy node type indicating the local node is not operating within a cluster.

CONTROL = 2

A node type which is allowed to view/manipulate the configuration of other nodes in the cluster.

LOGGER = 4

A node type responsible for log management.

MANAGER = 8

A node type responsible for policy management.

PROXY = 16

A node type for relaying worker node communication and synchronizing worker node state.

WORKER = 32

The node type doing all the actual traffic analysis.

TIME_MACHINE = 64

A node acting as a traffic recorder using the Time Machine software.

9.4 DCE_RPC Namespace

Namespace: DCE_RPC.

class zlogging.enum.DCE_RPC.**IfID**(*value*)

Bases: IntFlag

Enum: DCE_RPC::IfID.

See also:

base/bif/plugins/Zeek_DCE_RPC.types.bif.zeek

unknown_if = 1

epmapper = 2

lsarpc = 4

lsa_ds = 8

mgmt = 16

netlogon = 32

samr = 64

srvsvc = 128

spoolss = 256

drs = 512

winspipe = 1024

wkssvc = 2048

oxid = 4096

ISCMActivator = 8192

class zlogging.enum.DCE_RPC.**PType**(*value*)

Bases: IntFlag

Enum: DCE_RPC::PType.

See also:

base/bif/plugins/Zeek_DCE_RPC.types.bif.zeek

REQUEST = 1

PING = 2

RESPONSE = 4

FAULT = 8

WORKING = 16

NOCALL = 32

REJECT = 64

ACK = 128

CL_CANCEL = 256

FACK = 512

CANCEL_ACK = 1024

BIND = 2048

BIND_ACK = 4096

BIND_NAK = 8192

ALTER_CONTEXT = 16384

ALTER_CONTEXT_RESP = 32768

AUTH3 = 65536

SHUTDOWN = 131072

CO_CANCEL = 262144

ORPHANED = 524288

RTS = 1048576

9.5 HTTP Namespace

Namespace: HTTP.

class zlogging.enum.HTTP.**Tags**(*value*)

Bases: IntFlag

Enum: HTTP::Tags.

Indicate a type of attack or compromise in the record to be logged.

See also:

base/protocols/http/main.zeek

EMPTY = 1

Placeholder.

URI_SQLI = 2

(present if policy/protocols/http/detect-sqli.zeek is loaded) Indicator of a URI based SQL injection attack.

POST_SQLI = 4

(present if policy/protocols/http/detect-sqli.zeek is loaded) Indicator of client body based SQL injection attack. This is typically the body content of a POST request. Not implemented yet.

COOKIE_SQLI = 8

(present if policy/protocols/http/detect-sqli.zeek is loaded) Indicator of a cookie based SQL injection attack. Not implemented yet.

9.6 Input Namespace

Namespace: Input.

class zlogging.enum.Input.**Event**(*value*)

Bases: IntFlag

Enum: Input::Event.

Type that describes what kind of change occurred.

See also:

base/frameworks/input/main.zeek

EVENT_NEW = 1

New data has been imported.

EVENT_CHANGED = 2

Existing data has been changed.

EVENT_REMOVED = 4

Previously existing data has been removed.

class zlogging.enum.Input.**Mode**(*value*)

Bases: IntFlag

Enum: Input::Mode.

Type that defines the input stream read mode.

See also:

base/frameworks/input/main.zeek

MANUAL = 1

Do not automatically reread the file after it has been read.

REREAD = 2

Reread the entire file each time a change is found.

STREAM = 4

Read data from end of file each time new data is appended.

class zlogging.enum.Input.**Reader**(*value*)

Bases: IntFlag

Enum: Input::Reader.

See also:

base/frameworks/input/main.zeek

READER_ASCII = 1

READER_BENCHMARK = 2

READER_BINARY = 4

READER_CONFIG = 8

READER_RAW = 16

READER_SQLITE = 32

9.7 Intel Namespace

Namespace: Intel.

class zlogging.enum.Intel.**Type**(*value*)

Bases: IntFlag

Enum: Intel::Type.

Enum type to represent various types of intelligence data.

See also:

base/frameworks/intel/main.zeek

ADDR = 1

An IP address.

SUBNET = 2

A subnet in CIDR notation.

URL = 4

//".

Type
A complete URL without the prefix "http

SOFTWARE = 8

Software name.

EMAIL = 16

Email address.

DOMAIN = 32

DNS domain name.

USER_NAME = 64

A user name.

CERT_HASH = 128

Certificate SHA-1 hash.

PUBKEY_HASH = 256

Public key MD5 hash. (SSH server host keys are a good example.)

FILE_HASH = 512

(present if base/frameworks/intel/files.zeek is loaded) File hash which is non-hash type specific. It's up to the user to query for any relevant hash types.

FILE_NAME = 1024

(present if base/frameworks/intel/files.zeek is loaded) File name. Typically with protocols with definite indications of a file name.

class zlogging.enum.Intel.**Where**(*value*)

Bases: IntFlag

Enum: Intel::Where.

Enum to represent where data came from when it was discovered. The convention is to prefix the name with IN_.

See also:

base/frameworks/intel/main.zeek

IN_ANYWHERE = 1

A catchall value to represent data of unknown provenance.

Conn_IN_ORIG = 2

Conn::IN_ORIG (present if policy/frameworks/intel/seen/where-locations.zeek is loaded)

Conn_IN_RESP = 4

Conn::IN_RESP (present if policy/frameworks/intel/seen/where-locations.zeek is loaded)

Files_IN_HASH = 8

Files::IN_HASH (present if policy/frameworks/intel/seen/where-locations.zeek is loaded)

Files_IN_NAME = 16

Files::IN_NAME (present if policy/frameworks/intel/seen/where-locations.zeek is loaded)

DNS_IN_REQUEST = 32

DNS::IN_REQUEST (present if policy/frameworks/intel/seen/where-locations.zeek is loaded)

DNS_IN_RESPONSE = 64

DNS::IN_RESPONSE (present if policy/frameworks/intel/seen/where-locations.zeek is loaded)

HTTP_IN_HOST_HEADER = 128

HTTP::IN_HOST_HEADER (present if policy/frameworks/intel/seen/where-locations.zeek is loaded)

HTTP_IN_REFERRER_HEADER = 256

HTTP::IN_REFERRER_HEADER (present if policy/frameworks/intel/seen/where-locations.zeek is loaded)

HTTP_IN_USER_AGENT_HEADER = 512

HTTP::IN_USER_AGENT_HEADER (present if policy/frameworks/intel/seen/where-locations.zeek is loaded)

HTTP_IN_X_FORWARDED_FOR_HEADER = 1024

HTTP::IN_X_FORWARDED_FOR_HEADER (present if policy/frameworks/intel/seen/where-locations.zeek is loaded)

HTTP_IN_URL = 2048

HTTP::IN_URL (present if policy/frameworks/intel/seen/where-locations.zeek is loaded)

SMTP_IN_MAIL_FROM = 4096

SMTP::IN_MAIL_FROM (present if policy/frameworks/intel/seen/where-locations.zeek is loaded)

SMTP_IN_RCPT_TO = 8192

SMTP::IN_RCPT_TO (present if policy/frameworks/intel/seen/where-locations.zeek is loaded)

SMTP_IN_FROM = 16384

SMTP::IN_FROM (present if policy/frameworks/intel/seen/where-locations.zeek is loaded)

SMTP_IN_TO = 32768

SMTP::IN_TO (present if policy/frameworks/intel/seen/where-locations.zeek is loaded)

SMTP_IN_CC = 65536

SMTP::IN_CC (present if policy/frameworks/intel/seen/where-locations.zeek is loaded)

SMTP_IN_RECEIVED_HEADER = 131072

SMTP::IN_RECEIVED_HEADER (present if policy/frameworks/intel/seen/where-locations.zeek is loaded)

SMTP_IN_REPLY_TO = 262144

SMTP::IN_REPLY_TO (present if policy/frameworks/intel/seen/where-locations.zeek is loaded)

SMTP_IN_X_ORIGINATING_IP_HEADER = 524288

SMTP::IN_X_ORIGINATING_IP_HEADER (present if policy/frameworks/intel/seen/where-locations.zeek is loaded)

SMTP_IN_MESSAGE = 1048576

SMTP::IN_MESSAGE (present if policy/frameworks/intel/seen/where-locations.zeek is loaded)

SSH_IN_SERVER_HOST_KEY = 2097152

SSH::IN_SERVER_HOST_KEY (present if policy/frameworks/intel/seen/where-locations.zeek is loaded)

SSL_IN_SERVER_NAME = 4194304

SSL::IN_SERVER_NAME (present if policy/frameworks/intel/seen/where-locations.zeek is loaded)

SMTP_IN_HEADER = 8388608

SMTP::IN_HEADER (present if policy/frameworks/intel/seen/where-locations.zeek is loaded)

X509_IN_CERT = 16777216

X509::IN_CERT (present if policy/frameworks/intel/seen/where-locations.zeek is loaded)

SMB_IN_FILE_NAME = 33554432

SMB::IN_FILE_NAME (present if policy/frameworks/intel/seen/where-locations.zeek is loaded)

SSH_SUCCESSFUL_LOGIN = 67108864

SSH::SUCCESSFUL_LOGIN (present if policy/protocols/ssh/detect-bruteforcing.zeek is loaded) An indicator of the login for the intel framework.

9.8 JSON Namespace

Namespace: JSON.

class zlogging.enum.JSON.**TimestampFormat**(*value*)

Bases: IntFlag

Enum: JSON::TimestampFormat.

See also:

base/init-bare.zeek

TS_EPOCH = 1

Timestamps will be formatted as UNIX epoch doubles. This is the format that Zeek typically writes out timestamps.

