idstools Documentation

Release 0.6.4

Jason Ish

Aug 22, 2023

Contents

idstools is a Python library for working with SNORT(R) and Suricata logs and rules.

CHAPTER 1

Contents

1.1 Tools

1.1.1 rulecat - A Suricata rule update tool

Synopsis

`idstools-rulecat` [OPTIONS]

Description

`idstools-ruleset` aims to be a simple to use rule download and management tool for Suricata. It can also be used for Snort when no SO rule stub generation is required.

Options

-h, --help
Show help.

-v, --verbose
Be more verbose.

-t <directory>, **--temp-dir**=<directory>
Temporary working directory (default: /var/tmp/idstools-rulecat).

This is where downloaded files will be stored.

--suricata=<path>
The path to the Suricata program used to determine which version of the ET pro rules to download if not explicitly set in a `--url` argument.

--suricata-version <version>
Set the Suricata version to a specific version instead of checking the version of Suricata on the path.

--force
Force remote rule files to be downloaded if they otherwise wouldn't be due to just recently downloaded, or the remote checksum matching the cached copy.

-o, --output
The directory where rule individual rules files will be written to. One of `-o` or `--merged` is required.

--merged=<filename>
Write a single file containing all rules. This can be used in addition to `--output` or instead of `--output`.

--yaml-fragment=<filename.yaml>
Output a fragment of YAML containing the *rule-files* section will all downloaded rule files listed for inclusion in your *suricata.yaml*.

--url=<url>
A URL to download rules from. This option can be used multiple times.

--local=<filename or directory>
A path to a filename or directory of local rule files to include. May be specified multiple times and should not include files in the output path.

If the path is a directory all files ending in *.rules* will be loaded.

Wildcards are accepted but to avoid shell expansion the argument must be quoted, for example:

```
--local '/etc/suricata/custom-*.rules'
```

--sid-msg-map=<filename>
Output a v1 style sid-msg.map file.

--sid-msg-map-2=<filename>
Output a v2 style sid-msg.map file.

--disable=<disable.conf>
Specify the configuration file for disabling rules.

--enable=<enable.conf>
Specify the configuration file for enabling rules.

--modify=<modify.conf>
Specify the configuration file for rule modifications.

--drop=<drop.conf>
Specify the configuration file for rules to change to drop.

--ignore=<filename>
Filenames to ignore. This is a pattern that will be matched against the basename of a rule files.

This argument may be specified multiple times.

Default: *deleted.rules

Alternatively the **group** matcher may be used in the file passed to `--disable`.

--no-ignore
Disable the –ignore option. Most useful to disable the default ignore pattern without adding others.

--etopen
Download the ET open ruleset. This is the default if `--url` or `--etpro` are not provided.

If one of `etpro` or `--url` is also specified, this option will at the ET open URL to the list of remote ruleset to be downloaded.

--etpro=<code>
Download the ET pro ruleset using the provided code.

-q, --quiet
Run quietly. Only warning and error message will be displayed.

--dump-sample-configs
Output sample configuration files for the `--disable`, `--enable`, `--modify` and `--threshold-in` commands.

--threshold-in=<threshold.conf.in>
Specify the threshold.conf input template.

--threshold-out=<threshold.conf>
Specify the name of the processed threshold.conf to output.

--post-hook=<command>
A command to run after the rules have been updated; will not run if not change to the output files was made. For example:

```
--post-hook=sudo kill -USR2 $(cat /var/run/suricata.pid)
```

will tell Suricata to reload its rules.

-V, --version
Display the version of **idstools-rulecat**.

Examples

Download ET Open rules for the version of Suricata found on the path, saving the rules in /etc/suricata/rules:

```
idstools-rulecat -o /etc/suricata/rules
```

Download ET Pro rules for the version of Suricata found on the path, saving the rules in /etc/suricata/rules:

```
idstools-rulecat --etpro XXXXXXXXXXXXXXXX -o /etc/suricata/rules
```

Download ET open rules plus an additional rule files and save the rules in /etc/suricata/rules:

```
idstools-rulecat --etopen \
    --url https://sslbl.abuse.ch/blacklist/sslblacklist.rules \
    -o /etc/suricata/rules
```

Configuration File

Command line arguments can be put in a file, one per line and used as a configuration file. By default, idstools-rulecat will look for a file in the current directory named rulecat.conf.

Example configuration file:

```
--suricata=/usr/sbin/suricata
--merged=rules/merged.rules
--disable=disable.conf
--enable=enable.conf
--modify=modify.conf
--post-hook=sudo kill -USR2 $(cat /var/run/suricata.pid)
```

(continues on next page)

(continued from previous page)

```
--etpro=XXXXXXXXXXXXXXXX
--url=https://sslbl.abuse.ch/blacklist/sslblacklist.rules
```

If *rulecat.conf* is in the current directory it will be used just by calling `idstools-rulecat` with no arguments. Otherwise you can point *idstools-rulecat* at a configuration with the command `idstools-rulecat @/path/to/rulecat.conf`.

Example Configuration Files

Example Configuration to Enable Rules (–enable)

```
# idstools-rulecat - enable.conf

# Example of enabling a rule by signature ID (gid is optional).
# 1:2019401
# 2019401

# Example of enabling a rule by regular expression.
# - All regular expression matches are case insensitive.
# re:hearbleed
# re:MS(0[7-9]|10)-\d+
```

Example Configuration to Enable Disable (–disable)

```
# idstools - disable.conf

# Example of disabling a rule by signature ID (gid is optional).
# 1:2019401
# 2019401

# Example of disabling a rule by regular expression.
# - All regular expression matches are case insensitive.
# re:hearbleed
# re:MS(0[7-9]|10)-\d+
```

Example Configuration to convert Rules to Drop (–drop)

```
# idstools-rulecat - drop.conf
#
# Rules matching specifiers in this file will be converted to drop rules.
#
# Examples:
#
# 1:2019401
# 2019401
#
# re:heartbleed
# re:MS(0[7-9]|10)-\d+
```

Example Configuration to modify Rules (–modify)

```
# idstools-rulecat - modify.conf

# Format: <sid> "<from>" "<to>"

# Example changing the seconds for rule 2019401 to 3600.
#2019401 "seconds \d+" "seconds 3600"

# Change all trojan-activity rules to drop. Its better to setup a
# drop.conf for this, but this does show the use of back references.
#re:classtype:trojan-activity "(alert)(.*)" "drop\\2"

# For compatibility, most Oinkmaster modifysid lines should work as
# well.
#modifysid * "^drop(.*)noalert(.*)" | "alert${1}noalert${2}"
```

Source

idstools/scripts/rulecat.py

1.1.2 eve2pcap - Convert packets/payloads in eve logs to pcap

Convert packets in EVE logs to pcap.

eve2pcap will convert the packets or the payloads found in an eve log file to a pcap file.

Note that payload conversion requires Scapy, and will not recreate the original packets as the headers need to be built on the fly from the available information in the eve log.