TS_MILLIS = 2

Timestamps will be formatted as unsigned integers that represent the number of milliseconds since the UNIX epoch.

TS_ISO8601 = 4

Timestamps will be formatted in the ISO8601 DateTime format. Subseconds are also included which isn't actually part of the standard but most consumers that parse ISO8601 seem to be able to cope with that.

9.9 Known Namespace

Namespace: Known.

class zlogging.enum.Known.**ModbusDeviceType**(*value*)

Bases: IntFlag

Enum: Known::ModbusDeviceType.

See also:

policy/protocols/modbus/known-masters-slaves.zeek

MODBUS_MASTER = 1

MODBUS_SLAVE = 2

9.10 LoadBalancing Namespace

Namespace: LoadBalancing.

class zlogging.enum.LoadBalancing.**Method**(*value*)

Bases: IntFlag

Enum: LoadBalancing::Method.

See also:

policy/misc/load-balancing.zeek

AUTO_BPF = 1

Apply BPF filters to each worker in a way that causes them to automatically flow balance traffic between them.

9.11 Log Namespace

Namespace: Log.

class zlogging.enum.Log.**ID**(*value*)

Bases: IntFlag

Enum: Log::ID.

Type that defines an ID unique to each log stream. Scripts creating new log streams need to redef this enum to add their own specific log ID. The log ID implicitly determines the default name of the generated log file.

See also:

base/frameworks/logging/main.zeek

UNKNOWN = 1

Dummy place-holder.

PRINTLOG = 2

Print statements that have been redirected to a log stream.

Broker_LOG = 4

Broker::LOG (present if base/frameworks/broker/log.zeek is loaded)

Files_LOG = 8

Files::LOG (present if base/frameworks/files/main.zeek is loaded) Logging stream for file analysis.

Reporter_LOG = 16

Reporter::LOG (present if base/frameworks/reporter/main.zeek is loaded)

Cluster_LOG = 32

Cluster::LOG (present if base/frameworks/cluster/main.zeek is loaded)

Notice_LOG = 64

Notice::LOG (present if base/frameworks/notice/main.zeek is loaded) This is the primary logging stream for notices.

Notice_ALARM_LOG = 128

Notice::ALARM_LOG (present if base/frameworks/notice/main.zeek is loaded) This is the alarm stream.

Weird_LOG = 256

Weird::LOG (present if base/frameworks/notice/weird.zeek is loaded)

DPD_LOG = 512

DPD::LOG (present if base/frameworks/dpd/main.zeek is loaded)

Signatures_LOG = 1024

Signatures::LOG (present if base/frameworks/signatures/main.zeek is loaded)

PacketFilter_LOG = 2048

PacketFilter::LOG (present if base/frameworks/packet-filter/main.zeek is loaded)

Software_LOG = 4096

Software::LOG (present if base/frameworks/software/main.zeek is loaded)

Intel_LOG = 8192

Intel::LOG (present if base/frameworks/intel/main.zeek is loaded)

Config_LOG = 16384

Config::LOG (present if base/frameworks/config/main.zeek is loaded)

Tunnel_LOG = 32768

Tunnel::LOG (present if base/frameworks/tunnels/main.zeek is loaded)

OpenFlow_LOG = 65536

OpenFlow::LOG (present if base/frameworks/openflow/plugins/log.zeek is loaded)

NetControl_LOG = 131072

NetControl::LOG (present if base/frameworks/netcontrol/main.zeek is loaded)

NetControl_DROP = 262144

NetControl::DROP (present if base/frameworks/netcontrol/types.zeek is loaded) Stop forwarding all packets matching the entity. No additional arguments.

NetControl_SHUNT = 524288

NetControl::SHUNT (present if base/frameworks/netcontrol/shunt.zeek is loaded)

Conn_LOG = 1048576

Conn::LOG (present if base/protocols/conn/main.zeek is loaded)

DCE_RPC_LOG = 2097152

DCE_RPC::LOG (present if base/protocols/dce-rpc/main.zeek is loaded)

DHCP_LOG = 4194304

DHCP::LOG (present if base/protocols/dhcp/main.zeek is loaded)

DNP3_LOG = 8388608

DNP3::LOG (present if base/protocols/dnp3/main.zeek is loaded)

DNS_LOG = 16777216

DNS::LOG (present if base/protocols/dns/main.zeek is loaded)

FTP_LOG = 33554432

FTP::LOG (present if base/protocols/ftp/main.zeek is loaded)

SSL_LOG = 67108864

SSL::LOG (present if base/protocols/ssl/main.zeek is loaded)

X509_LOG = 134217728

X509::LOG (present if base/files/x509/main.zeek is loaded)

HTTP_LOG = 268435456

HTTP::LOG (present if base/protocols/http/main.zeek is loaded)

IRC_LOG = 536870912

IRC::LOG (present if base/protocols/irc/main.zeek is loaded)

KRB_LOG = 1073741824

KRB::LOG (present if base/protocols/krb/main.zeek is loaded)

Modbus_LOG = 2147483648

Modbus::LOG (present if base/protocols/modbus/main.zeek is loaded)

mysql_LOG = 4294967296

mysql::LOG (present if base/protocols/mysql/main.zeek is loaded)

NTLM_LOG = 8589934592

NTLM::LOG (present if base/protocols/ntlm/main.zeek is loaded)

NTP_LOG = 17179869184

NTP::LOG (present if base/protocols/ntp/main.zeek is loaded)

RADIUS_LOG = 34359738368

RADIUS::LOG (present if base/protocols/radius/main.zeek is loaded)

RDP_LOG = 68719476736

RDP::LOG (present if base/protocols/rdp/main.zeek is loaded)

RFB_LOG = 137438953472

RFB::LOG (present if base/protocols/rfb/main.zeek is loaded)

SIP_LOG = 274877906944

SIP::LOG (present if base/protocols/sip/main.zeek is loaded)

SNMP_LOG = 549755813888

SNMP::LOG (present if base/protocols/snmp/main.zeek is loaded)

SMB_AUTH_LOG = 1099511627776

SMB::AUTH_LOG (present if base/protocols/smb/main.zeek is loaded)

SMB_MAPPING_LOG = 2199023255552

SMB::MAPPING_LOG (present if base/protocols/smb/main.zeek is loaded)

SMB_FILES_LOG = 4398046511104

SMB::FILES_LOG (present if base/protocols/smb/main.zeek is loaded)

SMTP_LOG = 8796093022208

SMTP::LOG (present if base/protocols/smtp/main.zeek is loaded)

SOCKS_LOG = 17592186044416

SOCKS::LOG (present if base/protocols/socks/main.zeek is loaded)

SSH_LOG = 35184372088832

SSH::LOG (present if base/protocols/ssh/main.zeek is loaded)

Syslog_LOG = 70368744177664

Syslog::LOG (present if base/protocols/syslog/main.zeek is loaded)

PE_LOG = 140737488355328

PE::LOG (present if base/files/pe/main.zeek is loaded)

NetControl_CATCH_RELEASE = 281474976710656

NetControl::CATCH_RELEASE (present if policy/frameworks/netcontrol/catch-and-release.zeek is loaded)

Unified2_LOG = 562949953421312

Unified2::LOG (present if policy/files/unified2/main.zeek is loaded)

OCSP_LOG = 1125899906842624

OCSP::LOG (present if policy/files/x509/log-ocsp.zeek is loaded)

Barnyard2_LOG = 2251799813685248

Barnyard2::LOG (present if policy/integration/barnyard2/main.zeek is loaded)

CaptureLoss_LOG = 4503599627370496

CaptureLoss::LOG (present if policy/misc/capture-loss.zeek is loaded)

Traceroute_LOG = 9007199254740992

Traceroute::LOG (present if policy/misc/detect-traceroute/main.zeek is loaded)

LoadedScripts_LOG = 18014398509481984

LoadedScripts::LOG (present if policy/misc/loaded-scripts.zeek is loaded)

Stats_LOG = 36028797018963968

Stats::LOG (present if policy/misc/stats.zeek is loaded)

WeirdStats_LOG = 72057594037927936

WeirdStats::LOG (present if policy/misc/weird-stats.zeek is loaded)

Known_HOSTS_LOG = 144115188075855872

Known::HOSTS_LOG (present if policy/protocols/conn/known-hosts.zeek is loaded)

Known_SERVICES_LOG = 288230376151711744

Known::SERVICES_LOG (present if policy/protocols/conn/known-services.zeek is loaded)

Known_MODBUS_LOG = 576460752303423488

Known::MODBUS_LOG (present if policy/protocols/modbus/known-masters-slaves.zeek is loaded)

Modbus_REGISTER_CHANGE_LOG = 1152921504606846976

Modbus::REGISTER_CHANGE_LOG (present if policy/protocols/modbus/track-memmap.zeek is loaded)

MQTT_CONNECT_LOG = 2305843009213693952

MQTT::CONNECT_LOG (present if policy/protocols/mqtt/main.zeek is loaded)

MQTT_SUBSCRIBE_LOG = 4611686018427387904

MQTT::SUBSCRIBE_LOG (present if policy/protocols/mqtt/main.zeek is loaded)

MQTT_PUBLISH_LOG = 9223372036854775808

MQTT::PUBLISH_LOG (present if policy/protocols/mqtt/main.zeek is loaded)

SMB_CMD_LOG = 18446744073709551616

SMB::CMD_LOG (present if policy/protocols/smb/log-cmds.zeek is loaded)

Known_CERTS_LOG = 36893488147419103232

Known::CERTS_LOG (present if policy/protocols/ssl/known-certs.zeek is loaded)

ZeekygenExample_LOG = 73786976294838206464

ZeekygenExample::LOG (present if zeekygen/example.zeek is loaded)

class zlogging.enum.Log.**PrintLogType**(*value*)

Bases: IntFlag

Enum: `Log::PrintLogType`.

Configurations for `Log::print_to_log`.

See also:

base/frameworks/logging/main.zeek

REDIRECT_NONE = 1

No redirection of print statements.

REDIRECT_STDOUT = 2

Redirection of those print statements that were being logged to stdout, leaving behind those set to go to other specific files.

REDIRECT_ALL = 4

Redirection of all print statements.

class zlogging.enum.Log.**Writer**(*value*)

Bases: IntFlag

Enum: `Log::Writer`.

See also:

base/frameworks/logging/main.zeek

WRITER_ASCII = 1

WRITER_NONE = 2

WRITER_SQLITE = 4

9.12 MOUNT3 Namespace

Namespace: MOUNT3.

class zlogging.enum.MOUNT3.**auth_flavor_t**(*value*)

Bases: IntFlag

Enum: MOUNT3::auth_flavor_t.

See also:

base/bif/types.bif.zeek

AUTH_NULL = 1

AUTH_UNIX = 2

AUTH_SHORT = 4

AUTH_DES = 8

class zlogging.enum.MOUNT3.**proc_t**(*value*)

Bases: IntFlag

Enum: MOUNT3::proc_t.

See also:

base/bif/types.bif.zeek

PROC_NULL = 1

PROC_MNT = 2

PROC_DUMP = 4

PROC_UMNT = 8

PROC_UMNT_ALL = 16

PROC_EXPORT = 32

PROC_END_OF_PROCS = 64

class zlogging.enum.MOUNT3.**status_t**(*value*)

Bases: IntFlag

Enum: MOUNT3::status_t.

See also:

base/bif/types.bif.zeek

MNT3_OK = 1

MNT3ERR_PERM = 2

MNT3ERR_NOENT = 4

MNT3ERR_IO = 8

MNT3ERR_ACCES = 16

MNT3ERR_NOTDIR = 32

MNT3ERR_INVAL = 64

MNT3ERR_NAMETOOLONG = 128

MNT3ERR_NOTSUPP = 256

MNT3ERR_SERVERFAULT = 512

MOUNT3ERR_UNKNOWN = 1024

9.13 MQTT Namespace

Namespace: MQTT.

class zlogging.enum.MQTT.**SubUnsub**(*value*)

Bases: IntFlag

Enum: MQTT::SubUnsub.

See also:

policy/protocols/mqtt/main.zeek

SUBSCRIBE = 1

UNSUBSCRIBE = 2

9.14 NFS3 Namespace

Namespace: NFS3.

class zlogging.enum.NFS3.**createmode_t**(*value*)

Bases: IntFlag

Enum: NFS3::createmode_t.

See also:

base/bif/types.bif.zeek

UNCHECKED = 1

GUARDED = 2

EXCLUSIVE = 4

class zlogging.enum.NFS3.**file_type_t**(*value*)

Bases: IntFlag

Enum: NFS3::file_type_t.

See also:

base/bif/types.bif.zeek

FTYPE_REG = 1

FTYPE_DIR = 2

FTYPE_BLK = 4

FTYPE_CHR = 8

FTYPE_LNK = 16

FTYPE_SOCK = 32

FTYPE_FIFO = 64

class zlogging.enum.NFS3.**proc_t**(*value*)

Bases: IntFlag

Enum: NFS3::proc_t.

See also:

base/bif/types.bif.zeek

PROC_NULL = 1

PROC_GETATTR = 2

PROC_SETATTR = 4

PROC_LOOKUP = 8

PROC_ACCESS = 16

PROC_READLINK = 32

PROC_READ = 64

PROC_WRITE = 128

PROC_CREATE = 256

PROC_MKDIR = 512

PROC_SYMLINK = 1024

PROC_MKNOD = 2048

PROC_REMOVE = 4096

PROC_RMDIR = 8192

PROC_RENAME = 16384

PROC_LINK = 32768

PROC_READDIR = 65536

PROC_READDIRPLUS = 131072

PROC_FSSTAT = 262144

PROC_FSINFO = 524288

PROC_PATHCONF = 1048576

PROC_COMMIT = 2097152

PROC_END_OF_PROCS = 4194304

class zlogging.enum.NFS3.**stable_how_t**(*value*)

Bases: IntFlag

Enum: NFS3::stable_how_t.

See also:

base/bif/types.bif.zeek

UNSTABLE = 1

DATA_SYNC = 2

FILE_SYNC = 4

class zlogging.enum.NFS3.**status_t**(*value*)

Bases: IntFlag

Enum: NFS3::status_t.

See also:

base/bif/types.bif.zeek

NFS3ERR_OK = 1

NFS3ERR_PERM = 2

NFS3ERR_NOENT = 4

NFS3ERR_IO = 8

NFS3ERR_NXIO = 16

NFS3ERR_ACCES = 32

NFS3ERR_EXIST = 64

NFS3ERR_XDEV = 128

NFS3ERR_NODEV = 256

NFS3ERR_NOTDIR = 512

NFS3ERR_ISDIR = 1024

NFS3ERR_INVAL = 2048

NFS3ERR_FBIG = 4096

NFS3ERR_NOSPC = 8192

NFS3ERR_ROFS = 16384

NFS3ERR_MLINK = 32768

NFS3ERR_NAMETOOLONG = 65536

NFS3ERR_NOTEMPTY = 131072

NFS3ERR_DQUOT = 262144

NFS3ERR_STALE = 524288

NFS3ERR_REMOTE = 1048576

NFS3ERR_BADHANDLE = 2097152

NFS3ERR_NOT_SYNC = 4194304

NFS3ERR_BAD_COOKIE = 8388608

NFS3ERR_NOTSUPP = 16777216

NFS3ERR_TOOSMALL = 33554432

NFS3ERR_SERVERFAULT = 67108864

NFS3ERR_BADTYPE = 134217728

NFS3ERR_JUKEBOX = 268435456

NFS3ERR_UNKNOWN = 536870912

class zlogging.enum.NFS3.**time_how_t**(*value*)

Bases: IntFlag

Enum: NFS3::time_how_t.

See also:

base/bif/types.bif.zeek

DONT_CHANGE = 1

SET_TO_SERVER_TIME = 2

SET_TO_CLIENT_TIME = 4

9.15 `Notice` Namespace

Namespace: `Notice`.

class zlogging.enum.Notice.**Action**(*value*)

Bases: `IntFlag`

Enum: `Notice::Action`.

These are values representing actions that can be taken with notices.

See also:

base/frameworks/notice/main.zeek

ACTION_NONE = 1

Indicates that there is no action to be taken.

ACTION_LOG = 2

Indicates that the notice should be sent to the notice logging stream.

ACTION_EMAIL = 4

Indicates that the notice should be sent to the email address(es) configured in the Notice::mail_dest variable.

ACTION_ALARM = 8

Indicates that the notice should be alarmed. A readable ASCII version of the alarm log is emailed in bulk to the address(es) configured in Notice::mail_dest.

ACTION_EMAIL_ADMIN = 16

(present if base/frameworks/notice/actions/email_admin.zeek is loaded) Indicate that the generated email should be addressed to the appropriate email addresses as found by the Site::get_emails function based on the relevant address or addresses indicated in the notice.

ACTION_PAGE = 32

(present if base/frameworks/notice/actions/page.zeek is loaded) Indicates that the notice should be sent to the pager email address configured in the Notice::mail_page_dest variable.

ACTION_ADD_GEODATA = 64

(present if base/frameworks/notice/actions/add-geodata.zeek is loaded) Indicates that the notice should have geodata added for the "remote" host. Site::local_nets must be defined in order for this to work.

ACTION_DROP = 128

(present if policy/frameworks/notice/actions/drop.zeek is loaded) Drops the address via NetControl::drop_address_catch_release.

class zlogging.enum.Notice.**Type**(*value*)

Bases: `IntFlag`

Enum: `Notice::Type`.

Scripts creating new notices need to redef this enum to add their own specific notice types which would then get used when they call the `NOTICE` function. The convention is to give a general category along with the specific notice separating words with underscores and using leading capitals on each word except for abbreviations which are kept in all capitals. For example, SSH::Password_Guessing is for hosts that have crossed a threshold of failed SSH logins.

See also:

base/frameworks/notice/main.zeek

Tally = 1

Notice reporting a count of how often a notice occurred.

Weird_Activity = 2

Weird::Activity (present if base/frameworks/notice/weird.zeek is loaded) Generic unusual but notice-worthy weird activity.

Signatures_Sensitive_Signature = 4

Signatures::Sensitive_Signature (present if base/frameworks/signatures/main.zeek is loaded) Generic notice type for notice-worthy signature matches.

Signatures_Multiple_Signatures = 8

Signatures::Multiple_Signatures (present if base/frameworks/signatures/main.zeek is loaded) Host has triggered many signatures on the same host. The number of signatures is defined by the Signatures::vert_scan_thresholds variable.

Signatures_Multiple_Sig_Responders = 16

Signatures::Multiple_Sig_Responders (present if base/frameworks/signatures/main.zeek is loaded) Host has triggered the same signature on multiple hosts as defined by the Signatures::horiz_scan_thresholds variable.

Signatures_Count_Signature = 32

Signatures::Count_Signature (present if base/frameworks/signatures/main.zeek is loaded) The same signature has triggered multiple times for a host. The number of times the signature has been triggered is defined by the Signatures::count_thresholds variable. To generate this notice, the Signatures::SIG_COUNT_PER_RESP action must be set for the signature.

Signatures_Signature_Summary = 64

Signatures::Signature_Summary (present if base/frameworks/signatures/main.zeek is loaded) Summarize the number of times a host triggered a signature. The interval between summaries is defined by the Signatures::summary_interval variable.