Usage

```
usage: idstools-eve2pcap [-h] [-o <filename>] [--payload] [--dlt DLT]
                         filenames [filenames ...]

positional arguments:
  filenames

optional arguments:
  -h, --help     show this help message and exit
  -o <filename>  Output filename
  --payload      Convert payload instead of packet
  --dlt DLT
```

Source

idstools/scripts/eve2pcap.py

1.1.3 u2spewfoo - A python/idstools implementation of u2spewfoo

A python reimplementation of Snort's u2spewfoo.

Usage

```
usage: idstools-u2spewfoo [-h] [filename [filename ...]]

positional arguments:
  filename

optional arguments:
  -h, --help  show this help message and exit
```

Source

idstools/scripts/u2spewfoo.py

1.1.4 u2json - A unified2 to JSON converter

Read unified2 log files and output records as JSON.

Contents

Usage

```
usage: idstools-u2json [-h] [-C <classification.config>] [-S <msg-msg.map>]
                       [-G <gen-msg.map>] [--snort-conf <snort.conf>]
                       [--directory <spool directory>]
                       [--prefix <spool file prefix>] [--bookmark <filename>]
                       [--follow] [--delete] [--output <filename>] [--stdout]
                       [--sort-keys] [--verbose] [--packet-printable]
                       [--packet-hex] [--extra-printable]
                       [filenames [filenames ...]]

positional arguments:
  filenames

optional arguments:
  -h, --help            show this help message and exit
  -C <classification.config>
                        path to classification config
  -S <msg-msg.map>      path to sid-msg.map
  -G <gen-msg.map>      path to gen-msg.map
  --snort-conf <snort.conf>
```

(continues on next page)

(continued from previous page)

```
                        attempt to load classifications and map files based on
                        the location of the snort.conf
  --directory <spool directory>
                        spool directory (eg: /var/log/snort)
  --prefix <spool file prefix>
                        spool filename prefix (eg: unified2.log)
  --bookmark <filename>
                        enable bookmarking
  --follow              follow files/continuous mode (spool mode only)
  --delete              delete spool files
  --output <filename>   output filename (eg: /var/log/snort/alerts.json
  --stdout              also log to stdout if --output is a file
  --sort-keys           the output of dictionaries will be sorted by key
  --verbose             be more verbose
  --packet-printable    output printable packet data in addition to base64
  --packet-hex          output packet data as hex in addition to base64
  --extra-printable     output printable extra data in addition to base64

If --directory and --prefix are provided files will be read from the specified
'spool' directory. Otherwise files on the command line will be processed.
```

Example - View unified2 File as JSON

```
idstools-u2json /var/log/snort/unified2.log.1397575268
```

To resolve alert descriptions and classifications:

```
idstools-u2json --snort-conf /etc/snort/etc/snort.conf \
    /var/log/snort/unified2.log.1397575268
```

The above assumes that sid-msg.map, gen-msg.map and classification.config live alongside the specified snort.conf. If they do not, the options to specify each individually may be used:

```
idstools-u2json -C /etc/snort/etc/classification.config \
    -S /etc/snort/etc/sid-msg.map \
    -G /etc/snort/etc/gen-msg.map \
    /var/log/snort/unified2.log.1397575268
```

Example - Continuous Conversion to JSON

```
idstools-u2json --snort.conf /etc/snort/etc/snort.conf \
    --directory /var/log/snort \
    --prefix unified2.log \
    --follow \
    --bookmark \
    --delete \
    --output /var/log/snort/alerts.json \
```

The above command will operate like barnyard, reading all unified2.log files in /var/log/snort, waiting for new unified2 records when the end of the last file is reached.

Additionally the last read location will be bookmarked to avoid reading events multiple times, the unified2.log files will be deleted once converted to JSON, and JSON events will be written to /var/log/snort/alerts.json.

Configuration File

A configuration file is simply a file containing the command line arguments, one per line with an '=' separating the name from the argument. For example, to save the arguments used in example 2 above:

```
--snort-conf=/etc/snort/etc/snort.conf
--directory=/var/log/snort
--prefix=unified2.log
--follow
--bookmark
--delete
--output=/var/log/snort/alerts.json
```

Then call idstools-u2json like:

```
idstools-u2json @/path/to/config-file
```

Addtional arguments can also be provided like:

```
idstools-u2json @/path/to/config-file --stdout
```

Source

idstools/scripts/u2json.py

1.1.5 u2fast - Unified2 to fast style printer

Read unified2 log files and output events in "fast" style.

Usage

```
usage: idstools-u2fast [-h] [-C <classification.config>] [-S <msg-msg.map>]
                       [-G <gen-msg.map>] [--snort-conf <snort.conf>]
                       [--directory <spool directory>]
                       [--prefix <spool file prefix>] [--bookmark] [--follow]
                       [filenames [filenames ...]]

positional arguments:
  filenames

optional arguments:
  -h, --help            show this help message and exit
  -C <classification.config>
                        path to classification config
  -S <msg-msg.map>      path to sid-msg.map
  -G <gen-msg.map>      path to gen-msg.map
  --snort-conf <snort.conf>
                        attempt to load classifications and map files based on
                        the location of the snort.conf
  --directory <spool directory>
                        spool directory (eg: /var/log/snort)
  --prefix <spool file prefix>
                        spool filename prefix (eg: unified2.log)
```

(continues on next page)

(continued from previous page)

```
  --bookmark            enable bookmarking
  --follow              follow files/continuous mode (spool mode only)
```

Source

idstools/scripts/u2fast.py

1.1.6 u2eve - Unified2 to Suricata eve events

Read unified2 log files and output events as Suricata EVE JSON (or as close as possible).

Usage

```
usage: idstools-u2eve [-h] [-C <classification.config>] [-S <msg-msg.map>]
                      [-G <gen-msg.map>] [--snort-conf <snort.conf>]
                      [--directory <spool directory>]
                      [--prefix <spool file prefix>] [--bookmark] [--follow]
                      [--delete] [-o <filename>] [--stdout]
                      [--packet-printable] [--packet-hex]
                      [filenames [filenames ...]]

positional arguments:
  filenames

optional arguments:
  -h, --help            show this help message and exit
  -C <classification.config>
                        path to classification config
  -S <msg-msg.map>      path to sid-msg.map
  -G <gen-msg.map>      path to gen-msg.map
  --snort-conf <snort.conf>
                        attempt to load classifications and map files based on
                        the location of the snort.conf
  --directory <spool directory>
                        spool directory (eg: /var/log/snort)
  --prefix <spool file prefix>
                        spool filename prefix (eg: unified2.log)
  --bookmark            enable bookmarking
  --follow              follow files/continuous mode (spool mode only)
  --delete              delete spool files
  -o <filename>, --output <filename>
                        output filename (eg: /var/log/snort/alerts.json
  --stdout              also log to stdout if --output is a file
  --packet-printable    add packet_printable field to events
  --packet-hex          add packet_hex field to events

If --directory and --prefix are provided files will be read from the specified
'spool' directory. Otherwise files on the command line will be processed.
```

Example - View a unified2 file as eve

```
idstools-u2eve -C path/to/classification.config \
  -S /path/to/sid-msg.map \
  -G /path/to/gen-msg.map merged.log.1431384519
```

Example - Continuous conversion to eve

```
idstools-u2eve --snort-conf /etc/snort/etc/snort.conf \
    --directory /var/log/snort \
    --prefix unified2.log \
    --follow \
    --bookmark \
    --delete \
    --output /var/log/snort/alerts.json \
```

The above command will operate like barnyard, reading all unified2.log files in /var/log/snort, waiting for new unified2 records when the end of the last file is reached.

Additionally the last read location will be bookmarked to avoid reading events multiple times, the unified2.log files will be deleted once converted to JSON, and JSON events will be written to /var/log/snort/alerts.json.

Configuration File

A configuration file is simply a file containing the command line arguments, one per line with an '=' separating the name from the argument. For example, to save the arguments used in example 2 above:

```
--snort-conf=/etc/snort/etc/snort.conf
--directory=/var/log/snort
--prefix=unified2.log
--follow
--bookmark
--delete
--output=/var/log/snort/alerts.json
```

Then call idstools-u2eve like:

```
idstools-u2eve @/path/to/config-file
```

Addtional arguments can also be provided like:

```
idstools-u2eve @/path/to/config-file --stdout
```

Source

idstools/scripts/u2eve.py

1.1.7 gensidmsgmap - sid-msg.map generator

Generate sid-msg.map files (v1 and v2) from rule archives, files and/or directories.

Usage

```
usage: ../bin/idstools-gensidmsgmap [options] <file>...

options:

    -2, --v2      Output a new (v2) style sid-msg.map file.

The files passed on the command line can be a list of a filenames, a
tarball, a directory name (containing rule files) or any combination
of the above.
```

Source

idstools/scripts/gensidmsgmap.py

1.1.8 dumpdynamicrules - Snort SO stub generator helper

Dump Snort SO rule stub helper program. Can optionally repack a Snort rule tarball with the generated stubs, in place or to a new file.

Usage

```
usage: idstools-dumpdynamicrules [-h] [--snort SNORT] [--version VERSION]
                                 [--dist DIST] [--out OUT]
                                 [--repack [filename]] [-v]
                                 <path>

positional arguments:
  <path>                SO rule directory or rule tarball

optional arguments:
  -h, --help            show this help message and exit
  --snort SNORT         path to snort
  --version VERSION     Snort version
  --dist DIST           operating system/distribution
  --out OUT             path to output SO stubs to
  --repack [filename]   repack archive with generated SO stubs
  -v, --verbose         log more information
```

Source

idstools/scripts/dumpdynamicrules.py

1.2 Library

1.2.1 Unified2 File Reading

idstools provides unified2 readers for reading unified2 records.

Contents

Reader Objects

RecordReader

class idstools.unified2.**RecordReader**(*fileobj*)

RecordReader reads and decodes unified2 records from a file-like object.

Parameters **fileobj** – The file-like object to read from.

Example:

```
fileobj = open("/var/log/snort/merged.log.1382627987", "rb")
reader = RecordReader(fileobj):
for record in reader:
    print(record)
```

next()

Return the next record or None if EOF.

Records returned will be one of the types *Event*, *Packet*, *ExtraData* or *Unknown* if the record is of an unknown type.

tell()

Get the current offset in the underlying file object.

FileRecordReader

class idstools.unified2.**FileRecordReader**(**files*)

FileRecordReader reads and decodes unified2 records from one or more files supplied by filename.

Parameters **files...** – One or more filenames to read records from.

Example:

```
reader = unified2.RecordReader("unified2.log.1382627941",
                               "unified2.log.1382627966)
for record in reader:
    print(record)
```

next()
Return the next record or None if EOF.

Records returned will be one of the types *Event*, *Packet*, *ExtraData* or *Unknown* if the record is of an unknown type.

tell()
Returns the current filename and offset.

SpoolRecordReader

class idstools.unified2.**SpoolRecordReader**(*directory*, *prefix*, *init_filename=None*, *init_offset=None*, *follow=False*, *rollover_hook=None*)
SpoolRecordReader reads and decodes records from a unified2 spool directory.

Required parameters:

Parameters

- **directory** – Path to unified2 spool directory.
- **prefix** – Filename prefix for unified2 log files.

Optional parameters:

Parameters

- **init_filename** – Filename open on initialization.
- **init_offset** – Offset to seek to on initialization.
- **follow** – Set to true if reading should wait for the next record to become available.
- **rollover_hook** – Function to call on rollover of log file, the first parameter being the filename being closed, the second being the filename being opened.

Example with following and rollover deletion:

```
def rollover_hook(closed, opened):
    os.unlink(closed)

reader = unified2.SpoolRecordReader("/var/log/snort",
    "unified2.log", rollover_hook = rollover_hook,
    follow = True)
for record in reader:
    print(record)
```

next()
Return the next decoded unified2 record from the spool directory.

tell()
Return a tuple containing the filename and offset of the file currently being processed.

Record Types

Event

class idstools.unified2.**Event**(*event*)
Event represents a unified2 event record with a dict-like interface. The unified2 file format specifies multiple

types of event records, idstools normalizes them into a single type.

Fields:

- sensor-id
- event-id
- event-second
- event-microsecond
- signature-id
- generator-id
- signature-revision
- classification-id
- priority
- source-ip
- destination-ip
- sport-itype
- dport-icode
- protocol
- impact-flag
- impact
- blocked
- mpls-label
- vlan-id

Deprecated: Methods that return events rather than single records will also populate the fields *packets* and *extra-data*. These fields are lists of the `Packet` and `ExtraData` records associated with the event.

Packet

class idstools.unified2.**Packet**(**fields*, ***kwargs*)

Packet represents a unified2 packet record with a dict-like interface.

Fields:

- sensor-id
- event-id
- event-second
- packet-second
- packet-microsecond
- linktype
- length
- data

ExtraData

class idstools.unified2.**ExtraData**(**fields*, ***kwargs*)

ExtraData represents a unified2 extra-data record with a dict like interface.

Fields:

- event-type
- event-length
- sensor-id
- event-id
- event-second
- type
- data-type
- data-length
- data

1.2.2 Rule Parsing

The **idstools** rule parsing can parse individual rule strings as well as multiple rules from a file or file like objects.

The Rule Object

The parsing functions will return one, or a list of Rule objects that present the rule as a dictionary.

class idstools.rule.**Rule**(*enabled=None*, *action=None*, *group=None*)

Class representing a rule.

The Rule class is a class that also acts like a dictionary.