PacketFilter_Compile_Failure = 128

PacketFilter::Compile_Failure (present if base/frameworks/packet-filter/main.zeek is loaded) This notice is generated if a packet filter cannot be compiled.

PacketFilter_Install_Failure = 256

PacketFilter::Install_Failure (present if base/frameworks/packet-filter/main.zeek is loaded) Generated if a packet filter fails to install.

PacketFilter_Too_Long_To_Compile_Filter = 512

PacketFilter::Too_Long_To_Compile_Filter (present if base/frameworks/packet-filter/main.zeek is loaded) Generated when a notice takes too long to compile.

PacketFilter_Dropped_Packets = 1024

PacketFilter::Dropped_Packets (present if base/frameworks/packet-filter/netstats.zeek is loaded) Indicates packets were dropped by the packet filter.

ProtocolDetector_Protocol_Found = 2048

ProtocolDetector::Protocol_Found (present if policy/frameworks/dpd/detect-protocols.zeek is loaded)

ProtocolDetector_Server_Found = 4096

ProtocolDetector::Server_Found (present if policy/frameworks/dpd/detect-protocols.zeek is loaded)

Intel_Notice = 8192

Intel::Notice (present if policy/frameworks/intel/do_notice.zeek is loaded) This notice is generated when an intelligence indicator is denoted to be notice-worthy.

TeamCymruMalwareHashRegistry_Match = 16384

TeamCymruMalwareHashRegistry::Match (present if policy/frameworks/files/detect-MHR.zeek is loaded) The hash value of a file transferred over HTTP matched in the malware hash registry.

PacketFilter_No_More_Conn_Shunts_Available = 32768

PacketFilter::No_More_Conn_Shunts_Available (present if policy/frameworks/packet-filter/shunt.zeek is loaded) Indicative that PacketFilter::max_bpf_shunts connections are already being shunted with BPF filters and no more are allowed.

PacketFilter_Cannot_BPF_Shunt_Conn = 65536

PacketFilter::Cannot_BPF_Shunt_Conn (present if policy/frameworks/packet-filter/shunt.zeek is loaded) Limitations in BPF make shunting some connections with BPF impossible. This notice encompasses those various cases.

Software_Software_Version_Change = 131072

Software::Software_Version_Change (present if policy/frameworks/software/version-changes.zeek is loaded) For certain software, a version changing may matter. In that case, this notice will be generated. Software that matters if the version changes can be configured with the Software::interesting_version_changes variable.

Software_Vulnerable_Version = 262144

Software::Vulnerable_Version (present if policy/frameworks/software/vulnerable.zeek is loaded) Indicates that a vulnerable version of software was detected.

CaptureLoss_Too_Much_Loss = 524288

CaptureLoss::Too_Much_Loss (present if policy/misc/capture-loss.zeek is loaded) Report if the detected capture loss exceeds the percentage threshold.

Traceroute_Detected = 1048576

Traceroute::Detected (present if policy/misc/detect-traceroute/main.zeek is loaded) Indicates that a host was seen running traceroutes. For more detail about specific traceroutes that we run, refer to the traceroute.log.

Scan_Address_Scan = 2097152

Scan::Address_Scan (present if policy/misc/scan.zeek is loaded) Address scans detect that a host appears to be scanning some number of destinations on a single port. This notice is generated when more than Scan::addr_scan_threshold unique hosts are seen over the previous Scan::addr_scan_interval time range.

Scan_Port_Scan = 4194304

Scan::Port_Scan (present if policy/misc/scan.zeek is loaded) Port scans detect that an attacking host appears to be scanning a single victim host on several ports. This notice is generated when an attacking host attempts to connect to Scan::port_scan_threshold unique ports on a single host over the previous Scan::port_scan_interval time range.

Conn_Retransmission_Inconsistency = 8388608

Conn::Retransmission_Inconsistency (present if policy/protocols/conn/weirds.zeek is loaded) Possible evasion; usually just chud.

Conn_Content_Gap = 16777216

Conn::Content_Gap (present if policy/protocols/conn/weirds.zeek is loaded) Data has sequence hole; perhaps due to filtering.

DNS_External_Name = 33554432

DNS::External_Name (present if policy/protocols/dns/detect-external-names.zeek is loaded) Raised when a non-local name is found to be pointing at a local host. The Site::local_zones variable must be set appropriately for this detection.

FTP_Bruteforcing = 67108864

FTP::Bruteforcing (present if policy/protocols/ftp/detect-bruteforcing.zeek is loaded) Indicates a host bruteforcing FTP logins by watching for too many rejected usernames or failed passwords.

FTP_Site_Exec_Success = 134217728

FTP::Site_Exec_Success (present if policy/protocols/ftp/detect.zeek is loaded) Indicates that a successful response to a "SITE EXEC" command/arg pair was seen.

HTTP_SQL_Injection_Attacker = 268435456

HTTP::SQL_Injection_Attacker (present if policy/protocols/http/detect-sqli.zeek is loaded) Indicates that a host performing SQL injection attacks was detected.

HTTP_SQL_Injection_Victim = 536870912

HTTP::SQL_Injection_Victim (present if policy/protocols/http/detect-sqli.zeek is loaded) Indicates that a host was seen to have SQL injection attacks against it. This is tracked by IP address as opposed to hostname.

SMTP_Blocklist_Error_Message = 1073741824

SMTP::Blocklist_Error_Message (present if policy/protocols/smtp/blocklists.zeek is loaded) An SMTP server sent a reply mentioning an SMTP block list.

SMTP_Blocklist_Blocked_Host = 2147483648

SMTP::Blocklist_Blocked_Host (present if policy/protocols/smtp/blocklists.zeek is loaded) The originator's address is seen in the block list error message. This is useful to detect local hosts sending SPAM with a high positive rate.

SMTP_Suspicious_Origination = 4294967296

SMTP::Suspicious_Origination (present if policy/protocols/smtp/detect-suspicious-orig.zeek is loaded)

SSH_Password_Guessing = 8589934592

SSH::Password_Guessing (present if policy/protocols/ssh/detect-bruteforcing.zeek is loaded) Indicates that a host has been identified as crossing the SSH::password_guesses_limit threshold with failed logins.

SSH_Login_By_Password_Guesser = 17179869184

SSH::Login_By_Password_Guesser (present if policy/protocols/ssh/detect-bruteforcing.zeek is loaded) Indicates that a host previously identified as a "password guesser" has now had a successful login attempt. This is not currently implemented.

SSH_Watched_Country_Login = 34359738368

SSH::Watched_Country_Login (present if policy/protocols/ssh/geo-data.zeek is loaded) If an SSH login is seen to or from a "watched" country based on the SSH::watched_countries variable then this notice will be generated.

SSH_Interesting_Hostname_Login = 68719476736

SSH::Interesting_Hostname_Login (present if policy/protocols/ssh/interesting-hostnames.zeek is loaded) Generated if a login originates or responds with a host where the reverse hostname lookup resolves to a name matched by the SSH::interesting_hostnames regular expression.

SSL_Certificate_Expired = 137438953472

SSL::Certificate_Expired (present if policy/protocols/ssl/expiring-certs.zeek is loaded) Indicates that a certificate's NotValidAfter date has lapsed and the certificate is now invalid.

SSL_Certificate_Expires_Soon = 274877906944

SSL::Certificate_Expires_Soon (present if policy/protocols/ssl/expiring-certs.zeek is loaded) Indicates that a certificate is going to expire within SSL::notify_when_cert_expiring_in.

SSL_Certificate_Not_Valid_Yet = 549755813888

SSL::Certificate_Not_Valid_Yet (present if policy/protocols/ssl/expiring-certs.zeek is loaded) Indicates that a certificate's NotValidBefore date is future dated.

Heartbleed_SSL_Heartbeat_Attack = 1099511627776

Heartbleed::SSL_Heartbeat_Attack (present if policy/protocols/ssl/heartbleed.zeek is loaded) Indicates that a host performed a heartbleed attack or scan.

Heartbleed_SSL_Heartbeat_Attack_Success = 2199023255552

Heartbleed::SSL_Heartbeat_Attack_Success (present if policy/protocols/ssl/heartbleed.zeek is loaded) Indicates that a host performing a heartbleed attack was probably successful.

Heartbleed_SSL_Heartbeat_Odd_Length = 4398046511104

Heartbleed::SSL_Heartbeat_Odd_Length (present if policy/protocols/ssl/heartbleed.zeek is loaded) Indicates we saw heartbeat requests with odd length. Probably an attack or scan.

Heartbleed_SSL_Heartbeat_Many_Requests = 8796093022208

Heartbleed::SSL_Heartbeat_Many_Requests (present if policy/protocols/ssl/heartbleed.zeek is loaded) Indicates we saw many heartbeat requests without a reply. Might be an attack.

SSL_Invalid_Server_Cert = 17592186044416

SSL::Invalid_Server_Cert (present if policy/protocols/ssl/validate-certs.zeek is loaded) This notice indicates that the result of validating the certificate along with its full certificate chain was invalid.

SSL_Invalid_Ocsp_Response = 35184372088832

SSL::Invalid_Ocsp_Response (present if policy/protocols/ssl/validate-ocsp.zeek is loaded) This indicates that the OCSP response was not deemed to be valid.

SSL_Weak_Key = 70368744177664

SSL::Weak_Key (present if policy/protocols/ssl/weak-keys.zeek is loaded) Indicates that a server is using a potentially unsafe key.