Dictionary fields:

- **group**: The group the rule belongs to, typically the filename.
- **enabled**: True if rule is enabled (uncommented), False is disabled (commented)
- **action**: The action of the rule (alert, pass, etc) as a string
- **direction**: The direction string of the rule.
- **gid**: The gid of the rule as an integer
- **sid**: The sid of the rule as an integer
- **rev**: The revision of the rule as an integer
- **msg**: The rule message as a string
- **flowbits**: List of flowbit options in the rule
- **metadata**: Metadata values as a list
- **references**: References as a list
- **classtype**: The classification type
- **priority**: The rule priority, 0 if not provided

- **raw**: The raw rule as read from the file or buffer

  Parameters

  - **enabled** – Optional parameter to set the enabled state of the rule
  - **action** – Optional parameter to set the action of the rule
  - **group** – Optional parameter to set the group (filename) of the rule

---

Note: Parsed rules are primarily read only, with the exception of toggling the enabled state of the rule, modification is not really supported.

Parsing

`rule.`**`parse`**`(`*group=None*`)`
Parse a single rule for a string buffer.

Parameters **buf** – A string buffer containing a single Snort-like rule

Returns An instance of of *Rule* representing the parsed rule

`rule.`**`parse_fileobj`**`(`*group=None*`)`
Parse multiple rules from a file like object.

Note: At this time rules must exist on one line.

Parameters **fileobj** – A file like object to parse rules from.

Returns A list of *Rule* instances, one for each rule parsed

`rule.`**`parse_file`**`(`*group=None*`)`
Parse multiple rules from the provided filename.

Parameters **filename** – Name of file to parse rules from

Returns A list of *Rule* instances, one for each rule parsed

Printing

The string representation of the object will print the full rule respecting the enabled option of the rule.

For example:

```
>>> idstools.rule.parse('alert ip any any -> any any (msg:"TEST MESSAGE"; content:
↪"uid=0|28|root|29|"; classtype:bad-unknown; sid:10000000; rev:1;)')
>>> rule = idstools.rule.parse('alert ip any any -> any any (msg:"TEST MESSAGE";␣
↪content:"uid=0|28|root|29|"; classtype:bad-unknown; sid:10000000; rev:1;)')
>>> print(rule)
alert ip any any -> any any (msg:"TEST MESSAGE"; content:"uid=0|28|root|29|";␣
↪classtype:bad-unknown; sid:10000000; rev:1;)
>>> rule["enabled"] = False
>>> print(rule)
# alert ip any any -> any any (msg:"TEST MESSAGE"; content:"uid=0|28|root|29|";␣
↪classtype:bad-unknown; sid:10000000; rev:1;)
```

A brief description of the rule can be printed with *idstools.rule.Rule.brief()* or a string representing the rule ID can be printed with *idstools.rule.Rule.idstr()*.

Flowbit Resolution

The *idstools.rule.FlowbitResolver* is able to resolve the flowbits for a set of rules presented as a dictionary.

1.2.3 Maps

The maps module provides classes for mapping IDs to information objects such as signature IDs to a signature description.

Contents

SignatureMap

class idstools.maps.**SignatureMap**

SignatureMap maps signature IDs to a signature info dict.

The signature map can be build up from classification.config, gen-msg.map, and new and old-style sid-msg.map files.

The dict's in the map will have at a minimum the following fields:

- gid *(int)*
- sid *(int)*
- msg *(string)*
- refs *(list of strings)*

Signatures loaded from a new style sid-msg.map file will also have *rev*, *classification* and *priority* fields.

Example:

```
>>> from idstools import maps
>>> sigmap = maps.SignatureMap()
>>> sigmap.load_generator_map(open("tests/gen-msg.map"))
>>> sigmap.load_signature_map(open("tests/sid-msg-v2.map"))
>>> print(sigmap.get(1, 2495))
{'classification': 'misc-attack', 'rev': 8, 'priority': 0, 'gid': 1,
'sid': 2495,
'msg': 'GPL NETBIOS SMB DCEPRC ORPCThis request flood attempt',
'ref': ['bugtraq,8811', 'cve,2003-0813', 'nessus,12206',
'url,www.microsoft.com/technet/security/bulletin/MS04-011.mspx']}
```

get(*generator_id*, *signature_id*)

Get signature info by generator_id and signature_id.

Parameters

- **generator_id** – The generator id of the signature to lookup.
- **signature_id** – The signature id of the signature to lookup.

For convenience, if the generator_id is 3 and the signature is not found, a second lookup will be done using a generator_id of 1.

load_generator_map (*fileobj*)
Load the generator message map (gen-msg.map) from a file-like object.

load_signature_map (*fileobj*, *defaultgid=1*)
Load signature message map (sid-msg.map) from a file-like object.

ClassificationMap

class idstools.maps.**ClassificationMap** (*fileobj=None*)
ClassificationMap maps classification IDs and names to a dict object describing a classification.

Parameters **fileobj** – (Optional) A file like object to load classifications from on initialization.

The classification dicts stored in the map have the following fields:

- name *(string)*
- description *(string)*
- priority *(int)*

Example:

```
>>> from idstools import maps
>>> classmap = maps.ClassificationMap()
>>> classmap.load_from_file(open("tests/classification.config"))

>>> classmap.get(3)
{'priority': 2, 'name': 'bad-unknown', 'description': 'Potentially Bad Traffic'}
>>> classmap.get_by_name("bad-unknown")
{'priority': 2, 'name': 'bad-unknown', 'description': 'Potentially Bad Traffic'}
```

get (*class_id*)
Get a classification by ID.

Parameters **class_id** – The classification ID to get.

Returns A dict describing the classification or None.

get_by_name (*name*)
Get a classification by name.

Parameters **name** – The name of the classification

Returns A dict describing the classification or None.

load_from_file (*fileobj*)
Load classifications from a Snort style classification.config file object.

1.2.4 API - Auto Generated API Documentation

idstools

idstools package

Subpackages

idstools.rulecat package

Subpackages

idstools.rulecat.configs package

Module contents

Submodules

idstools.rulecat.loghandler module

class idstools.rulecat.loghandler.**SuriColourLogHandler**(*stream=None*)
Bases: logging.StreamHandler

An alternative stream log handler that logs with Suricata inspired log colours.

BLUE = '\x1b[34m'

GREEN = '\x1b[32m'

ORANGE = '\x1b[38;5;208m'

RED = '\x1b[31m'

REDB = '\x1b[1;31m'

RESET = '\x1b[0m'

YELLOW = '\x1b[33m'

YELLOWB = '\x1b[1;33m'

emit(*record*)
Emit a record.

If a formatter is specified, it is used to format the record. The record is then written to the stream with a trailing newline. If exception information is present, it is formatted using traceback.print_exception and appended to the stream. If the stream has an 'encoding' attribute, it is used to determine how to do the output to the stream.

formatTime(*record*)

Module contents

idstools.scripts package

Submodules

idstools.scripts.dumpdynamicrules module

Dump Snort SO rule stub helper program. Can optionally repack a Snort rule tarball with the generated stubs, in place or to a new file.

idstools.scripts.dumpdynamicrules.**find_snort**()
Find the path to Snort from the PATH.

idstools.scripts.dumpdynamicrules.**main**()

idstools.scripts.dumpdynamicrules.**mktempdir**(*delete_on_exit=True*)
Create a temporary directory that is removed on exit.

idstools.scripts.dumpdynamicrules.**repack**(*prefix*, *stubs*, *filename*)

idstools.scripts.eve2pcap module

Convert packets in EVE logs to pcap.

eve2pcap will convert the packets or the payloads found in an eve log file to a pcap file.

Note that payload conversion requires Scapy, and will not recreate the original packets as the headers need to be built on the fly from the available information in the eve log.

class idstools.scripts.eve2pcap.**Pcap**(*pcap_t*)

dump_fopen(*fileno*)
Not quite a direct wrapper around pcap_dump_fopen - instead of a file pointer, take a file descriptor.

dump_open(*filename*)

get_err()

classmethod open_dead(*linktype*, *snaplen*)

class idstools.scripts.eve2pcap.**PcapDumper**(*pcap_dumper_t*)
Minimal wrapper around pcap_dumper_t.

close()

dump(*pkthdr*, *packet*)

idstools.scripts.eve2pcap.**eve2pcap**(*event*)

idstools.scripts.eve2pcap.**main**()

idstools.scripts.eve2pcap.**parse_timestamp**(*timestamp*)

idstools.scripts.eve2pcap.**payload2packet**(*event*)

class idstools.scripts.eve2pcap.**pcap_pkthdr**
Bases: _ctypes.Structure

Internal class representing struct pcap_pkthdr.

caplen
Structure/Union member

pktlen
Structure/Union member

ts_sec
Structure/Union member

ts_usec
Structure/Union member

idstools.scripts.gensidmsgmap module

Generate sid-msg.map files (v1 and v2) from rule archives, files and/or directories.

idstools.scripts.gensidmsgmap.**file_iterator**(*files*)

idstools.scripts.gensidmsgmap.**main**()

idstools.scripts.gensidmsgmap.**usage**(*file=<open file '<stderr>', mode 'w'>*)

idstools.scripts.rulecat module

class idstools.scripts.rulecat.**AllRuleMatcher**
Bases: object

Matcher object to match all rules.

match(*rule*)

classmethod parse(*buf*)

class idstools.scripts.rulecat.**DropRuleFilter**(*matcher*)
Bases: object

Filter to modify an idstools rule object to a drop rule.

filter(*rule*)

is_noalert(*rule*)

match(*rule*)

class idstools.scripts.rulecat.**Fetch**(*args*)
Bases: object

check_checksum(*tmp_filename*, *url*)

extract_files(*filename*)

fetch(*url*)

get_tmp_filename(*url*)

progress_hook(*content_length*, *bytes_read*)

progress_hook_finish()

run()

url_basename(*url*)
Return the base filename of the URL.

class idstools.scripts.rulecat.**FileTracker**
Used to check if files are modified.

Usage: Add files with add(filename) prior to modification. Test with any_modified() which will return True if any of the checksums have been modified.

add(*filename*)

any_modified()

md5(*filename*)

class idstools.scripts.rulecat.**FilenameMatcher**(*filename*)
Bases: object

Matcher object to match a rule by its filename. This is similar to a group but has no specifier prefix.

match(*rule*)

classmethod parse(*buf*)

class idstools.scripts.rulecat.**GroupMatcher**(*pattern*)
Bases: object

Matcher object to match an idstools rule object by its group (ie: filename).

match(*rule*)

classmethod parse(*buf*)

class idstools.scripts.rulecat.**IdRuleMatcher**(*generatorId*, *signatureId*)
Bases: object

Matcher object to match an idstools rule object by its signature ID.

match(*rule*)

classmethod parse(*buf*)

class idstools.scripts.rulecat.**ModifyRuleFilter**(*matcher*, *pattern*, *repl*)
Bases: object

Filter to modify an idstools rule object.

Important note: This filter does not modify the rule inplace, but instead returns a new rule object with the modification.

filter(*rule*)

match(*rule*)

classmethod parse(*buf*)

class idstools.scripts.rulecat.**ReRuleMatcher**(*pattern*)
Bases: object

Matcher object to match an idstools rule object by regular expression.

match(*rule*)

classmethod parse(*buf*)

class idstools.scripts.rulecat.**ThresholdProcessor**

extract_pattern(*buf*)

extract_regex(*buf*)

patterns = [<_sre.SRE_Pattern object>, <_sre.SRE_Pattern object>, <_sre.SRE_Pattern ob

process(*filein*, *fileout*, *rulemap*)

replace(*threshold*, *rule*)

idstools.scripts.rulecat.**build_report**(*prev_rulemap*, *rulemap*)
Build a report of changes between 2 rulemaps.

Returns a dict with the following keys that each contain a list of rules. - added - removed - modified

idstools.scripts.rulecat.**build_rule_map**(*rules*)
Turn a list of rules into a mapping of rules.

In case of gid:sid conflict, the rule with the higher revision number will be used.

idstools.scripts.rulecat.**dump_sample_configs**()

idstools.scripts.rulecat.**ignore_file**(*ignore_files*, *filename*)

idstools.scripts.rulecat.**load_drop_filters**(*filename*)

idstools.scripts.rulecat.**load_filters**(*filename*)

idstools.scripts.rulecat.**load_local**(*local*, *files*)
Load local files into the files dict.

idstools.scripts.rulecat.**load_matchers**(*filename*)

idstools.scripts.rulecat.**main**()

idstools.scripts.rulecat.**parse_rule_match**(*match*)

idstools.scripts.rulecat.**resolve_etopen_url**(*suricata_version*)

idstools.scripts.rulecat.**resolve_etpro_url**(*etpro*, *suricata_version*)

idstools.scripts.rulecat.**resolve_flowbits**(*rulemap*, *disabled_rules*)

idstools.scripts.rulecat.**write_merged**(*filename*, *rulemap*)

idstools.scripts.rulecat.**write_sid_msg_map**(*filename*, *rulemap*, *version=1*)

idstools.scripts.rulecat.**write_to_directory**(*directory*, *files*, *rulemap*)

idstools.scripts.rulecat.**write_yaml_fragment**(*filename*, *files*)

idstools.scripts.rulemod module

idstools.scripts.rulemod.**main**()

idstools.scripts.rulemod.**match_all**(*matchers*, *rule*)

idstools.scripts.u2eve module

Read unified2 log files and output events as Suricata EVE JSON (or as close as possible).

class idstools.scripts.u2eve.**EveFilter**(*msgmap=None*, *classmap=None*, *packet_printable=False*, *packet_hex=False*)