SSL_Old_Version = 140737488355328

SSL::Old_Version (present if policy/protocols/ssl/weak-keys.zeek is loaded) Indicates that a server is using a potentially unsafe version

SSL_Weak_Cipher = 281474976710656

SSL::Weak_Cipher (present if policy/protocols/ssl/weak-keys.zeek is loaded) Indicates that a server is using a potentially unsafe cipher

ZeekygenExample_Zeekygen_One = 562949953421312

ZeekygenExample::Zeekygen_One (present if zeekygen/example.zeek is loaded) Any number of this type of comment will document "Zeekygen_One".

ZeekygenExample_Zeekygen_Two = 1125899906842624

ZeekygenExample::Zeekygen_Two (present if zeekygen/example.zeek is loaded) Any number of this type of comment will document "ZEEKYGEN_TWO".

ZeekygenExample_Zeekygen_Three = 2251799813685248

ZeekygenExample::Zeekygen_Three (present if zeekygen/example.zeek is loaded)

ZeekygenExample_Zeekygen_Four = 4503599627370496

ZeekygenExample::Zeekygen_Four (present if zeekygen/example.zeek is loaded) Omitting comments is fine, and so is mixing ## and ##<, but it's probably best to use only one style consistently.

9.16 OpenFlow Namespace

Namespace: OpenFlow.

class zlogging.enum.OpenFlow.**ofp_action_type**(*value*)

Bases: IntFlag

Enum: OpenFlow::ofp_action_type.

Openflow action_type definitions.

The openflow action type defines what actions openflow can take to modify a packet.

See also:

base/frameworks/openflow/consts.zeek

OFPAT_OUTPUT = 1

Output to switch port.

OFPAT_SET_VLAN_VID = 2

Set the 802.1q VLAN id.

OFPAT_SET_VLAN_PCP = 4

Set the 802.1q priority.

OFPAT_STRIP_VLAN = 8

Strip the 802.1q header.

OFPAT_SET_DL_SRC = 16

Ethernet source address.

OFPAT_SET_DL_DST = 32

Ethernet destination address.

OFPAT_SET_NW_SRC = 64

IP source address.

OFPAT_SET_NW_DST = 128

IP destination address.

OFPAT_SET_NW_TOS = 256

IP ToS (DSCP field, 6 bits).

OFPAT_SET_TP_SRC = 512

TCP/UDP source port.

OFPAT_SET_TP_DST = 1024

TCP/UDP destination port.

OFPAT_ENQUEUE = 2048

Output to queue.

OFPAT_VENDOR = 4096

Vendor specific.

class zlogging.enum.OpenFlow.**ofp_config_flags**(*value*)

Bases: IntFlag

Enum: OpenFlow::ofp_config_flags.

Openflow config flag definitions.

TODO: describe.

See also:

base/frameworks/openflow/consts.zeek

OFPC_FRAG_NORMAL = 1

No special handling for fragments.

OFPC_FRAG_DROP = 2

Drop fragments.

OFPC_FRAG_REASM = 4

Reassemble (only if OFPC_IP_REASM set).

OFPC_FRAG_MASK = 8

class zlogging.enum.OpenFlow.**ofp_flow_mod_command**(*value*)

Bases: IntFlag

Enum: OpenFlow::ofp_flow_mod_command.

Openflow flow_mod_command definitions.

The openflow flow_mod_command describes of what kind an action is.

See also:

base/frameworks/openflow/consts.zeek

OFPFC_ADD = 1

New flow.

OFPFC_MODIFY = 2

Modify all matching flows.

OFPFC_MODIFY_STRICT = 4

Modify entry strictly matching wildcards.

OFPFC_DELETE = 8

Delete all matching flows.

OFPFC_DELETE_STRICT = 16

Strictly matching wildcards and priority.

class zlogging.enum.OpenFlow.**Plugin**(*value*)

Bases: IntFlag

Enum: OpenFlow::Plugin.

Available openflow plugins.

See also:

base/frameworks/openflow/types.zeek

INVALID = 1

Internal placeholder plugin.

RYU = 2

(present if base/frameworks/openflow/plugins/ryu.zeek is loaded)

OFLOG = 4

(present if base/frameworks/openflow/plugins/log.zeek is loaded)

BROKER = 8

(present if base/frameworks/openflow/plugins/broker.zeek is loaded)

9.17 NetControl Namespace

Namespace: NetControl.

class zlogging.enum.NetControl.**InfoCategory**(*value*)

Bases: IntFlag

Enum: NetControl::InfoCategory.

Type of an entry in the NetControl log.

See also:

base/frameworks/netcontrol/main.zeek

MESSAGE = 1

A log entry reflecting a framework message.

ERROR = 2

A log entry reflecting a framework message.

RULE = 4

A log entry about a rule.

class zlogging.enum.NetControl.**InfoState**(*value*)

Bases: IntFlag

Enum: NetControl::InfoState.

State of an entry in the NetControl log.

See also:

base/frameworks/netcontrol/main.zeek

REQUESTED = 1

The request to add/remove a rule was sent to the respective backend.

SUCCEEDED = 2

A rule was successfully added by a backend.

EXISTS = 4

A backend reported that a rule was already existing.

FAILED = 8

A rule addition failed.

REMOVED = 16

A rule was successfully removed by a backend.

TIMEOUT = 32

A rule timeout was triggered by the NetControl framework or a backend.

class zlogging.enum.NetControl.**EntityType**(*value*)

Bases: IntFlag

Enum: NetControl::EntityType.

Type defining the entity that a rule applies to.

See also:

base/frameworks/netcontrol/types.zeek

ADDRESS = 1

Activity involving a specific IP address.

CONNECTION = 2

Activity involving all of a bi-directional connection's activity.

FLOW = 4

Activity involving a uni-directional flow's activity. Can contain wildcards.

MAC = 8

Activity involving a MAC address.

class zlogging.enum.NetControl.**RuleType**(*value*)

Bases: IntFlag

Enum: NetControl::RuleType.

Type of rules that the framework supports. Each type lists the extra NetControl::Rule fields it uses, if any.

Plugins may extend this type to define their own.

See also:

base/frameworks/netcontrol/types.zeek

DROP = 1

Stop forwarding all packets matching the entity. No additional arguments.

MODIFY = 2

Modify all packets matching entity. The packets will be modified according to the mod entry of the rule.

REDIRECT = 4

Redirect all packets matching entity to a different switch port, given in the out_port argument of the rule.

WHITELIST = 8

Whitelists all packets of an entity, meaning no restrictions will be applied. While whitelisting is the default if no rule matches, this type can be used to override lower-priority rules that would otherwise take effect for the entity.

class zlogging.enum.NetControl.**TargetType**(*value*)

Bases: IntFlag

Enum: NetControl::TargetType.

Type defining the target of a rule.

Rules can either be applied to the forward path, affecting all network traffic, or on the monitor path, only affecting the traffic that is sent to Zeek. The second is mostly used for shunting, which allows Zeek to tell the networking hardware that it wants to no longer see traffic that it identified as benign.

See also:

base/frameworks/netcontrol/types.zeek

FORWARD = 1

MONITOR = 2

class zlogging.enum.NetControl.**CatchReleaseActions**(*value*)

Bases: IntFlag

Enum: NetControl::CatchReleaseActions.

The enum that contains the different kinds of messages that are logged by catch and release.

See also:

policy/frameworks/netcontrol/catch-and-release.zeek

INFO = 1

Log lines marked with info are purely informational; no action was taken.

ADDED = 2

A rule for the specified IP address already existed in NetControl (outside of catch-and-release). Catch and release did not add a new rule, but is now watching the IP address and will add a new rule after the current rule expires.

DROP = 4

(present if base/frameworks/netcontrol/types.zeek is loaded) Stop forwarding all packets matching the entity. No additional arguments.

DROPPED = 8

A drop was requested by catch and release. An address was successfully blocked by catch and release.

UNBLOCK = 16

An address was unblocked after the timeout expired.

FORGOTTEN = 32

An address was forgotten because it did not reappear within the watch_until interval.

SEEN_AGAIN = 64

A watched IP address was seen again; catch and release will re-block it.

9.18 ProtocolDetector Namespace

Namespace: ProtocolDetector.

class zlogging.enum.ProtocolDetector.**dir**(*value*)

Bases: IntFlag

Enum: ProtocolDetector::dir.

See also:

policy/frameworks/dpd/detect-protocols.zeek

NONE = 1

INCOMING = 2

OUTGOING = 4

BOTH = 8

9.19 Reporter Namespace

Namespace: Reporter.

class zlogging.enum.Reporter.**Level**(*value*)

Bases: IntFlag

Enum: Reporter::Level.

See also:

base/bif/types.bif.zeek

INFO = 1

WARNING = 2

ERROR = 4

9.20 SMB Namespace

Namespace: SMB.

class zlogging.enum.SMB.**Action**(*value*)

Bases: IntFlag

Enum: SMB::Action.

Abstracted actions for SMB file actions.

See also:

base/protocols/smb/main.zeek

FILE_READ = 1

FILE_WRITE = 2

FILE_OPEN = 4

FILE_CLOSE = 8

FILE_DELETE = 16

FILE_RENAME = 32

FILE_SET_ATTRIBUTE = 64

PIPE_READ = 128

PIPE_WRITE = 256

PIPE_OPEN = 512

PIPE_CLOSE = 1024

PRINT_READ = 2048

PRINT_WRITE = 4096

PRINT_OPEN = 8192

PRINT_CLOSE = 16384

9.21 SOCKS Namespace

Namespace: SOCKS.

class zlogging.enum.SOCKS.**RequestType**(*value*)

Bases: IntFlag

Enum: SOCKS::RequestType.

See also:

base/protocols/socks/consts.zeek

CONNECTION = 1

PORT = 2

UDP_ASSOCIATE = 4

9.22 SSL Namespace

Namespace: SSL.

class zlogging.enum.SSL.**SctSource**(*value*)

Bases: IntFlag

Enum: SSL::SctSource.

List of the different sources for Signed Certificate Timestamp.

See also:

policy/protocols/ssl/validate-sct.zeek

SCT_X509_EXT = 1

Signed Certificate Timestamp was encountered in the extension of an X.509 certificate.

SCT_TLS_EXT = 2

Signed Certificate Timestamp was encountered in an TLS session extension.

SCT_OCSP_EXT = 4

Signed Certificate Timestamp was encountered in the extension of an stapled OCSP reply.

9.23 `Signatures` Namespace

Namespace: `Signatures`.

class zlogging.enum.Signatures.**Action**(*value*)

Bases: `IntFlag`

Enum: `Signatures::Action`.

These are the default actions you can apply to signature matches. All of them write the signature record to the logging stream unless declared otherwise.

See also:

base/frameworks/signatures/main.zeek

SIG_IGNORE = 1

Ignore this signature completely (even for scan detection). Don't write to the signatures logging stream.

SIG_QUIET = 2

Process through the various aggregate techniques, but don't report individually and don't write to the signatures logging stream.

SIG_LOG = 4

Generate a notice.

SIG_FILE_BUT_NO_SCAN = 8

The same as Signatures::SIG_LOG, but ignore for aggregate/scan processing.

SIG_ALARM = 16

Generate a notice and set it to be alarmed upon.

SIG_ALARM_PER_ORIG = 32

Alarm once per originator.

SIG_ALARM_ONCE = 64

Alarm once and then never again.

SIG_COUNT_PER_RESP = 128

Count signatures per responder host and alarm with the Signatures::Count_Signature notice if a threshold defined by Signatures::count_thresholds is reached.

SIG_SUMMARY = 256

Don't alarm, but generate per-orig summary.

9.24 `Software` Namespace

Namespace: `Software`.

class zlogging.enum.Software.**Type**(*value*)

Bases: `IntFlag`

Enum: `Software::Type`.

Scripts detecting new types of software need to redef this enum to add their own specific software types which would then be used when they create `Software::Info` records.

See also:

base/frameworks/software/main.zeek

UNKNOWN = 1

A placeholder type for when the type of software is not known.

OS_WINDOWS = 2

OS::WINDOWS (present if policy/frameworks/software/windows-version-detection.zeek is loaded) Identifier for Windows operating system versions

DHCP_SERVER = 4

DHCP::SERVER (present if policy/protocols/dhcp/software.zeek is loaded) Identifier for web servers in the software framework.

DHCP_CLIENT = 8

DHCP::CLIENT (present if policy/protocols/dhcp/software.zeek is loaded) Identifier for web browsers in the software framework.

FTP_CLIENT = 16

FTP::CLIENT (present if policy/protocols/ftp/software.zeek is loaded) Identifier for FTP clients in the software framework.

FTP_SERVER = 32

FTP::SERVER (present if policy/protocols/ftp/software.zeek is loaded) Not currently implemented.

HTTP_WEB_APPLICATION = 64

HTTP::WEB_APPLICATION (present if policy/protocols/http/detect-webapps.zeek is loaded) Identifier for web applications in the software framework.

HTTP_BROWSER_PLUGIN = 128

HTTP::BROWSER_PLUGIN (present if policy/protocols/http/software-browser-plugins.zeek is loaded) Identifier for browser plugins in the software framework.

HTTP_SERVER = 256

HTTP::SERVER (present if policy/protocols/http/software.zeek is loaded) Identifier for web servers in the software framework.

HTTP_APPSERVER = 512

HTTP::APPSERVER (present if policy/protocols/http/software.zeek is loaded) Identifier for app servers in the software framework.

HTTP_BROWSER = 1024

HTTP::BROWSER (present if policy/protocols/http/software.zeek is loaded) Identifier for web browsers in the software framework.

MySQL_SERVER = 2048

MySQL::SERVER (present if policy/protocols/mysql/software.zeek is loaded) Identifier for MySQL servers in the software framework.

SMTP_MAIL_CLIENT = 4096

SMTP::MAIL_CLIENT (present if policy/protocols/smtp/software.zeek is loaded)

SMTP_MAIL_SERVER = 8192

SMTP::MAIL_SERVER (present if policy/protocols/smtp/software.zeek is loaded)

SMTP_WEBMAIL_SERVER = 16384

SMTP::WEBMAIL_SERVER (present if policy/protocols/smtp/software.zeek is loaded)

SSH_SERVER = 32768

SSH::SERVER (present if policy/protocols/ssh/software.zeek is loaded) Identifier for SSH clients in the software framework.

SSH_CLIENT = 65536

SSH::CLIENT (present if policy/protocols/ssh/software.zeek is loaded) Identifier for SSH servers in the software framework.

9.25 SumStats Namespace

Namespace: SumStats.

class zlogging.enum.SumStats.**Calculation**(*value*)

Bases: IntFlag

Enum: SumStats::Calculation.

Type to represent the calculations that are available. The calculations are all defined as plugins.

See also:

base/frameworks/sumstats/main.zeek

PLACEHOLDER = 1

AVERAGE = 2

(present if base/frameworks/sumstats/plugins/average.zeek is loaded) Calculate the average of the values.

HLL_UNIQUE = 4

(present if base/frameworks/sumstats/plugins/hll_unique.zeek is loaded) Calculate the number of unique values.

LAST = 8

(present if base/frameworks/sumstats/plugins/last.zeek is loaded) Keep last X observations in a queue.

MAX = 16

(present if base/frameworks/sumstats/plugins/max.zeek is loaded) Find the maximum value.

MIN = 32

(present if base/frameworks/sumstats/plugins/min.zeek is loaded) Find the minimum value.

SAMPLE = 64

(present if base/frameworks/sumstats/plugins/sample.zeek is loaded) Get uniquely distributed random samples from the observation stream.

VARIANCE = 128

(present if base/frameworks/sumstats/plugins/variance.zeek is loaded) Calculate the variance of the values.

STD_DEV = 256

(present if base/frameworks/sumstats/plugins/std-dev.zeek is loaded) Calculate the standard deviation of the values.

SUM = 512

(present if base/frameworks/sumstats/plugins/sum.zeek is loaded) Calculate the sum of the values. For string values, this will be the number of strings.

TOPK = 1024

(present if base/frameworks/sumstats/plugins/topk.zeek is loaded) Keep a top-k list of values.

UNIQUE = 2048

(present if base/frameworks/sumstats/plugins/unique.zeek is loaded) Calculate the number of unique values.

9.26 Tunnel Namespace

Namespace: Tunnel.

class zlogging.enum.Tunnel.**Type**(*value*)

Bases: IntFlag

Enum: Tunnel::Type.

See also:

base/bif/types.bif.zeek

NONE = 1

IP = 2

AYIYA = 4

TEREDO = 8

SOCKS = 16

GTPv1 = 32

HTTP = 64

GRE = 128

VXLAN = 256

class zlogging.enum.Tunnel.**Action**(*value*)

Bases: IntFlag

Enum: Tunnel::Action.

Types of interesting activity that can occur with a tunnel.

See also:

base/frameworks/tunnels/main.zeek

DISCOVER = 1

A new tunnel (encapsulating "connection") has been seen.

CLOSE = 2

A tunnel connection has closed.

EXPIRE = 4

No new connections over a tunnel happened in the amount of time indicated by Tunnel::expiration_interval.

9.27 Weird Namespace

Namespace: Weird.

class zlogging.enum.Weird.**Action**(*value*)

Bases: IntFlag

Enum: Weird::Action.

Types of actions that may be taken when handling weird activity events.

See also:

base/frameworks/notice/weird.zeek

ACTION_UNSPECIFIED = 1

A dummy action indicating the user does not care what internal decision is made regarding a given type of weird.

ACTION_IGNORE = 2

No action is to be taken.

ACTION_LOG = 4

Log the weird event every time it occurs.

ACTION_LOG_ONCE = 8

Log the weird event only once.

ACTION_LOG_PER_CONN = 16

Log the weird event once per connection.

ACTION_LOG_PER_ORIG = 32

Log the weird event once per originator host.

ACTION_NOTICE = 64

Always generate a notice associated with the weird event.

ACTION_NOTICE_ONCE = 128

Generate a notice associated with the weird event only once.

ACTION_NOTICE_PER_CONN = 256

Generate a notice for the weird event once per connection.

ACTION_NOTICE_PER_ORIG = 512

Generate a notice for the weird event once per originator host.

9.28 ZeekygenExample Namespace

Namespace: ZeekygenExample.

class zlogging.enum.ZeekygenExample.**SimpleEnum**(*value*)

Bases: IntFlag

Enum: ZeekygenExample::SimpleEnum.

Documentation for the "SimpleEnum" type goes here. It can span multiple lines.

See also:

zeekygen/example.zeek

ONE = 1

Documentation for particular enum values is added like this. And can also span multiple lines.

TWO = 2

Or this style is valid to document the preceding enum value.

THREE = 4

FOUR = 8

And some documentation for "FOUR".

FIVE = 16

Also "FIVE".

zlogging.enum.**globals**(**namespaces*, *bare=False*)

Generate Bro/Zeek `enum` namespace.

Parameters

- ***namespaces** (*str*) – Namespaces to be loaded.
- **bare** (*bool*) – If `True`, do not load `zeek` namespace by default.

Returns

Global enum namespace.

Warns

BroDeprecationWarning – If `bro` namespace used.

Raises

ValueError – If `namespace` is not defined.

Return type

dict[str, Enum]

Note: For back-port compatibility, the `bro` namespace is an alias of the `zeek` namespace.