Bases: object

filter(*event*)

format_event(*event*)

format_hex(*data*)

format_packet(*packet*)

getprotobynumber(*protocol*)

resolve_classification(*event*, *default=None*)

resolve_msg(*event*, *default=None*)

class idstools.scripts.u2eve.**OutputWrapper**(*filename*, *fileobj=None*)
Bases: object

reopen()

write(*buf*)

class idstools.scripts.u2eve.**RolloverHandler**(*delete*)
Bases: object

on_rollover(*closed*, *opened*)

class idstools.scripts.u2eve.**Writer**(*outputs*, *formatter*)

write(*event*)

idstools.scripts.u2eve.**calculate_flow_id**(*event*)

idstools.scripts.u2eve.**get_tzoffset**(*sec*)

idstools.scripts.u2eve.**load_from_snort_conf**(*snort_conf*, *classmap*, *msgmap*)

idstools.scripts.u2eve.**main**()

idstools.scripts.u2eve.**render_timestamp**(*sec*, *usec*)

idstools.scripts.u2fast module

Read unified2 log files and output events in "fast" style.

idstools.scripts.u2fast.**load_from_snort_conf**(*snort_conf*, *classmap*, *msgmap*)

idstools.scripts.u2fast.**main**()

idstools.scripts.u2fast.**print_event**(*event*, *msgmap*, *classmap*)

idstools.scripts.u2fast.**print_time**(*sec*, *usec*)

idstools.scripts.u2json module

Read unified2 log files and output records as JSON.

class idstools.scripts.u2json.**Formatter**(*msgmap=None*, *classmap=None*, *packet_printable=False*, *packet_hex=False*, *extra_printable=False*)
Bases: object

format(*record*)

format_event(*record*)

format_extra_data(*record*)

format_hex(*data*)

format_packet(*record*)

key(*key*)

resolve_classification(*event*, *default=None*)

resolve_msg(*event*, *default=None*)

class idstools.scripts.u2json.**OutputWrapper**(*filename*, *fileobj=None*)
Bases: object

reopen()

write(*buf*)

idstools.scripts.u2json.**load_from_snort_conf**(*snort_conf*, *classmap*, *msgmap*)

idstools.scripts.u2json.**main**()

idstools.scripts.u2json.**rollover_hook**(*closed*, *opened*)
The rollover hook for the spool reader. Will delete the closed file.

idstools.scripts.u2spewfoo module

A python reimplementation of Snort's u2spewfoo.

idstools.scripts.u2spewfoo.**main**()

idstools.scripts.u2spewfoo.**print_address**(*addr*)

idstools.scripts.u2spewfoo.**print_char**(*char*)

idstools.scripts.u2spewfoo.**print_event**(*event*)

idstools.scripts.u2spewfoo.**print_extra**(*extra*)

idstools.scripts.u2spewfoo.**print_packet**(*packet*)

idstools.scripts.u2spewfoo.**print_raw**(*raw*)

idstools.scripts.u2spewfoo.**print_record**(*record*)

idstools.scripts.u2spewfoo.**printable_chars**(*buf*)

Module contents

Submodules

idstools.maps module

Provide mappings from ID's to descriptions.

Includes mapping classes for event ID messages and classification information.

class idstools.maps.**ClassificationMap**(*fileobj=None*)
Bases: object

ClassificationMap maps classification IDs and names to a dict object describing a classification.

Parameters **fileobj** – (Optional) A file like object to load classifications from on initialization.

The classification dicts stored in the map have the following fields:

- name *(string)*
- description *(string)*
- priority *(int)*

Example:

```
>>> from idstools import maps
>>> classmap = maps.ClassificationMap()
>>> classmap.load_from_file(open("tests/classification.config"))

>>> classmap.get(3)
{'priority': 2, 'name': 'bad-unknown', 'description': 'Potentially Bad Traffic'}
>>> classmap.get_by_name("bad-unknown")
{'priority': 2, 'name': 'bad-unknown', 'description': 'Potentially Bad Traffic'}
```

add(*classification*)
Add a classification to the map.

get(*class_id*)
Get a classification by ID.

Parameters **class_id** – The classification ID to get.

Returns A dict describing the classification or None.

get_by_name(*name*)
Get a classification by name.

Parameters **name** – The name of the classification

Returns A dict describing the classification or None.

load_from_file(*fileobj*)
Load classifications from a Snort style classification.config file object.

size()

class idstools.maps.**SignatureMap**
Bases: object

SignatureMap maps signature IDs to a signature info dict.

The signature map can be build up from classification.config, gen-msg.map, and new and old-style sid-msg.map files.

The dict's in the map will have at a minimum the following fields:

- gid *(int)*
- sid *(int)*
- msg *(string)*
- refs *(list of strings)*

Signatures loaded from a new style sid-msg.map file will also have *rev*, *classification* and *priority* fields.

Example:

```
>>> from idstools import maps
>>> sigmap = maps.SignatureMap()
>>> sigmap.load_generator_map(open("tests/gen-msg.map"))
>>> sigmap.load_signature_map(open("tests/sid-msg-v2.map"))
>>> print(sigmap.get(1, 2495))
{'classification': 'misc-attack', 'rev': 8, 'priority': 0, 'gid': 1,
'sid': 2495,
'msg': 'GPL NETBIOS SMB DCEPRC ORPCThis request flood attempt',
'ref': ['bugtraq,8811', 'cve,2003-0813', 'nessus,12206',
'url,www.microsoft.com/technet/security/bulletin/MS04-011.mspx']}
```

get(*generator_id*, *signature_id*)
Get signature info by generator_id and signature_id.

Parameters

- **generator_id** – The generator id of the signature to lookup.
- **signature_id** – The signature id of the signature to lookup.

For convenience, if the generator_id is 3 and the signature is not found, a second lookup will be done using a generator_id of 1.

load_generator_map(*fileobj*)
Load the generator message map (gen-msg.map) from a file-like object.

load_signature_map(*fileobj*, *defaultgid=1*)
Load signature message map (sid-msg.map) from a file-like object.

size()

idstools.net module

Module for network related operations.

idstools.net.**get**(*url*, *fileobj*, *progress_hook=None*)
Perform a GET request against a URL writing the contents into the provideded file like object.

Parameters

- **url** – The URL to fetch
- **fileobj** – The fileobj to write the content to
- **progress_hook** – The function to call with progress updates

Returns Returns a tuple containing the number of bytes read and the result of the info() function from urllib2.urlopen().

Raises Exceptions from urllib2.urlopen() and writing to the provided fileobj may occur.

idstools.packet module

Provides basic packet decoding.

idstools.packet.**decode_ethernet**(*pkt*)
Decode an ethernet packet.

idstools.packet.**decode_icmp**(*pkt*)
Decode an ICMP packet.

idstools.packet.**decode_icmp6**(*pkt*)
Decode an ICMPv6 packet.

idstools.packet.**decode_ip**(*pkt*)
Decode an IP packet.

idstools.packet.**decode_ip6**(*pkt*)
Decode an IPv6 packet.

idstools.packet.**decode_tcp**(*pkt*)
Decode a TCP packet.

idstools.packet.**decode_udp**(*pkt*)
Decode a UDP packet.

idstools.packet.**printable_ethernet_addr**(*addr*)
Return a formatted ethernet address from its raw form.

idstools.rule module

Module for parsing Snort-like rules.

Parsing is done using regular expressions and the job of this module is to do its best at parsing out fields of interest from the rule rather than perform a sanity check.

The methods that parse multiple rules for a provided input (parse_file, parse_fileobj) return a list of rules instead of dict keyed by ID as its not the job of this module to detect or deal with duplicate signature IDs.

class idstools.rule.**FlowbitResolver**
Bases: object

get_required_flowbits(*rules*)

get_required_rules(*rulemap*, *flowbits*, *include_enabled=False*)
Returns a list of rules that need to be enabled in order to satisfy the list of required flowbits.

getters = ['isset', 'isnotset']

parse_flowbit(*flowbit*)

resolve(*rules*)

set_required_flowbits(*rules*, *required*)

setters = ['set', 'setx', 'unset', 'toggle']

class idstools.rule.**Rule**(*enabled=None*, *action=None*, *group=None*)
Bases: dict

Class representing a rule.

The Rule class is a class that also acts like a dictionary.

Dictionary fields:

- **group**: The group the rule belongs to, typically the filename.
- **enabled**: True if rule is enabled (uncommented), False is disabled (commented)
- **action**: The action of the rule (alert, pass, etc) as a string
- **direction**: The direction string of the rule.
- **gid**: The gid of the rule as an integer
- **sid**: The sid of the rule as an integer
- **rev**: The revision of the rule as an integer
- **msg**: The rule message as a string
- **flowbits**: List of flowbit options in the rule
- **metadata**: Metadata values as a list
- **references**: References as a list
- **classtype**: The classification type

- **priority**: The rule priority, 0 if not provided
- **raw**: The raw rule as read from the file or buffer

Parameters

- **enabled** – Optional parameter to set the enabled state of the rule
- **action** – Optional parameter to set the action of the rule
- **group** – Optional parameter to set the group (filename) of the rule

brief()
A brief description of the rule.

Returns A brief description of the rule

Return type string

format()

id
The ID of the rule.

Returns A tuple (gid, sid) representing the ID of the rule

Return type A tuple of 2 ints

idstr
Return the gid and sid of the rule as a string formatted like: '[GID:SID]'

rebuild_options()
Rebuild the rule options from the list of options.

idstools.rule.**add_option**(*rule*, *name*, *value*, *index=None*)

idstools.rule.**enable_flowbit_dependencies**(*rulemap*)
Helper function to resolve flowbits, wrapping the FlowbitResolver class.

idstools.rule.**find_opt_end**(*options*)
Find the end of an option (;) handling escapes.

idstools.rule.**format_sidmsgmap**(*rule*)
Format a rule as a sid-msg.map entry.

idstools.rule.**format_sidmsgmap_v2**(*rule*)
Format a rule as a v2 sid-msg.map entry.

eg: gid || sid || rev || classification || priority || msg || ref0 || refN

idstools.rule.**parse**(*buf*, *group=None*)
Parse a single rule for a string buffer.

Parameters **buf** – A string buffer containing a single Snort-like rule

Returns An instance of of *Rule* representing the parsed rule

idstools.rule.**parse_file**(*filename*, *group=None*)
Parse multiple rules from the provided filename.

Parameters **filename** – Name of file to parse rules from

Returns A list of *Rule* instances, one for each rule parsed

idstools.rule.**parse_fileobj**(*fileobj*, *group=None*)
Parse multiple rules from a file like object.

Note: At this time rules must exist on one line.

Parameters **fileobj** – A file like object to parse rules from.

Returns A list of *Rule* instances, one for each rule parsed

idstools.rule.**remove_option**(*rule*, *name*)

idstools.snort module

class idstools.snort.**SnortApp**(*config=None*, *path=None*, *os=None*, *dynamic_engine_lib=None*)
Bases: object

Snort represents the Snort application.

Parameters

- **config** – A dictionary configuration object. The dictionary can contain the same fields as the following parameters. Parameters take precedence over the config dictionary.
- **path** – The path to the Snort binary.

dump_dynamic_rules(*dynamic_detection_lib_dir*, *verbose=False*)

exists()

find_dynamic_detection_lib_dir(*prefix*)
Find the dynamic SO rule directory in prefix based on what we know about Snort.

get_arch()

set_dynamic_engine_lib(*dynamic_engine_lib*, *config*)

version()

idstools.suricata module

class idstools.suricata.**SuricataVersion**(*major*, *minor*, *patch*, *full*, *short*, *raw*)
Bases: tuple

full
Alias for field number 3

major
Alias for field number 0

minor
Alias for field number 1

patch
Alias for field number 2

raw
Alias for field number 5

short
Alias for field number 4

idstools.suricata.**get_path**(*program='suricata'*)
Find Suricata in the shell path.

idstools.suricata.**get_version**(*path=None*)
Get a SuricataVersion named tuple describing the version.

If no path argument is found, the envionment PATH will be searched.

idstools.suricata.**parse_version**(*buf*)

idstools.unified2 module

Unified2 record and event reading.

Unified2 is a file format used by the Snort and Suricata IDS engines for logging events.

For more information on the unified2 file format see:

http://manual.snort.org/node44.html

```
usage: from idstools import unified2
```

class idstools.unified2.**AbstractDecoder**(*fields*)
Bases: object

Base class for decoders.

class idstools.unified2.**Aggregator**
Bases: object

Deprecated: Applications requiring the aggregation of packets and extra data with an event should implement their own aggregation logic.

A class implementing something like the aggregator pattern to aggregate records until an event can be built.

add(*record*)
Add a new record to aggregator.

Parameters **record** – The decoded unified2 record to add.

Returns If adding a new record allows an event to be completed, an *Event* will be returned.

flush()
Flush the queue. This converts the records in the queue into an Event.

If using the Aggregator directly, you'll want to call flush after adding all your records to get the final event.

Returns An *Event* or None if there are no records.

class idstools.unified2.**Event**(*event*)
Bases: dict

Event represents a unified2 event record with a dict-like interface. The unified2 file format specifies multiple types of event records, idstools normalizes them into a single type.

Fields:

- sensor-id
- event-id
- event-second
- event-microsecond

- signature-id
- generator-id
- signature-revision
- classification-id
- priority
- source-ip
- destination-ip
- sport-itype
- dport-icode
- protocol
- impact-flag
- impact
- blocked
- mpls-label
- vlan-id

Deprecated: Methods that return events rather than single records will also populate the fields *packets* and *extra-data*. These fields are lists of the *Packet* and *ExtraData* records associated with the event.

class idstools.unified2.**EventDecoder**(*fields*)
Bases: *idstools.unified2.AbstractDecoder*

Decoder for event type records.

decode(*buf*)
Decodes a buffer into an *Event* object.

decode_ip(*addr*)

class idstools.unified2.**ExtraData**(**fields*, ***kwargs*)
Bases: dict

ExtraData represents a unified2 extra-data record with a dict like interface.