CHAPTER

TEN

INSTALLATION

Note: ZLogging supports Python all versions above and includes **3.6**.

```
pip install zlogging
```

CHAPTER

ELEVEN

USAGE

Currently ZLogging supports the two builtin formats as supported by the Bro/Zeek logging framework, i.e. ASCII and JSON.

A typical ASCII log file would be like:

```
#separator \x09
#set_separator	,
#empty_field	(empty)
#unset_field	-
#path	http
#open	2020-02-09-18-54-09
#fields	ts	uid	id.orig_h	id.orig_p	id.resp_h	id.resp_p	␣
↪	trans_depth	method	host	uri	referrer	version	user_agent	␣
↪origin	request_body_len	response_body_len	status_code	status_msg	␣
↪	info_code	info_msg	tags	username	password	proxied	orig_
↪fuids	orig_filenames	orig_mime_types	resp_fuids	resp_filenames	resp_mime_
↪types
#types	time	string	addr	port	addr	port	count	string	string	␣
↪string	string	string	string	string	count	count	count	string	count	string␣
↪	set[enum]	string	string	set[string]	vector[string]	vector[string]	␣
↪vector[string]	vector[string]	vector[string]	vector[string]
1581245648.761106	CSksID3S6ZxplpvmXg	192.168.2.108	56475	151.139.128.14	80	␣
↪	1	GET	ocsp.sectigo.com	/
↪MFYwVKADAgEAME0wSzBJMAkGBSsOAwIaBQAEFEML0g5PE3oabJGPJOXafjJNRzPIBBSNjF7EVK2K4Xfpm/
↪mbBeG4AY1h4QIQfdsAWJ+CXcbhDVFyNWosjQ==	-	1.1	com.apple.trustd/2.0	-␣
↪	0	471	200	OK	-	-	(empty)	-	-	-	-
↪	-	-	FPtlyEAhcf8orBPu7	-	application/ocsp-response
1581245651.379048	CuvUnl4HyhQbCs4tXe	192.168.2.108	56483	23.59.247.10	80	␣
↪	1	GET	isrg.trustid.ocsp.identrust.com	/
↪MFYwVKADAgEAME0wSzBJMAkGBSsOAwIaBQAEFG/0aE1DEtJIYoGcwCs9Rywdii+mBBTEp7Gkeyxx+tvhS5B1/
↪8QVYIWJEAIQCgFBQgAAAVOFc2oLheynCA==	-	1.1	com.apple.trustd/2.0	-	␣
↪	0	1398	200	OK	-	-	(empty)	-	-	-	-	␣
↪	-	-	FRfFoq3hSZkdCNDf9l	-	application/ocsp-response
1581245654.396334	CWo4pd1z97XLB2o0h2	192.168.2.108	56486	23.59.247.122	80	␣
↪	1	GET	isrg.trustid.ocsp.identrust.com	/
↪MFYwVKADAgEAME0wSzBJMAkGBSsOAwIaBQAEFG/0aE1DEtJIYoGcwCs9Rywdii+mBBTEp7Gkeyxx+tvhS5B1/
↪8QVYIWJEAIQCgFBQgAAAVOFc2oLheynCA==	-	1.1	com.apple.trustd/2.0	-	␣
↪	0	1398	200	OK	-	-	(empty)	-	-	-	-	␣
↪	-	-	FvQehf1pRsGmwDUzJe	-	application/ocsp-response
1581245692.728840	CxFQzh2ePtsnQhFNX3	192.168.2.108	56527	23.59.247.10	80	␣
↪	1	GET	isrg.trustid.ocsp.identrust.com	/
↪MFYwVKADAgEAME0wSzBJMAkGBSsOAwIaBQAEFG/0aE1DEtJIYoGcwCs9Rywdii+mBBTEp7Gkeyxx+tvhS5B1/
↪8QVYIWJEAIQCgFBQgAAAVOFc2oLheynCA==	-	1.1	com.apple.trustd/2.0	-	␣
↪	0	1398	200	OK	-	-	(empty)	-	-	-	-	␣
↪	-	-	FIeFj8WWNyhA1psGg	-	application/ocsp-response
```

(continues on next page)

(continued from previous page)

```
1581245701.693971    CPZSNk1Y6kDvAN0KZ8      192.168.2.108   56534   23.59.247.122   80
↪    1       GET     isrg.trustid.ocsp.identrust.com /
↪MFYwVKADAgEAME0wSzBJMAkGBSsOAwIaBQAEFG/0aE1DEtJIYoGcwCs9Rywdii+mBBTEp7Gkeyxx+tvhS5B1/
↪8QVYIWJEAIQCgFBQgAAAVOFc2oLheynCA==       -       1.1     com.apple.trustd/2.0    -
↪   0       1398    200     OK      -       -       (empty) -       -       -       -
↪    -       -       F0fGHe4RPuNBhYWNv6      -       application/ocsp-response
1581245707.848088    Cnab6CHFOprdppKi5       192.168.2.108   56542   23.59.247.122   80
↪    1       GET     isrg.trustid.ocsp.identrust.com /
↪MFYwVKADAgEAME0wSzBJMAkGBSsOAwIaBQAEFG/0aE1DEtJIYoGcwCs9Rywdii+mBBTEp7Gkeyxx+tvhS5B1/
↪8QVYIWJEAIQCgFBQgAAAVOFc2oLheynCA==       -       1.1     com.apple.trustd/2.0    -
↪   0       1398    200     OK      -       -       (empty) -       -       -       -
↪    -       -       FgDBep1h7EPHC8qQB6      -       application/ocsp-response
1581245952.784242    CPNd6t3ofePpdNjErl      192.168.2.108   56821   176.31.225.118  80
↪    1       GET     tracker.trackerfix.com  /announce?info_hash=y\x82es"\x1dV\xde|m\xbe
↪"\xe5\xef\xbe\x04\xb3\x1fW\xfc&peer_id=-qB4210-0ZOn5Ifyl*WF&port=63108&uploaded=0&
↪downloaded=0&left=3225455594&corrupt=0&key=6B23B036&event=started&numwant=200&
↪compact=1&no_peer_id=1&supportcrypto=1&redundant=0   -       1.1     -       -       0
↪       0       307     Temporary Redirect      -       -       (empty) -       -       -
↪       -       -       -       -       -       -
1581245960.123295    CfAkwf2CFI13b24gqf      192.168.2.108   56889   176.31.225.118  80
↪    1       GET     tracker.trackerfix.com  /announce?info_hash=!u7\xdad\x94x\xecS\x80\
↪x89\x04\x9c\x13#\x84M\x1b\xcd\x1a&peer_id=-qB4210-i36iloGe*QT9&port=63108&uploaded=0&
↪downloaded=0&left=1637966572&corrupt=0&key=ECE6637E&event=started&numwant=200&
↪compact=1&no_peer_id=1&supportcrypto=1&redundant=0       -       1.1     -       -
↪   0       0       307     Temporary Redirect      -       -       (empty) -       -
↪    -       -       -       -       -       -       -
#close       2020-02-09-19-01-40
```

Its corresponding JSON log file would be like:

```
{"ts": 1581245648.761106, "uid": "CSksID3S6ZxplpvmXg", "id.orig_h": "192.168.2.108", "id.
↪orig_p": 56475, "id.resp_h": "151.139.128.14", "id.resp_p": 80, "trans_depth": 1,
↪"method": "GET", "host": "ocsp.sectigo.com", "uri": "/
↪MFYwVKADAgEAME0wSzBJMAkGBSsOAwIaBQAEFEML0g5PE3oabJGPJOXafjJNRzPIBBSNjF7EVK2K4Xfpm/
↪mbBeG4AY1h4QIQfdsAWJ+CXcbhDVFyNWosjQ==", "referrer": "-", "version": "1.1", "user_agent
↪": "com.apple.trustd/2.0", "origin": "-", "request_body_len": 0, "response_body_len":
↪471, "status_code": 200, "status_msg": "OK", "info_code": null, "info_msg": "-", "tags
↪": [], "username": "-", "password": "-", "proxied": null, "orig_fuids": null, "orig_
↪filenames": null, "orig_mime_types": null, "resp_fuids": ["FPtlyEAhcf8orBPu7"], "resp_
↪filenames": null, "resp_mime_types": ["application/ocsp-response"]}
{"ts": 1581245651.379048, "uid": "CuvUnl4HyhQbCs4tXe", "id.orig_h": "192.168.2.108", "id.
↪orig_p": 56483, "id.resp_h": "23.59.247.10", "id.resp_p": 80, "trans_depth": 1, "method
↪": "GET", "host": "isrg.trustid.ocsp.identrust.com", "uri": "/
↪MFYwVKADAgEAME0wSzBJMAkGBSsOAwIaBQAEFG/0aE1DEtJIYoGcwCs9Rywdii+mBBTEp7Gkeyxx+tvhS5B1/
↪8QVYIWJEAIQCgFBQgAAAVOFc2oLheynCA==", "referrer": "-", "version": "1.1", "user_agent":
↪"com.apple.trustd/2.0", "origin": "-", "request_body_len": 0, "response_body_len":
↪1398, "status_code": 200, "status_msg": "OK", "info_code": null, "info_msg": "-", "tags
↪": [], "username": "-", "password": "-", "proxied": null, "orig_fuids": null, "orig_
↪filenames": null, "orig_mime_types": null, "resp_fuids": ["FRfFoq3hSZkdCNDf91"], "resp_
↪filenames": null, "resp_mime_types": ["application/ocsp-response"]}
{"ts": 1581245654.396334, "uid": "CWo4pd1z97XLB2o0h2", "id.orig_h": "192.168.2.108", "id.
↪orig_p": 56486, "id.resp_h": "23.59.247.122", "id.resp_p": 80, "trans_depth": 1,
↪"method": "GET", "host": "isrg.trustid.ocsp.identrust.com", "uri": "/
↪MFYwVKADAgEAME0wSzBJMAkGBSsOAwIaBQAEFG/0aE1DEtJIYoGcwCs9Rywdii+mBBTEp7Gkeyxx+tvhS5B1/
↪8QVYIWJEAIQCgFBQgAAAVOFc2oLheynCA==", "referrer": "-", "version": "1.1", "user_agent":
↪"com.apple.trustd/2.0", "origin": "-", "request_body_len": 0, "response_body_len":
↪1398, "status_code": 200, "status_msg": "OK", "info_code": null, "info_msg": "-", "tags
↪": [], "username": "-", "password": "-", "proxied": null, "orig_fuids": null, "orig_
↪filenames": null, "orig_mime_types": null, "resp_fuids": ["FvQehf1pRsGmwDUzJe"], "resp_
```