Fields:

- event-type
- event-length
- sensor-id
- event-id
- event-second
- type
- data-type
- data-length
- data

class idstools.unified2.**ExtraDataDecoder**(*fields*)
Bases: *idstools.unified2.AbstractDecoder*

Decoder for extra data type records.

decode(*buf*)
Decodes a buffer into an *ExtraData* object.

class idstools.unified2.**Field**(*name*, *length*, *fmt=None*)
Bases: object

A class to represent a field in a unified2 record. Used for building the decoders.

fmt
Builds a format string for struct.unpack.

class idstools.unified2.**FileEventReader**(**files*)
Bases: object

Deprecated: Event readers have been deprecated due to the deprecation of the Aggregator.

FileEventReader reads records from one or more filenames and aggregates them into events.

Parameters **files...** – One or more files to read events from.

Example:

```
reader = unified2.FileEventReader("unified2.log.1382627941",
                                  "unified2.log.1382627966)
for event in reader:
    print(event)
```

next()
Return the next *Event* or None if EOF.

class idstools.unified2.**FileRecordReader**(**files*)
Bases: object

FileRecordReader reads and decodes unified2 records from one or more files supplied by filename.

Parameters **files...** – One or more filenames to read records from.

Example:

```
reader = unified2.RecordReader("unified2.log.1382627941",
                               "unified2.log.1382627966)
for record in reader:
    print(record)
```

next()
Return the next record or None if EOF.

Records returned will be one of the types *Event*, *Packet*, *ExtraData* or *Unknown* if the record is of an unknown type.

tell()
Returns the current filename and offset.

class idstools.unified2.**Packet**(**fields*, ***kwargs*)
Bases: dict

Packet represents a unified2 packet record with a dict-like interface.

Fields:

- sensor-id
- event-id
- event-second
- packet-second
- packet-microsecond
- linktype
- length
- data

class idstools.unified2.**PacketDecoder**(*fields*)
Bases: *idstools.unified2.AbstractDecoder*

Decoder for packet type records.

decode(*buf*)
Decodes a buffer into a *Packet* object.

class idstools.unified2.**RecordReader**(*fileobj*)
Bases: object

RecordReader reads and decodes unified2 records from a file-like object.

Parameters **fileobj** – The file-like object to read from.

Example:

```
fileobj = open("/var/log/snort/merged.log.1382627987", "rb")
reader = RecordReader(fileobj):
for record in reader:
    print(record)
```

next()
Return the next record or None if EOF.

Records returned will be one of the types *Event*, *Packet*, *ExtraData* or *Unknown* if the record is of an unknown type.

tell()
Get the current offset in the underlying file object.

class idstools.unified2.**SpoolEventReader**(*directory*, *prefix*, *follow=False*, *delete=False*, *bookmark=False*)
Bases: object

Deprecated: Event readers have been deprecated due to the deprecation of the Aggregator.

SpoolEventReader reads records from a unified2 spool directory and aggregates them into events.

Required parameters:

Parameters

- **directory** – Path to unified2 spool directory.
- **prefix** – Filename prefix for unified2 log files.

Optional parameters:

Parameters

- **follow** – Set to true to follow the log files. Reading will wait until an event is available before returning.
- **delete** – If True, unified2 files will be deleted when reading has moved onto the next one.
- **bookmark** – If True, the reader will remember its location and start reading from the bookmarked location on initialization.

Example:

```
reader = unified2.SpoolEventReader("/var/log/snort", "unified2.log")
for event in reader:
    print(event)
```

next()
Return the next *Event*.

If in follow mode and EOF is head, this method will sleep and and try again.

rollover_hook(*closed*, *opened*)

tell()
See SpoolRecordReader.tell().

class idstools.unified2.**SpoolRecordReader**(*directory*, *prefix*, *init_filename=None*, *init_offset=None*, *follow=False*, *rollover_hook=None*)

Bases: object

SpoolRecordReader reads and decodes records from a unified2 spool directory.

Required parameters:

Parameters

- **directory** – Path to unified2 spool directory.
- **prefix** – Filename prefix for unified2 log files.

Optional parameters:

Parameters

- **init_filename** – Filename open on initialization.
- **init_offset** – Offset to seek to on initialization.
- **follow** – Set to true if reading should wait for the next record to become available.
- **rollover_hook** – Function to call on rollover of log file, the first parameter being the filename being closed, the second being the filename being opened.

Example with following and rollover deletion:

```
def rollover_hook(closed, opened):
    os.unlink(closed)

reader = unified2.SpoolRecordReader("/var/log/snort",
    "unified2.log", rollover_hook = rollover_hook,
    follow = True)
for record in reader:
    print(record)
```

get_filenames()
Return the filenames (sorted) from the spool directory.

iter_next()
Return the next record or None if EOF.

If in follow mode and EOF, this method will sleep and and try again.

Returns A record of type *Event*, *Packet*, *ExtraData* or *Unknown* if the record is of an unknown type.

next()
Return the next decoded unified2 record from the spool directory.

open_file(*filename*)

open_next()
Open the next available file. If a new file is opened its filename will be returned, otherwise None will be returned.

tell()
Return a tuple containing the filename and offset of the file currently being processed.

class idstools.unified2.**Unified2Bookmark**(*directory=None*, *prefix=None*, *filename=None*)
Bases: object

Class to represent a "bookmark" for unified2 spool directories.

get()
Get the current bookmark.

Returns a tuple of filename and offset.

update(*filename*, *offset*)
Update the bookmark with the given filename and offset.

class idstools.unified2.**Unknown**(*record_type*, *buf*)
Bases: object

Class to represent an unknown record type.

In the unlikely case that a record is of an unknown type, an instance of *Unknown* will be used to hold the record type and buffer.

exception idstools.unified2.**UnknownRecordType**(*record_type*)
Bases: exceptions.Exception

idstools.unified2.**decode_record**(*record_type*, *buf*)
Decodes a raw record into an object representing the record.

Parameters

- **record_type** – The type of record.
- **buf** – Buffer containing the raw record.

Returns The decoded record as a *Event*, *Packet*, *ExtraData* or *Unknown* if the record is of an unknown type.

idstools.unified2.**read_record**(*fileobj*)
Reads a unified2 record from the provided file object.

Parameters **fileobj** – The file like object to read from. Currently this object needs to support read, seek and tell.

Returns If a complete record is read a Record will be returned, otherwise None will be returned.

If some data is read, but not enough for a whole record, the location of the file object will be reset and a EOFError exception will be raised.

idstools.util module

Module for utility functions that don't really fit anywhere else.

idstools.util.**decode_inet_addr**(*addr*)

idstools.util.**format_printable**(*data*)
Given a buffer, return a string with the printable characters. A '.' will be used for all non-printable characters.

idstools.util.**md5_hexdigest**(*filename*)
Compute the MD5 checksum for the contents of the provided filename.

Parameters **filename** – Filename to computer MD5 checksum of.

Returns A string representing the hex value of the computed MD5.

idstools.util.**mktempdir**(*delete_on_exit=True*)
Create a temporary directory that is removed on exit.

Module contents

CHAPTER 2

Indices and Tables

- genindex
- modindex
- search

Python Module Index

i

Index

Symbols

A

B

C

D

E

F

G

I

K

L

M

N

O

P

R

S

T