(continues on next page)

(continued from previous page)


```
{"ts": 1581245692.72884, "uid": "CxFQzh2ePtsnQhFNX3", "id.orig_h": "192.168.2.108", "id.
↪orig_p": 56527, "id.resp_h": "23.59.247.10", "id.resp_p": 80, "trans_depth": 1, "method
↪": "GET", "host": "isrg.trustid.ocsp.identrust.com", "uri": "/
↪MFYwVKADAgEAME0wSzBJMAkGBSsOAwIaBQAEFG/0aE1DEtJIYoGcwCs9Rywdii+mBBTEp7Gkeyxx+tvhS5B1/
↪8QVYIWJEAIQCgFBQgAAAVOFc2oLheynCA==", "referrer": "-", "version": "1.1", "user_agent":
↪"com.apple.trustd/2.0", "origin": "-", "request_body_len": 0, "response_body_len":␣
↪1398, "status_code": 200, "status_msg": "OK", "info_code": null, "info_msg": "-", "tags
↪": [], "username": "-", "password": "-", "proxied": null, "orig_fuids": null, "orig_
↪filenames": null, "orig_mime_types": null, "resp_fuids": ["FIeFj8WWNyhA1psGg"], "resp_
↪filenames": null, "resp_mime_types": ["application/ocsp-response"]}
{"ts": 1581245701.693971, "uid": "CPZSNk1Y6kDvAN0KZ8", "id.orig_h": "192.168.2.108", "id.
↪orig_p": 56534, "id.resp_h": "23.59.247.122", "id.resp_p": 80, "trans_depth": 1,
↪"method": "GET", "host": "isrg.trustid.ocsp.identrust.com", "uri": "/
↪MFYwVKADAgEAME0wSzBJMAkGBSsOAwIaBQAEFG/0aE1DEtJIYoGcwCs9Rywdii+mBBTEp7Gkeyxx+tvhS5B1/
↪8QVYIWJEAIQCgFBQgAAAVOFc2oLheynCA==", "referrer": "-", "version": "1.1", "user_agent":
↪"com.apple.trustd/2.0", "origin": "-", "request_body_len": 0, "response_body_len":␣
↪1398, "status_code": 200, "status_msg": "OK", "info_code": null, "info_msg": "-", "tags
↪": [], "username": "-", "password": "-", "proxied": null, "orig_fuids": null, "orig_
↪filenames": null, "orig_mime_types": null, "resp_fuids": ["F0fGHe4RPuNBhYWNv6"], "resp_
↪filenames": null, "resp_mime_types": ["application/ocsp-response"]}
{"ts": 1581245707.848088, "uid": "Cnab6CHFOprdppKi5", "id.orig_h": "192.168.2.108", "id.
↪orig_p": 56542, "id.resp_h": "23.59.247.122", "id.resp_p": 80, "trans_depth": 1,
↪"method": "GET", "host": "isrg.trustid.ocsp.identrust.com", "uri": "/
↪MFYwVKADAgEAME0wSzBJMAkGBSsOAwIaBQAEFG/0aE1DEtJIYoGcwCs9Rywdii+mBBTEp7Gkeyxx+tvhS5B1/
↪8QVYIWJEAIQCgFBQgAAAVOFc2oLheynCA==", "referrer": "-", "version": "1.1", "user_agent":
↪"com.apple.trustd/2.0", "origin": "-", "request_body_len": 0, "response_body_len":␣
↪1398, "status_code": 200, "status_msg": "OK", "info_code": null, "info_msg": "-", "tags
↪": [], "username": "-", "password": "-", "proxied": null, "orig_fuids": null, "orig_
↪filenames": null, "orig_mime_types": null, "resp_fuids": ["FgDBep1h7EPHC8qQB6"], "resp_
↪filenames": null, "resp_mime_types": ["application/ocsp-response"]}
{"ts": 1581245952.784242, "uid": "CPNd6t3ofePpdNjErl", "id.orig_h": "192.168.2.108", "id.
↪orig_p": 56821, "id.resp_h": "176.31.225.118", "id.resp_p": 80, "trans_depth": 1,
↪"method": "GET", "host": "tracker.trackerfix.com", "uri": "/announce?info_hash=y\\
↪x82es\"\\x1dV\\xde|m\\xbe\"\\xe5\\xef\\xbe\\x04\\xb3\\x1fW\\xfc&peer_id=-qB4210-
↪0ZOn5Ifyl*WF&port=63108&uploaded=0&downloaded=0&left=3225455594&corrupt=0&key=6B23B036&
↪event=started&numwant=200&compact=1&no_peer_id=1&supportcrypto=1&redundant=0",
↪"referrer": "-", "version": "1.1", "user_agent": "-", "origin": "-", "request_body_len
↪": 0, "response_body_len": 0, "status_code": 307, "status_msg": "Temporary Redirect",
↪"info_code": null, "info_msg": "-", "tags": [], "username": "-", "password": "-",
↪"proxied": null, "orig_fuids": null, "orig_filenames": null, "orig_mime_types": null,
↪"resp_fuids": null, "resp_filenames": null, "resp_mime_types": null}
{"ts": 1581245960.123295, "uid": "CfAkwf2CFI13b24gqf", "id.orig_h": "192.168.2.108", "id.
↪orig_p": 56889, "id.resp_h": "176.31.225.118", "id.resp_p": 80, "trans_depth": 1,
↪"method": "GET", "host": "tracker.trackerfix.com", "uri": "/announce?info_hash=!u7\\
↪xdad\\x94x\\xecS\\x80\\x89\\x04\\x9c\\x13#\\x84M\\x1b\\xcd\\x1a&peer_id=-qB4210-
↪i36iloGe*QT9&port=63108&uploaded=0&downloaded=0&left=1637966572&corrupt=0&key=ECE6637E&
↪event=started&numwant=200&compact=1&no_peer_id=1&supportcrypto=1&redundant=0",
↪"referrer": "-", "version": "1.1", "user_agent": "-", "origin": "-", "request_body_len
↪": 0, "response_body_len": 0, "status_code": 307, "status_msg": "Temporary Redirect",
↪"info_code": null, "info_msg": "-", "tags": [], "username": "-", "password": "-",
↪"proxied": null, "orig_fuids": null, "orig_filenames": null, "orig_mime_types": null,
↪"resp_fuids": null, "resp_filenames": null, "resp_mime_types": null}
```

11.1 How to Load/Parse a Log File?

To load (parse) a log file generically, i.e. when you don't know what format the log file is, you can simple call the *parse()*, *load()*, or *loads()* functions:

```
# to parse log at filename
>>> parse('path/to/log')
# to load log from a file object
>>> with open('path/to/log', 'rb') as file:
...     load(file)
# to load log from a string
>>> with open('/path/to/log', 'rb') as file:
...     loads(file.read())
```

Note: When calling *load()*, the file object must be opened in binary mode.

When calling *loads()*, if the `data` suplied is an encoded string (`str`), the function will first try to decode it as a bytestring (`bytes`) with `'ascii'` encoding.

If you do know the format, you may call the specified functions for each format, e.g. *parse_ascii()* and *parse_json()*, etc.

See also:

- *parse_ascii()*
- *parse_json()*
- *load_ascii()*
- *load_json()*
- *loads_ascii()*
- *loads_json()*

If you would like to customise your own parser, just subclass *BaseParser* and implement your own ideas.

11.2 How to Dump/Write a Log File?

Before dumping (writing) a log file, you need to create a log **data model** first. Just like in the Bro/Zeek script language, when customise logging, you need to notify the logging framework with a new log stream. Here, in `ZLogging`, we introduced **data model** for the same purpose.

A **data model** is a subclass of *Model* with fields and data types declared. A typical **data model** can be as following:

```
class MyLog(Model):
    field_one = StringType()
    field_two = SetType(element_type=PortType)
```

where `field_one` is `string` type, i.e. *StringType*; and `field_two` is `set[port]` types, i.e. *SetType* of *PortType*.

Or you may use type annotations as **PEP 484** introduced when declaring **data models**. All available type hints can be found in `zlogging.typing`:

```
class MyLog(Model):
    field_one: zeek_string
    field_two: zeek_set[zeek_port]
```

See also:

See *BaseType* and *Model* for more information about the data types and data model.

After declaration of your **data model**, you can know dump (write) your log file with the corresponding functions.

See also:

- *write_ascii()*
- *write_json()*
- *dump_ascii()*
- *dump_json()*
- *dumps_ascii()*
- *dumps_json()*

If you would like to customise your own writer, just subclass `BaseWriter` and implement your own ideas.

CHAPTER

TWELVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

Z

INDEX

Symbols

A

B

C

D

E

F

G

I

J

L

M

N

O

P

R

S

T

U

V

W

Z